Universidade do Minho
Escola de Engenharia

Jorge Alexandre de Albuquerque Loureiro

Restruturação Dinâmica de Estruturas Multidimensionais de Dados em Tempo Útil

Janeiro de 2007

Restruturação Dinâmica de Estruturas Multidimensionais de Dados em Tempo Útil
Jorge Alexandre de Albuquerque Loureiro
UMinho | 2007

Universidade do Minho
Escola de Engenharia

Jorge Alexandre de Albuquerque Loureiro

Restruturação Dinâmica de Estruturas Multidimensionais de Dados em Tempo Útil

Dissertação de Doutoramento em
Informática na especialidade de Inteligência Artificial

Trabalho efectuado sob a orientação do
Professor Doutor Orlando Manuel de Oliveira Belo

Investigação Subsidiada por

Janeiro de 2007

À memória de meus Pais

Agradecimentos

Muitos foram aqueles que colocaram um empenho e dedicação especial e deram a sua contribuição e apoio para a realização de um trabalho desta natureza. Na impossibilidade de enumerar todos, parece justo particularizar aqueles que merecem uma menção especial, mas a todos endereço, desde já, o meu sentido bem-hajam. Também algumas instituições merecem destaque, pois que o seu apoio material e disponibilidade tornaram possível o desenvolvimento deste trabalho tão absorvente.

Ao Professor Orlando Belo, orientador científico e metodológico desta investigação, não só pelo seu apoio, incentivo, entusiasmo e pronta disponibilidade constantemente manifestados, mas também pela confiança em mim depositada, quero manifestar a minha profunda e sincera gratidão. As suas reflexões e críticas, surgidas do seu saber, experiência e competência profissional, guiaram-me na realização deste trabalho de investigação. O seu trabalho de revisão final de artigos científicos e desta dissertação, companheirismo e apoio na apresentação de algumas comunicações foi inestimável. É importante dizê-lo: sem ele, este trabalho não teria chegado a bom porto.

Ao primo, Professor Azevedo Silva, pelas suas palavras de confiança e incentivo e pela sua disponibilidade para empreender a revisão preliminar deste trabalho.

À minha filha Diana, pelas suas muitas horas de revisão prévia dos relatórios técnicos, artigos e desta dissertação. Ao seu olho clínico e lógica férrea, crivo implacável de inconsistências e frases inacabadas ou incompreensíveis, muito devo.

Aos colegas da Escola Superior de Tecnologia de Viseu, muitos do Departamento de Informática, mas não só, pelo incentivo, ajuda, amizade e compreensão que me transmitiram ao longo destes últimos anos. Aqui cumpre-me destacar a Cristina, companheira de várias jornadas, por me dar

algo da sua força nos momentos de desânimo, pelos seus avisados conselhos, além da boa disposição, sempre patente, que tornou curtas, as longas viagens.

Agradeço o apoio concedido no âmbito do Programa de Desenvolvimento Educativo para Portugal (PRODEP), através da concessão de uma bolsa de Doutoramento, o que permitiu a dispensa integral de serviço docente pelo período de 2 anos e 9 meses.

Agradeço igualmente à Escola Superior de Tecnologia de Viseu que me permitiu ter usufruído da bolsa de doutoramento PRODEP III, acção 5.3 – Formação Avançada no Ensino Superior, Concurso N.º 02/PRODEP/2003. A esta mesma Escola e ao Departamento de Informática, agradeço algum apoio financeiro recebido para inscrição em duas conferências.

Aos membros da família, que acompanharam, de um modo mais ou menos próximo, a realização desta tese, envio um forte abraço de agradecimento, cumprindo-me destacar as tias da Mourinha, pelo seu cuidado e apoio sempre manifestado nas suas palavras carinhosas.

Finalmente, os meus agradecimentos à minha família mais chegada, Nanda, Diana e Paula. Especialmente à minha Mulher, que constituiu um importante suporte psicológico e uma constante fonte de estímulo nas horas de desalento, um beijo de muito Amor. Também pelos momentos que não lhes pude dedicar, pela compreensão das outras prioridades que se me impunham e por perdoarem as minhas frequentes ausências.

A todos, família, amigos e colegas reitero os meus agradecimentos pelo incentivo, paciência e compreensão.

Resumo

Restruturação Dinâmica de Estruturas Multidimensionais de Dados em Tempo Útil

O crescimento do tamanho dos Data Warehouses e do número dos utilizadores impuseram um stresse sucessivamente crescente nos sistemas de processamento analítico. Desde cedo se percebeu que a materialização de estruturas multidimensionais era uma forma de melhorar o tempo de resposta a interrogações de carácter agregado. Também a distribuição dessas estruturas podia ser uma mais-valia para aliviar o problema, permitindo uma escalabilidade a custos controlados, maior disponibilidade e eliminação de pontos-quentes. Mas estas soluções não são de implementação simples. Se a selecção das estruturas multidimensionais apropriadas se constitui como um problema NP-hard, a segunda vertente vem aumentar ainda a complexidade da primeira, uma vez que incorpora na equação de custos uma dimensão adicional – espaço – gerando novas dependências, cujas semânticas são capturadas pelo *lattice* distribuído. Já não se trata apenas de seleccionar as estruturas mais adequadas, considerando um perfil de carga, mas também materializá-las no(s) nó(s) mais vantajoso(s). Transversal a esta dupla abordagem da solução está uma outra dimensão, sempre omnipresente – o tempo – implicando o "envelhecimento" de cada solução proposta e a necessidade da sua afinação periódica.

Esta dissertação surge assim na confluência das três problemáticas aludidas: a selecção das estruturas multidimensionais, a sua possível distribuição e a temporalidade de recalibração. A solução para esta tripla abordagem implicou: 1) a evolução de modelos já existentes, mas estendidos para incorporar a dimensão espaço com os seus custos de comunicação,

heterogeneidade das redes de interligação dos nós da arquitectura e respectiva capacidade de processamento; 2) a concepção de algoritmos para a estimativa de custos, capazes de simular a execução paralela de tarefas (típica ao dispor-se de diversos nós de armazenamento e processamento); 3) a proposta de novas heurísticas para a solução do problema ou melhoria de propostas já existentes.

Em resultado da investigação empreendida, desenvolveu-se o núcleo de um protótipo de uma ferramenta para a administração de data warehouses, na forma de um conjunto de algoritmos que permitem empreender a optimização das estruturas multidimensionais de dados supondo uma qualquer distribuição espacial das estruturas e nós arquitecturais. Esta ferramenta constitui-se como uma bancada de trabalho pois foi concebida de forma a possibilitar investigação futura e suportar evoluções sucessivas e a sua inclusão num sistema de gestão global de um sistema de data warehousing.

Abstract

Multidimensional Data Structures Dynamic Restructuring in Useful Time

The increasing size of data warehouses and the number of users imposed a succeeding stress on the OLAP systems. From early times, it was perceived that the materialization of multidimensional structures was a way of improving the answering time of those systems to aggregated queries. Moreover, the distribution of those structures could be another way to deal with that problem, allowing scalability with controlled costs, a better availability and bottleneck avoidance. But these solutions aren't of easy implementation. If the proper selection of the multidimensional structures is a NP-hard problem, yet the other prospect increases the complexity of the problem, as it implies the inclusion of a new dimension – space – into the costs' equation, generating new dependencies, whose semantics are captured by the distributed cube lattice. Now, we have to deal not only with the selection of the most profitable multidimensional structures, but also to materialize them into the most advantageous node(s). A new dimension transverses this double solution's approach – time – implying the "aging" of each proposed solution and the need to a periodic tuning.

This dissertation emerges in the confluence of the mentioned triple points: the selection of the multidimensional structures, its possible distribution and the timing of recalibration. The solution for this triple approach implied: 1) the evolution of existent cost models, now extended to include the space dimension with its communication costs, the heterogeneity of nodes' interconnecting networks and the processing power of the OLAP server nodes; 2) the design of cost estimation algorithms, which are able to simulate the parallel execution of tasks (a typical situation, having

several storage and processing nodes); 3) the proposal of several new approximate optimizing algorithms or the improving of pre-existent proposals applied to the optimization of the multidimensional structures.

This research allowed the development of a prototype's core tool for data warehouses administration, shaped as a set of algorithms which allows making the optimization of the multidimensional data structures, supposing any spatial distribution of the architectural structures and nodes. This tool is just like a workbench, as it was designed in order to allow the future research work and the easy support of succeeding evolutions and also its inclusion into a global data warehousing management system.

Índice

Índice de Figuras

Índice de Tabelas

Índice de Algoritmos

Capítulo 1

Introdução

1.1 Sistemas de Processamento Analítico

Com a crescente globalização, abertura e desregulamentação dos mercados, a economia actual evidencia um ambiente progressivamente mais dinâmico e volátil. A rapidez evolutiva faz surgir a todo o instante não só novas oportunidades de negócio, mas também novas ameaças à posição competitiva de uma organização. É crucial para as empresas poder responder às novas situações que se lhe colocam, tomando decisões atempadas, conexas e ajustadas. O decisor, confrontado com este ambiente de incerteza, desespera por algo capaz de o ajudar e guiar. Esta realidade transformou a informação no "novo *Graal*", em cuja demanda todas as organizações estão envolvidas. A disponibilidade de informação em tempo real, que permita uma caracterização do negócio, não só a nível geral, mas também de detalhes acerca de tópicos específicos, tornou-se condição de sobrevivência. Esta informação correcta, de acesso global e atempado, permite às empresas tomar as decisões certas e no momento adequado: eis por que a maioria das empresas grandes e médias criaram *Data Warehouses* (DW), definidos como "uma colecção de dados empresariais, orientados a temas, integrados, mostrando a variabilidade no tempo e não voláteis" [Inmon, 1996].

Esta abordagem implica, em regra, a extracção periódica de dados dos sistemas operacionais (normalmente sob a forma de instantâneos temporais) que, depois de sujeitos a um conjunto de operações, cujo objectivo é assegurar a sua desejável consistência e pureza, são integrados no DW. Os DW surgiram numa perspectivação dual dos sistemas de informação. Postularam a separação do mundo operacional do informacional, respondendo a fraquezas e incongruências

várias que eram patentes nos sistemas de informação originais para a gestão. Também, e particularmente, a disponibilização de dados fiáveis para análise veio permitir aos decisores compreender o que os dados realmente significam. Mas os decisores vêem o negócio de uma forma caracteristicamente multi-perspectiva: um dado facto em análise (a informação de interesse para o decisor) pode ser analisado segundo caracterizações várias (as dimensões) que devem ser passíveis de manipulação directa pelo próprio utilizador. Assim, surgem os sistemas capazes de possibilitar ao decisor a investigação dos dados segundo diferentes perspectivas de negócio e navegar por eles de acordo com o resultado das observações prévias. Os sistemas capazes de possibilitarem este tipo de análise foram cunhados em [Codd. et al., 1993] de sistemas OLAP (*On-Line Analytical Processing*) ou sistemas de processamento analítico, usando dados organizados multidimensionalmente num modelo conhecido como cubo de dados [Chaudury & Dayall, 1997], uma simplificação da metáfora multidimensional para três dimensões. Estes permitem organizar a informação segundo as várias linhas de análise definidas pelos agentes de decisão – as dimensões – e armazenar tanto os dados base como os resultados de processos de cálculo realizados sobre um conjunto de medidas – os indicadores de gestão – previamente definidas. As dimensões estão normalmente organizadas em hierarquias. Por exemplo, a hierarquia tempo poderia ser do tipo: dia → mês → trimestre → ano, ou então dia → semana → ano. A organização dos dados na sua forma multidimensional e hierárquica vai assegurar as operações típicas em OLAP, como *drill-down*, *roll-up*, *pivoting* e *slice and dice* [Chaudury & Dayall, 1997].

As interrogações OLAP típicas e manipulações multidimensionais devem ser, desejavelmente, respondidas num intervalo de tempo curto. O próprio acrónimo OLAP traduz explicitamente essa característica. Também a qualidade das decisões, produtividade e satisfação dos agentes de decisão dependem fortemente dos tempos de resposta às interrogações colocadas. Um exemplo da conveniência e necessidade das agregações será, porventura, mais ilustrativo. Suponhamos que se pretende uma análise comparativa das vendas de uma cadeia de supermercados, deste ano, face ao ano transacto, de um conjunto de famílias de produtos, por meses, para as lojas da zona norte do país. Está-se em presença de três dimensões de análise – produtos, lojas e tempo – sendo as vendas o dado em análise. Ora, para satisfazer esta consulta, uma solução poderia ser: pesquisar os dados base (a nível de detalhe – vendas correspondentes a cada talão) e agrupá-los até ao nível conveniente da hierarquia em cada dimensão. Estar-se-ia em presença dos chamados cálculos *on-the-fly*, muito consumidores de tempo e recursos de processamento, especialmente para grandes volumes de dados. Contrariamente, se a agregação loja-mês-produto tiver sido previamente calculada e armazenada, a satisfação dessa interrogação resume-se a uma consulta

simples de alguns valores. Todavia, a elegância desta solução tem um preço: o pré-cálculo e armazenamento (materialização) da agregação, denominada vista ou subcubo materializado. O preço desta solução é pago em espaço de armazenamento e, especialmente, em tempo de cálculo e materialização (ou manutenção), já que é necessário assegurar a consistência das agregações materializadas com as relações base, sempre que estas são modificadas em resultado das transacções ou eventos ocorridos. Mas o número possível das agregações é muito elevado. Basta, por exemplo, considerar cinco dimensões com quatro níveis de hierarquias por dimensão (valores modestos, em sistemas OLAP reais), para se ter 1024 agregações, que terão de ser alojadas (armazenadas) e mantidas. Para um valor mais elevado de dimensões e complexidade, o seu número torna-se muito elevado, tornando-se incomportável a sua materialização total e havendo, portanto, que procurar soluções tendentes à limitação do seu número.

Percebeu-se também, já, que uma dada agregação, sendo idealmente adequada à resposta a uma interrogação e podendo responder a muitas outras, pode também, e dado um perfil de interrogações, ser inútil - não haver qualquer interrogação que a utilize. O perfil de utilização pode, assim, ser utilizado para seleccionar, de entre as possíveis agregações, quais as que se revelam mais benéficas, atendendo a um ou mais constrangimentos (espaço de materialização e/ou tempo disponível para a manutenção das estruturas). É este, na sua essência, o denominado problema de selecção de vistas ou subcubos (pss), de características reconhecidamente NP-hard [Harinarayan et al., 1996]. A selecção adequada pode traduzir-se em ganhos muito elevados, conseguindo-se, com uma percentagem baixa (da ordem dos 10-20%) do espaço de materialização total do subcubo, valores quase idênticos para o tempo de resposta às interrogações colocadas, tomando, como valor comparativo base, a materialização completa do cubo. Este ganho potencial mostrou a relevância do problema, que mereceu, desde muito cedo, a atenção da comunidade científica e para o qual foi sendo proposto um variado número de soluções. O surgimento de novos desafios e soluções arquitecturais incentivou a investigação neste domínio, já que importa adaptar e estender as velhas soluções às novas realidades.

Neste contexto, aparece a possibilidade de tirar partido do perfil de utilizador, permitindo a aludida possibilidade de selecção das estruturas mais apropriadas. Mas, alterando-se as necessidades dos utilizadores, será necessária a recalibração das estruturas multidimensionais. Vai haver subcubos materializados que deixarão de ser úteis, e outros, não materializados, que seriam, porventura, benéficos. Esta recalibração (restruturação) deveria, desejavelmente, acompanhar de muito perto as mutações no perfil de utilização. Mas, devido à dimensão das estruturas multidimensionais, o

custo é, normalmente, muito elevado. Eis por que este processo assume, em regra, três possíveis abordagens:

1. A denominada abordagem estática, tendo como alvo grandes estruturas e um âmbito alargado, efectuada a intervalos temporais longos, dado o seu custo muito elevado.
2. A abordagem dinâmica, que permite um ajuste mais fino, conseguida, em regra, por captura e materialização, em caches apropriadas, das respostas às interrogações colocadas por utilizadores.
3. A abordagem pró-activa, onde é assumida uma vertente especulativa, procurando adequar as estruturas às necessidades futuras do utilizador.

Com o volume sempre crescente de dados, o tamanho dos cubos materializados tornou-se muito grande. Também o número de utilizadores aumentou. Os dois factores impuseram um stresse crescente aos sistemas de processamento analítico - hardware sucessivamente mais poderoso foi necessário a custos financeiros exponencialmente crescentes. A distribuição das estruturas multidimensionais por vários nós, doravante designados por nós-servidores OLAP (NSO) é outra via que pode ser tomada. Esta solução vai permitir uma fácil escalabilidade[1] a custos controlados, já que utiliza hardware mais simples, mas com capacidade agregada idêntica. Esta nova realidade aumenta a complexidade do problema da selecção de subcubos, introduzindo uma nova dimensão: espaço, capturável pela utilização de facilidades de comunicação, mas que acrescenta um novo custo a levar em conta. As dependências agregacionais entre subcubos assumem agora duas realidades: intra e inter-nó. Em resumo, não basta agora seleccionar os subcubos mais adequados, mas impõe-se providenciar a sua correcta alocação, havendo que ter em conta, neste momento, o espaço por nó, o perfil de utilização por nó, os links de comunicação e custos inerentes, e as dependências inter-nós.

[1] Tradução para *scalability*. Apesar de o termo não ser contemplado no Dicionário da Língua Portuguesa Contemporânea, Academia das Ciências de Lisboa e Editorial Verbo, 2001, nem no Dicionário Houaiss da Língua Portuguesa, Círculo de Leitores, Lisboa, 2002, os dois dicionários de Língua Portuguesa tidos como mais representativos, duas possíveis traduções parecem perfilar-se: escalabilidade e escalaridade. Da primeira surgem 211000 entradas no Google, enquanto da segunda, apenas 210. Também em ciberdúvidas www.[3] surge o primeiro como a melhor tradução e de igual forma em Wikipédia www.[5], a escalabilidade é definida como a propriedade de um sistema qualquer que lhe confere a capacidade de aumentar o seu desempenho sob carga quando recursos (tipicamente hardware, no caso de computadores) são acrescentados a esse sistema. Um tal sistema denomina-se de escalável. Por outro lado, escalaridade aparece ligada principalmente a super-escalaridade dos processadores. Também em Glossário Tecnológico Dígitro www.[4], escalabilidade é definida como a característica do sistema ou equipamento que pode crescer em escala, isto é, que possibilita incrementos de capacidade ou funcionalidades, acompanhando as necessidades dos utilizadores.

1.2 Motivação, Vectores e Objectivos de Investigação

Cedo se percebeu que os sistemas de processamento analítico são peças fundamentais no apoio à tomada de decisões, cuja qualidade vai depender de muitos factores, de que poderemos salientar a própria natureza dos dados disponíveis e a obtenção da informação pretendida, quando desejada. Para estes factores contribuem a extensão dos dados recolhidos (em profundidade temporal, abrangência de sectores de actividade e geográfica), a sua confiabilidade, a sua organização e a arquitectura do sistema instalado. Um crescimento na extensão dos dados vai provocar um crescimento da complexidade dimensional do cubo, fazendo aumentar o número de agregações e o seu tamanho, mas também o tamanho intrínseco de cada agregação. Este facto impõe um stresse acrescido sobre o hardware, degradando o tempo de resposta às interrogações e aumentando o tempo necessário ao refrescamento do DW e agregações. A distribuição das estruturas multidimensionais, sob a forma de novas soluções arquitecturais, surge como uma possível solução, sendo necessário prover à sua optimização, pela adaptação das soluções existentes e, porventura, utilizando heurísticas recentes que surgem a cada passo, constituindo-se como a motivação para este trabalho. Importa investigar a aplicação de umas e outras aos sistemas de processamento analítico de forma a optimizá-los e contribuir para a minimização do tempo de resposta a interrogações e assim à satisfação dos decisores e maximização da sua produtividade. A resolução desta problemática pode ser resumida em três grandes vectores complementares, cuja investigação parcial norteou este trabalho:

1. se a materialização das agregações constitui condição de desempenho, uma mais eficaz selecção dos subcubos traduzir-se-á, decerto, numa melhoria da satisfação dos utilizadores e qualidade das decisões suportadas pelo conhecimento obtido;
2. com o crescimento do volume de dados, a plataforma de hardware de suporte OLAP torna-se de difícil crescimento: uma distribuição do cubo por vários servidores permite essa escalabilidade a custos controlados;
3. com a alteração do perfil de utilização, importa efectuar a recalibração das estruturas multidimensionais.

Os três vectores e sua interdependência, que resumem a motivação deste trabalho, mostram-se na Figura 1.1.

O primeiro vector constituiu, desde muito cedo, motivo de grande esforço de investigação. Trata-se de um problema de optimização de carácter combinatório, pois que uma dada agregação pode estar ou não materializada. Novos algoritmos de solução aproximada para estes problemas surgem

a cada passo e, por outro lado, novas formas de aplicar os já utilizados podem ser intentadas, nomeadamente, sob a forma de heurísticas híbridas. A complementaridade faz-se, aqui, já que os algoritmos podem, por um lado, actuar na optimização de sistemas OLAP de repositório centralizado e, por outro, assumindo uma qualquer arquitectura de distribuição, desde que o modelo de custos referente à arquitectura em causa esteja disponível (complementaridade para o vector 2). Contudo, há que ter em conta que um maior grau de distribuição implica, em regra, uma maior complexidade, colocando assim requisitos de satisfação mais apertada aos algoritmos de optimização, nomeadamente no que concerne à respectiva escalabilidade. A complementaridade para o vector 3 pode traduzir-se nas três abordagens já referidas (estática, dinâmica e pró-activa), actuando especialmente sempre que o objecto de restruturação sejam os repositórios de carácter mais estável, estando as caches, em regra, excluídas.

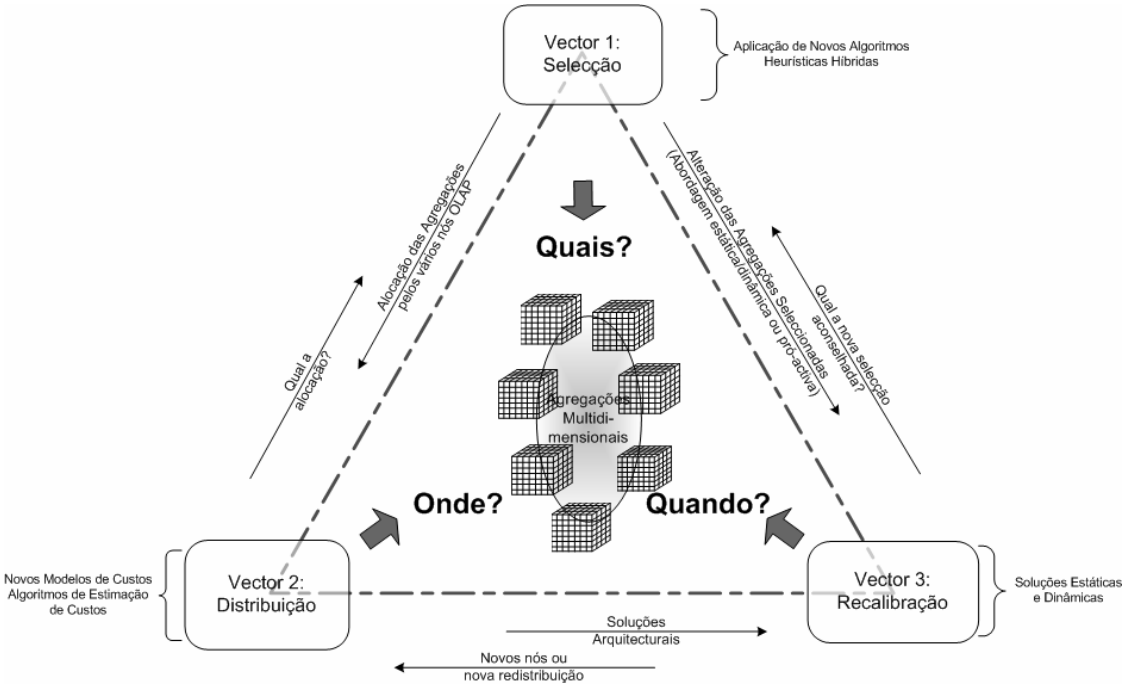


Figura 1.1. Complementaridade dos três vectores de investigação.

O segundo vector tem desenvolvimentos recentes, assumindo uma forma de distribuição local do cubo, estando, neste caso, os vários servidores em locais espacialmente próximos, ou assumindo uma perspectiva mais alargada, na qual se poderá dispor de um conjunto de servidores OLAP interligados por uma WAN e constituindo-se como actores de diferentes papéis: servidores base de

refrescamento periódico típico, caches OLAP intermédias [Kalnis & Papadias, 2001] e mesmo caches em clientes numa base *peer-to-peer* [Kalnis et al., 2002a]. Para qualquer destas soluções há que encontrar um modelo de custos que permita aquilatar do benefício da materialização de uma dada agregação (ou fragmento) numa determinada localização. Depois, o vector selecção pode encarregar-se de propor soluções, sempre que a recalibração o entender oportuno.

No que diz respeito ao terceiro vector, há que diferenciar as três soluções quanto à frequência de recalibração das estruturas referidas na secção anterior: estática, dinâmica e pró-activa. Se bem que as duas últimas implicam um esforço, em regra, pequeno, a primeira obriga a reconstruções de custos de materialização muito elevados. Para a abordagem estática, a utilização dos algoritmos de selecção do vector 1 e modelos e algoritmos de estimativa de custos fornecidos pelo vector 2 permitem um cálculo do desvio face à distribuição de melhor qualidade proposta pelo algoritmo de selecção, em face de um novo perfil de utilização, o que poderá desencadear o processo de recalibração em larga escala. Esta poderá também ocorrer a intervalos temporais mais curtos (por exemplo a cada refrescamento) através da simples eliminação de umas (poucas) agregações e inclusão de outras, mais benéficas, observando o constrangimento temporal de manutenção e continuando a utilizar os mesmos meios. Já para as duas últimas abordagens, que implicam normalmente repositórios de carácter transitório (caches), o processo de decisão de selecção será, em regra, distribuído, ainda que os modelos e estimativa de custos possam ser utilizados, tal como os algoritmos de selecção, que poderão, igualmente, surgir sob a forma de agentes residentes em cada nó servidor. Além disso, a utilização de heurísticas e modelos de custo poderão permitir a sugestão de modificações da arquitectura, nomeadamente a inclusão ou a eliminação de nós e alteração de características (capacidade de processamento ou armazenamento, por exemplo).

No horizonte, ou subjacente a todos os vectores acabados de descrever, está sempre o utilizador. Este aparece imediatamente no vector 1 e está subentendido nos vectores 2 e 3, já que a fácil escalabilidade e a recalibração das estruturas são consequência das acções dos utilizadores e meios para a sua satisfação. Pode assim dizer-se que a resultante dos três vectores descritos conflui no objectivo principal deste trabalho de investigação: contribuir para a satisfação dos utilizadores das plataformas de processamento analítico, mormente dos decisores, aumentando a sua produtividade e qualidade das suas decisões, através da melhoria substancial do tempo de resposta às interrogações OLAP colocadas. Na Figura 1.1 mostram-se (com chamadas nas chavetas) as formas de actuação que se constituem como objectivos deste trabalho: a acção sobre os três vectores referidos anteriormente.

Se a acção em todos os vectores e de igual extensão seria ideal, há factores temporais limitativos e, portanto, importa restringir o seu domínio. Assim, no vector selecção, a actuação coloca-se ao nível da optimização das estruturas multidimensionais de dados na sua vertente centralizada e, com especial ênfase, na sua vertente distribuída. A prossecução deste objectivo envolve a concepção, desenvolvimento e avaliação de algoritmos de selecção e alocação de cubos para sistemas OLAP centralizados e distribuídos, utilizando heurísticas estabelecidas (*greedy*), evolucionárias (genéticas e co-evolucionárias), de enxame (optimização por enxame de partículas), *hill climbing* com *simulated annealing* e por combinação de vários métodos de pesquisa através de um mecanismo de metamorfose, onde cada entidade de busca vai assumir a forma mais adequada ao estado actual do processo de pesquisa. No vector 2, foca-se, com especial ênfase, a perspectiva distribuída OLAP, sendo desenvolvidos modelos de custos lineares e não lineares capazes de permitir o cálculo de custos de interrogação e manutenção, recorrendo para tal a algoritmos de cálculo com características apropriadas que, em arquitecturas distribuídas, incluem simulação de execução paralela, especialmente para a estimativa dos custos de manutenção a usar pelos algoritmos de selecção do vector 1. Quanto ao vector 3, o domínio é algo restrito, pois que o objectivo cinge-se à abordagem estática ou dinâmica, endereçada a estruturas estáveis. Na verdade, os modelos e algoritmos de cálculo de custos permitem uma abrangência temporal mais alargada, mas importa limitar o âmbito deste trabalho.

1.3 Exequibilidade dos Objectivos Propostos

A viabilidade da prossecução dos objectivos apresentados é sustentada, por um lado, pela percepção apriorística das vantagens que a distribuição de dados pode trazer na resolução de muitos dos problemas atrás focados (especialmente quanto à escalabilidade fácil, a custos controlados) e, mais ainda, da prova da eficácia desta solução, especialmente no domínio transaccional, onde está bem estabelecida a sua viabilidade prática. Por outro lado, os estudos de desempenho dos algoritmos referenciados em problemas genéricos de carácter combinatório e, também, a sua aplicação, já reportada, ainda que em âmbitos diversos, a este mesmo problema (essencialmente em arquitecturas centralizadas), permitem esperar um bom desempenho para os novos enquadramentos propostos.

Também uma análise SWOT – pontos fortes (forças), pontos fracos (fraquezas), oportunidades e ameaças) [Adams, 2005], [Pearce & Robinson, 2005] – relativa aos sistemas de processamento

analítico, ajuda a justificar as formas de actuação definidas na secção 1.2 e, por outro lado, mostra também o seu impacto nos sistemas de processamento analítico e a sua viabilidade. Segundo a metodologia relacionada com o planeamento estratégico, há que actuar de forma a maximizar os pontos fortes e aproveitar as oportunidades para combater as ameaças e minorar os pontos fracos. Veja-se em que medida os objectivos traçados e sua viabilidade permitem o fortalecimento da posição dos sistemas OLAP, discutindo o seu impacto em cada um dos vectores da análise SWOT mostrada na Figura 1.2.

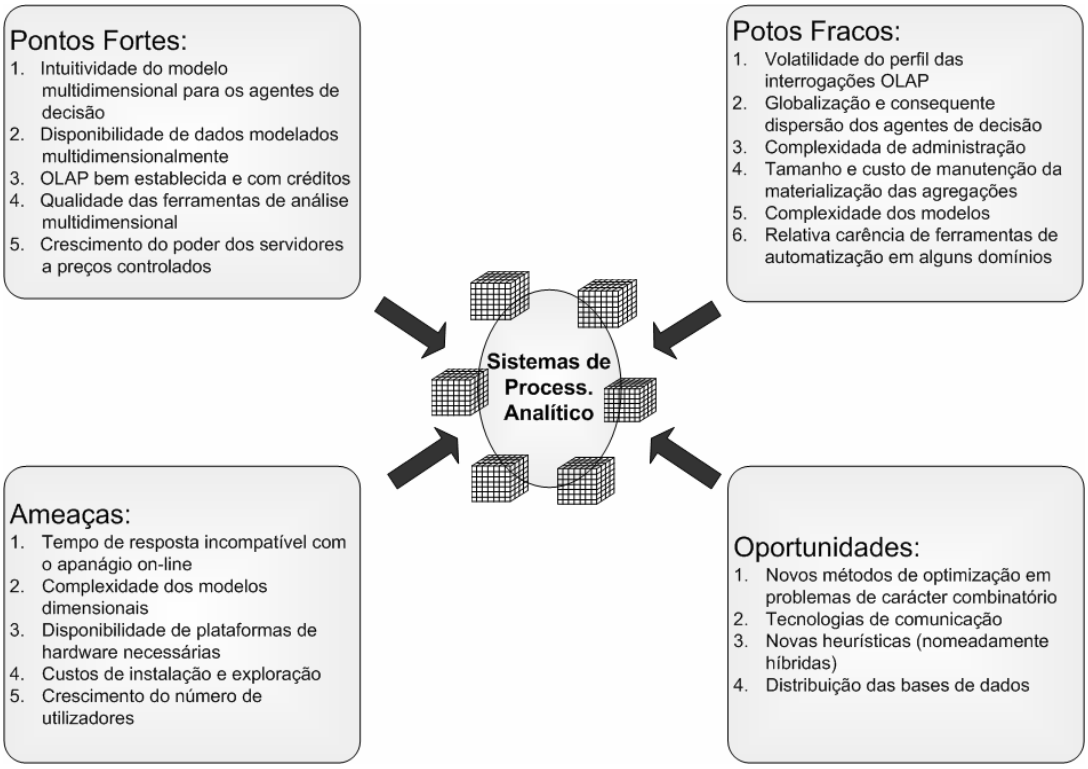


Figura 1.2. Análise S.W.O.T. relativa a sistemas OLAP.

Iniciando a discussão pelas oportunidades, e relativamente a cada uma, há a considerar o seguinte:

1. Os novos métodos de optimização permitirão uma mais conveniente selecção das estruturas a materializar que terão um impacto positivo em todas as ameaças: promoverão uma melhoria do tempo de resposta das interrogações colocadas pelos utilizadores; permitirão que bons tempos de resposta sejam conseguidos com a utilização

de plataformas de hardware menos poderosas (deixa de ser necessária uma abordagem do tipo "força bruta"), implicando custos de instalação e exploração mais baixos; e, especialmente, usando arquitecturas OLAP multi-nó (que aqui serão designadas pelo acrónimo M-OLAP), será possível o crescimento dos modelos dimensionais (com consequente aumento do tamanho e número de agregações) e do número de utilizadores, simplesmente por adição de mais nós e redistribuição dos subcubos. Este mesmo suporte simples de arquitecturas distribuídas terá um impacto positivo sobre todos os pontos fracos enumerados, incluindo o primeiro, se for permitido o suporte simultâneo de agregações em refrescamento incremental e integral.

2. As facilidades de comunicação, com larguras de banda crescentemente superiores e menores latências, permitem a distribuição do cubo OLAP e a construção de arquitecturas distribuídas, uma amálgama de repositórios estáticos e dinâmicos. Estas reduzirão a pressão de todas as ameaças referidas sobre o OLAP, já que permitirão um *scale-up* simples e eficaz. Permitirão também actuar sobre o 1.º, 2.º e 4.º pontos fracos enumerados, já que permitirão a distribuição dos repositórios de carácter transitório e terão um impacto positivo sobre os custos de comunicação no acesso remoto ou no refrescamento de repositórios remotos e diminuirão os custos de manutenção das agregações.
3. As novas heurísticas híbridas, permitindo conceber e desenvolver novos algoritmos de selecção e alocação de cubos, terão um impacto idêntico ao referido em 1., pelo que não serão tecidos mais comentários.
4. Relativamente à distribuição das bases de dados (ponto 2.), na prática, está dependente de facilidades de comunicação e disponibilização de algoritmos de selecção e alocação apropriados (pontos 1. e 3.). Assim, os impactos positivos enumerados em todos os pontos anteriores terão, desta oportunidade, uma quota parte.

Quanto às oportunidades atrás referidas vão intensificar os pontos fortes números 2 e 3 mostrados na Figura 1.2, já que:

1) Melhoram a disponibilidade de dados modelados multidimensionalmente, ao permitir uma fácil escalabilidade e assim a disponibilização de meios que suportem maiores repositórios.
2) Reforçam os créditos do OLAP, aumentando a satisfação dos utilizadores e retorno do investimento (ROI) realizado, ao melhorar a qualidade das decisões tomadas.

Relativamente ao ponto forte número 5, conhecido o aumento da capacidade dos próprios servidores, especialmente com a disponibilização de processadores e dispositivos de armazenamento cada vez mais poderosos a preços idênticos (atente-se à chamada lei de Moore [Moore, 1965],www.[1],www.[2] que pode expressar-se como previsão de duplicação de capacidade de processamento e armazenamento a cada 18 meses), vai contribuir para combater todas as ameaças identificadas.

1.4 Estrutura da Dissertação

Este trabalho tem como objecto as estruturas multidimensionais de dados que, de uma forma simplista, passaremos a designar por cubo de dados. Importa, portanto, enquadrar o leitor na problemática a ser tratada, mostrando as suas vertentes, com especial ênfase no cubo de dados e processamento analítico, seus desafios e soluções. Depois desta introdução, inicia-se o capítulo 2 com um conjunto de conceitos e noções associadas ao cubo de dados e sua utilidade. Mostra-se depois a sua evolução e desafios colocados. A enumeração das soluções com uma análise custo X benefício vai dar sustentabilidade à direcção tomada neste trabalho. Estas soluções, estando claramente no domínio da optimização, e, dado o custo incomportável do cálculo de custos em situações de simulação real, serão sempre suportadas por modelos de custos, discussão esta que termina o capítulo 2 e introduz o seguinte.

O capítulo 3 inicia com a descrição do modelo de custos linear e *lattice* de dependências. Apresenta, depois, modelos de custos sucessivamente mais abrangentes, resultantes de evoluções do modelo de custos linear, procurando adaptá-lo a novas condições ambientais. Estes modelos suportarão a concepção e desenvolvimento de algoritmos de estimativa de custos adequados à arquitectura OLAP em causa, que serão apresentados e discutidos ainda neste capítulo, que termina com a avaliação dos algoritmos propostos.

Estando em posse das ferramentas de estimativa de custos, importa agora lançar a tarefa da optimização das estruturas multidimensionais, quer na sua abordagem tradicional centralizada, quer especialmente na sua vertente distribuída. Perfila-se um conjunto de algoritmos: desde a tradicional heurística *greedy*, enveredando depois decididamente pelas aproximações evolucionárias, de enxame e com *hill climbing*, com aplicação de hibridações, sempre que resultados reportados em avaliações de desempenho o pareçam aconselhar. A descrição da

concepção dos vários algoritmos, a sua descrição formal e avaliação simulada experimental na sua aplicação às várias soluções arquitecturais vai ser o objecto dos capítulos seguintes deste trabalho.

Assim, no capítulo 4, vai propor-se um algoritmo de optimização estabelecido em 1995/97: optimização por enxame de partículas, na sua variante discreta (ODiEP) também denominado (OEP-Q), já que a variante discreta introduz um mecanismo probabilístico, com possível simbologia e analogia quântica. A aplicação do algoritmo com mapeamento do problema no espaço multidimensional de pesquisa ao problema de selecção do cubo para uma arquitectura centralizada, aplicando um constrangimento espacial de materialização, vai ser central neste capítulo, sendo efectuada igualmente uma avaliação comparativa da nova solução em face das soluções já existentes: algoritmos *greedy* e genéticos padrão.

O capítulo 5 alarga a abrangência do processo de optimização a uma arquitectura M-OLAP. O problema aqui já não se constitui apenas como a selecção dos cubos, sendo também necessário estabelecer onde será efectuada a respectiva materialização. O objecto de optimização continua a ser a minimização dos custos de interrogação e manutenção, aplicando-se igualmente um constrangimento espacial (agora relativo a cada nó), tendo como referencial o modelo de custos linear estendido e respectivos algoritmos de cálculo de custos. Abordagens evolucionárias (genéticas padrão) e co-evolucionárias, ODiEP e sua variante cooperativa e multi-fase são sucessivamente propostas e avaliadas, sendo igualmente utilizada uma heurística greedy para avaliação comparativa de desempenho.

No capítulo 6 vai utilizar-se o modelo de custos mais abrangente, com inclusão de não linearidades e suporte ao cálculo de custos de manutenção integral ou incremental, aplicado mais uma vez à arquitectura M-OLAP. Vai propor-se a utilização de dois outros algoritmos: 1) *Hill climber* com *simulated annealing* (HC-SA) e 2) metamorfose. No primeiro, temos uma população de *hill climbers* que procura a minimização de custos de interrogação, mas observando um constrangimento temporal de manutenção. Esta optimização vai ser tratada numa dupla perspectiva: estática e dinâmica. A primeira, se for permitida, por exemplo, a recriação das várias estruturas multidimensionais, supondo uma reconfiguração em larga escala (provavelmente com custos de manutenção bastante elevados), podendo mesmo tratar-se da sua completa recriação. Já a segunda, que poderemos englobar no que se convencionou denominar abordagem dinâmica, vai proceder a ajustes "finos" acomodáveis na janela de manutenção (obedecendo assim ao constrangimento de manutenção) e efectuando este ajuste a intervalos temporais curtos (correspondentes, por exemplo, ao período típico de refrescamento das estruturas), daí a

caracterização dinâmica dada a esta perspectiva. Ao permitir a simulação de desmaterialização de estruturas multidimensionais pré-existentes com sua substituição por outras (a ser consideradas como custos de manutenção integral) adicionados ao custo de manutenção incremental das estruturas mantidas, o acompanhar de perto das alterações havidas no perfil de utilização permitirá decerto uma melhoria do tempo de resposta às interrogações colocadas, evitando o "envelhecimento" das estruturas e sua actualização em massa típico na primeira perspectiva, onde, durante um período normalmente longo, se assiste a uma desadaptação progressiva das estruturas em face da carga colocada. No segundo algoritmo, esta mesma perspectiva é tratada, utilizando um algoritmo que constitui como que uma tripla abordagem, mas mais do que uma simples justaposição dos elementos constituintes e mais do que uma simbiose: utiliza uma comunidade de entidades de busca capazes de transmutação. Na realidade, cada entidade de busca pode decidir transformar a sua forma e comportamento actual, metamorfoseando-se. A descrição do algoritmo e sua avaliação experimental são também apresentadas no capítulo 6. Algumas conclusões, relativas às soluções algorítmicas apresentadas, completam este capítulo.

Esta dissertação termina com o capítulo 7, no qual se fará uma avaliação comparativa das soluções propostas e discussão do seu enquadramento arquitectural, mostrando em que medida os objectivos foram atingidos e terminando com propostas de trabalho futuro. Desde já, poderá mencionar-se a inclusão de outras formas no algoritmo metamorfose (por exemplo a pesquisa tabu) e incluir heurísticas de pesquisa local em algum dos intervenientes ou utilizar algumas variantes. Outra vertente pode constituir-se como a utilização de um outro algoritmo de enxame já tão aplicado a problemas de carácter combinatório: optimização por colónia de formigas. Apesar de baseado num paradigma algo diverso, onde o número de estados é limitado e existe a possibilidade de usar heurísticas locais, a sua adaptação ao problema é possível, havendo que avaliar o seu desempenho. Mais ainda, a hibridação deste paradigma com o algoritmo HC-SA poderá permitir a melhoria do seu desempenho, nomeadamente com a inclusão de uma partilha de conhecimento na forma de *stigmergy*.

Capítulo 2

Optimização de Estruturas Multidimensionais de Dados

2.1 Estruturas Multidimensionais de Dados

As estruturas multidimensionais de dados são o componente central de qualquer sistema de processamento analítico. Independentemente da sua localização e distribuição, permitem o armazenamento de dados modelados sob o paradigma multidimensional, derivado da forma nativa como os gestores e outros agentes de decisão vêem um negócio – uma visão multi-perspectiva que permite a caracterização múltipla de um facto. Os modelos de dados multidimensionais são concebidos expressamente com o propósito de suportar a análise OLAP: navegar os dados a vários níveis de agregação, seleccionando umas perspectivas e escondendo outras, usar funções de agregação diversas, rodar as perspectivas, etc. A visualização desta metáfora multidimensional segundo três dimensões é apresentada como um cubo (onde cada dimensão espacial corresponde a cada caracterização do facto em análise). Em sistemas analíticos reais, o número de dimensões é, normalmente, maior, devendo-se então falar em hipercubo ou multicubo (generalizado para *n* dimensões), mas, por simplicidade, a designação cubo continua a ser a preferida, já que a única que permite um mapeamento visual imediato.

O modelo relacional introduzido nos anos 70 por Codd possui um conjunto de características que o torna ideal para a modelação de dados relativa a sistemas de processamento transaccional (OLTP). A normalização assegura a consistência e a integridade dos dados, permitida pelas características das transacções que aí decorrem: muitas transacções concorrentes, mas individualmente

pequenas. Cada transacção vai lidar com um pequeno número de registos, em ambiente de actualização. Este modelo normalizado é, contudo, inadequado para análise de dados. Uma consulta irá requerer, em regra, operações de junção entre muitas tabelas (dada a dispersão dos dados consequência da normalização). Ora, as operações de junção são inerentemente "caras", mesmo quanto se utiliza algoritmos avançados, o que torna as análises demoradas. Mas o ambiente cada vez mais competitivo e a necessidade de tomar decisões num ambiente em mudança rápida, obrigou a lançar mão de todos os meios tendentes à minimização da incerteza. Um maior conhecimento permite que as decisões e gestão das organizações sejam as mais adequadas, permitindo assim alcançar uma melhor posição no mercado. Mas esta necessidade esbarrava com as limitações do SQL, já que, qualquer resultado a uma consulta é apresentada como uma lista, de difícil e longa leitura, pois que uma simples tabela de dupla entrada não era suportada. O decisor necessitava de poder manipular livremente os dados e visualizá-los segundo a sua forma nativa de ver o negócio. A resposta a estas necessidades fez surgir, nos anos 90, uma nova classe de aplicações analíticas. O próprio Codd baptizou-as de OLAP (*On-Line Analytical Processing*) [Codd. et al., 1993], sendo uma das suas características mais salientes o facto de permitirem a análise de dados multidimensional. Esta multidimensionalidade inerente aos modelos de dados OLAP deriva, de forma imediata, da visão que o analista tem do negócio e da forma como é permitida a interacção com o utilizador. O negócio é visto sob a forma multi-perspectiva, em que um dado facto pode ser analisado segundo caracterizações várias, que devem ser passíveis de manipulação directa pelo próprio utilizador. Ou seja, o utilizador deve poder interagir directamente com o esquema lógico multidimensional do sistema OLAP para formular consultas. Assim, um sistema OLAP é modelado sob a forma de um esquema multidimensional [Kimball, 1996] que tem um impacto directo sobre a funcionalidade do sistema e é conduzido pelos requisitos do utilizador.

2.2 Os Cubos

Um cubo de dados, em termos técnicos, não é mais do que uma projecção multidimensional redundante de uma relação, tendo sido proposto em [Gray et al., 1996] como uma generalização do operador SQL *Group-By*. Um cubo pode ser visualizado como uma grelha multidimensional construída a partir dos valores das dimensões. Cada célula nesta grelha contém uma série de medidas (valores numéricos que "vivem" dentro da célula), todos caracterizados pela mesma combinação de coordenadas, instância para dimensões ou níveis.

A noção de dimensão é um conceito essencial e diferenciador em dados multidimensionais. As dimensões são utilizadas com dois propósitos: a selecção de dados e o seu agrupamento a um dado nível. Cada dimensão está organizada sob a forma de uma ou várias hierarquias. Por exemplo, a hierarquia da dimensão produto poderia ser, como é mostrado na Figura 2.1, do tipo: *produto → tipo* ou *produto → tamanho*, sendo o produto, neste caso, uma dimensão múltipla. Cada hierarquia é composta de um número de níveis, cada um representando um nível de detalhe que pode interessar às análises a ser executadas. Também, para cada par dimensão/nível, teremos um conjunto de instâncias (por exemplo para mês, ter-se-á Janeiro, Fevereiro, etc. ou para zona geográfica, a zona norte, centro, etc.) que se denominam tipicamente por membros de dimensão ou valores de dimensão.

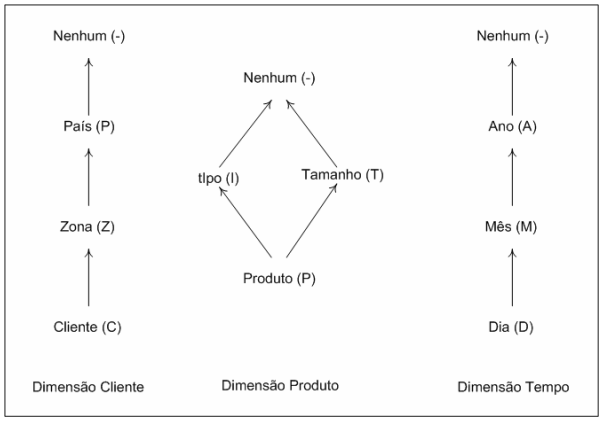


Figura 2.1. Exemplo de esquema dimensional adaptado de [TPC-R, 2002].

Na Figura 2.2 mostra-se um exemplo de um cubo de dados. Mostram-se três dimensões (produto, cliente e tempo) que irão caracterizar as medidas número de produtos vendidos e valor das vendas (o facto em análise), relativa a uma cadeia de supermercados.

O nível das dimensões ao qual ocorre a combinação de valores irá determinar a granularidade[1] do facto. No exemplo da Figura 2.2 (à esquerda), a granularidade do cubo é "produto por cliente por mês", aqui representado por "cliente x produto x mês" ou simplesmente pela combinação de letras

[1] Tradução para *granularity*, designando a qualidade ou condição de granular. Refira-se que o termo é inexistente no Dicionário da Língua Portuguesa Contemporânea da Academia das Ciências, onde mesmo granulosidade também não é referenciado. No entanto ambas as palavras surgem no Dicionário Houaiss da Língua Portuguesa, sendo aqui preferida a locoção ganularidade, pois que além do termo ter um significado mais consentâneo com o conceito, é uma tradução mais imediata do termo em inglês.

correspondentes a cada dimensão e hierarquia (cpm). A granularidade será mais grossa se o nível de combinação dos valores das dimensões for mais alto, por exemplo "tipo de produto x zona x mês" (conforme a hierarquia dimensional mostrada na Figura 2.1), ou mais fina, por exemplo "produto x cliente x dia", em caso contrário. Um cubo de granularidade mais grossa pode, em regra, ser obtido a partir de cubos de granularidade mais fina, seguindo simplesmente as hierarquias dimensionais que, no limite, podem implicar mesmo o seu colapso (a dimensão é agregada até ao nível máximo), nenhum na Figura 2.1, cujo significado ficará claro, já de seguida. No exemplo da Figura 2.2 (à direita), a dimensão cliente é colapsada, sendo agregados os valores das células correspondentes a cada cliente x produto x mês, gerando-se o cubo produto x mês.

Pode agora clarificar-se o significado de "nenhum (-)", muitas vezes representado também como "todos", o topo de cada hierarquia. Esta dupla e antagónica designação será compreensível, pois que resulta de duas perspectivas diversas de olhar o processo de agregação. A nomeação "nenhum" deriva do facto de resultar do processo de colapso da dimensão, onde a dimensão é omitida - daí o (-) - que o representa também. Já a designação "todos" significa que todos os membros da dimensão são agregados. No contexto do presente trabalho, a designação (-) será preferida, excepto quando se pretender traduzir a noção de agregação de todos os membros de uma dimensão (Figura 2.2, à direita).

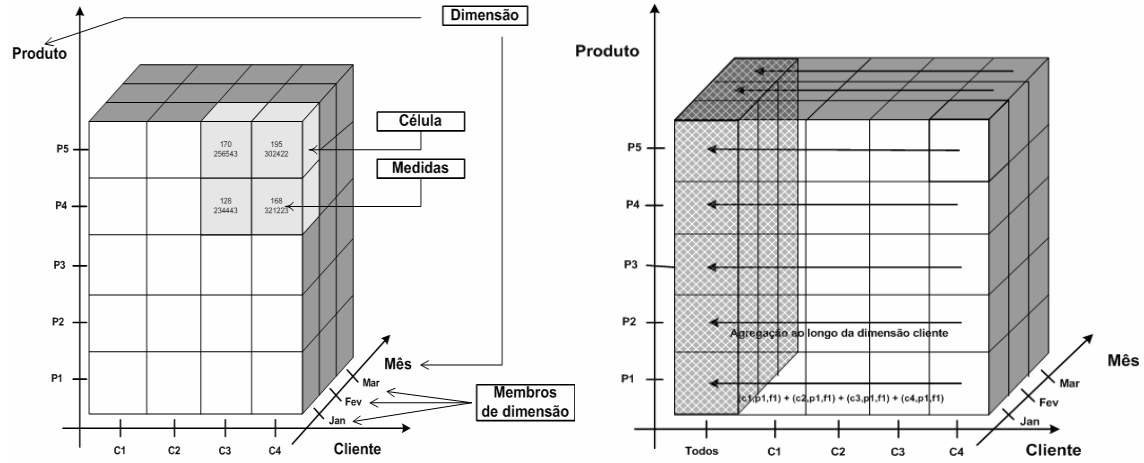


Figura 2.2. Visualização de um cubo de dados (cliente x produto x mês).

Esta possibilidade de gerar um cubo utilizando outros evidencia as relações de dependência existentes que sugeriram o denominado *lattice* de dependências, introduzido em [Harinarayan et al., 1996] (Figura 2.3), uma conceptualização de todas as agregações possíveis e respectivas dependências. No caso apresentado, são supostas três dimensões (cliente, produto e tempo) e

também a inexistência de níveis hierárquicos intermédios (por exemplo o nível cliente agrupa imediatamente no nível todos). Como há três dimensões, vão ter-se 2^3 cubos possíveis. Genericamente, o número de cubos é dado pela fórmula $\prod_{i=1}^{d} h_i$, onde h_i será o número de hierarquias da dimensão *i* e *d* o número de dimensões. Tratando-se de um produto, o número de cubos relativo a um dado esquema dimensional assume um carácter de explosão combinatória, podendo facilmente ser muito grande. Bastará ter cinco dimensões cada uma com quatro níveis hierárquicos e ter-se-á: 4x4x4x4x4=1024 cubos possíveis. Esta explosão combinatória dá origem a um problema da maior relevância em sistemas analíticos: a grande dimensão física dos dados agregados quando materializados e a necessidade absoluta de limitar o seu tamanho e, especialmente, o seu custo de actualização, sempre que tiverem de ser refrescados.

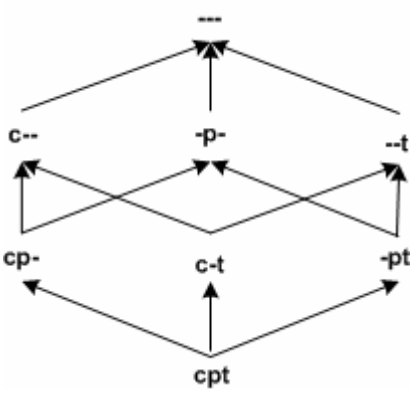


Figura 2.3. *Lattice* de dependências relativo às dimensões produto, cliente e tempo.

2.3 Utilidade e Justificação dos Cubos

Um cubo de dados, não é mais do que uma agregação de dados a um nível determinado de granularidade, entendida esta como o maior ou menor nível de detalhe. Vendo o *lattice* apresentado na Figura 2.3, a agregação (cpt) terá uma menor granularidade do que a agregação (c--). Mas, o termo cubo designa, muitas vezes, o próprio espaço multidimensional e, especialmente, o conjunto de todas as agregações possíveis, ou mesmo um seu subconjunto. Se, muitas vezes, o significado pode ser inferido do contexto em que o termo é aplicado, outras vezes, o seu uso pode suscitar algumas dúvidas e gerar várias incompreensões. Assim, e para clarificar o seu significado no contexto do presente trabalho, será usada a seguinte terminologia: cubo OLAP,

ou simplesmente cubo designará o conjunto de todos as agregações possíveis (cálculo de todos os *Group-By* possíveis) de um qualquer esquema dimensional; uma agregação será designada de subcubo, cubóide ou ainda vista; já o resultado de uma operação de restrição num subcubo será designado como fragmento (de subcubo).

A resposta a interrogações agregadas não carece, necessariamente, do cálculo e armazenamento do cubo ou parte dele. Os dados das relações base, de um maior nível de detalhe, podem ser utilizados. Contudo, implicam obviamente a sua leitura e posterior agregação, operações estas que podem ser muito demoradas, já que podem envolver muitos registos na operação, mesmo quando os índices mais convenientes estiverem disponíveis. Esta situação é, de todo, incompatível com a caracterização *on-line* do processamento analítico, condição última de produtividade dos agentes de decisão e da qualidade das respectivas decisões. Mas se uma agregação conveniente estiver já calculada e armazenada, a resposta à interrogação pode ser imediata. Atente-se na elegância da solução. Uma consulta do tipo "mostra-me as vendas totais mensais relativas ao ano de 2006" será respondida por leitura directa dos dados correspondentes ao subcubo (--t) do *lattice* apresentado na Figura 2.3 (algumas dezenas de células), que carecerá de poucas operações de restrição ou agregação (desnecessária se o mês for a granularidade considerada para a data), evitando-se assim a leitura, selecção e posterior agregação dos próprios dados base (eventualmente muitos milhões de registos). O ganho cifra-se em várias ordens de grandeza. Mesmo se o subcubo idealmente mais adaptado não estiver disponível, as dependências entre subcubos podem ser utilizadas para encontrar outro que, alternativamente, possa ser utilizado, implicando provavelmente a leitura de um maior número de células e sua posterior agregação, mas, ainda assim, a um custo definitivamente muito inferior aos custos incorridos com a utilização dos dados base. Conhecidos os subcubos materializados e seus custos, será muito simples seleccionar aquele a utilizar para gerar a resposta a uma dada interrogação, já que a minimização do tempo de resposta é, em regra, o objectivo último. O exemplo da Figura 2.4, em cujo *lattice* apenas alguns dos subcubos se encontram materializados, permite ilustrar a situação. Entre parêntesis está indicado o custo de utilização de cada subcubo. O utilizador pretende conhecer as vendas agregadas por cliente. Uma vez que o subcubo (c--) não está materializado (que implicaria um custo de 50), há que encontrar alternativas: os subcubos (c-t), (cpt) e ainda as próprias relações base, a custos 600, 6000 e 18000, respectivamente. Neste caso seria seleccionado o subcubo (c-t) para responder à interrogação, já que seria aquele que implicaria o menor custo (de entre os disponíveis). Claro que a resposta ideal a esta interrogação seria conseguida a partir de (c--) que, estando materializado, implicaria um custo de resposta bastante inferior. Mas esta

afirmação só seria verdade para esta interrogação, outra interrogação ou interrogações implicariam, para uma resposta ideal, diferentes combinações de subcubos materializados. Em resumo, pode dizer-se que a materialização do cubo (ou de uma parte) é condição de desempenho num sistema de processamento analítico e, por outro lado, que, na impossibilidade de materializar todo o cubo, o perfil de utilização pode ser utilizado para a sua optimização. Mas, como o próprio DW, esta solução tem um preço: o espaço de armazenamento, o tempo de cálculo e materialização e, especialmente, tempo de actualização, já que as alterações das relações base deverão ser reflectidas em todos e quaisquer subcubos materializados. Além disso, ocorre um problema adicional: a selecção dos subcubos mais adequados. Esta problemática irá ser discutida mais adiante, ainda neste capítulo.

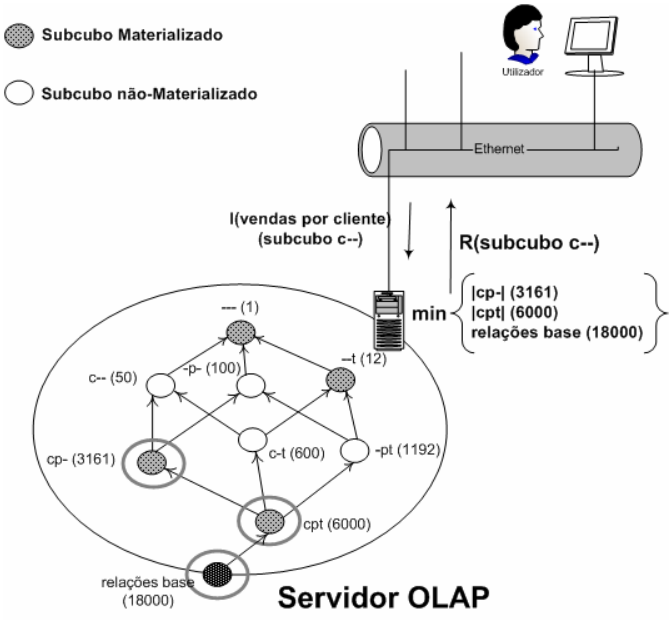


Figura 2.4. Selecção de um subcubo para resposta a uma interrogação.

2.4 Exploração de Cubos de um Cubo

Para melhor se compreender como a forma de utilização dos sistemas OLAP determina este esquema de modelação, importa descrever um exemplo típico de uma sessão de consulta, a um cubo de dados relativo a uma empresa de comercialização de veículos automóveis. O esquema multidimensional (que utiliza a notação ME/R [Sapia et al., 1999]) é mostrado na Figura 2.5, em que estão representadas as três dimensões: veículo (com hierarquias modelo → marca →

fabricante), tempo (dia → mês → trimestre → ano ou dia → ano) e localização (stande → cidade → zona geográfica → país).

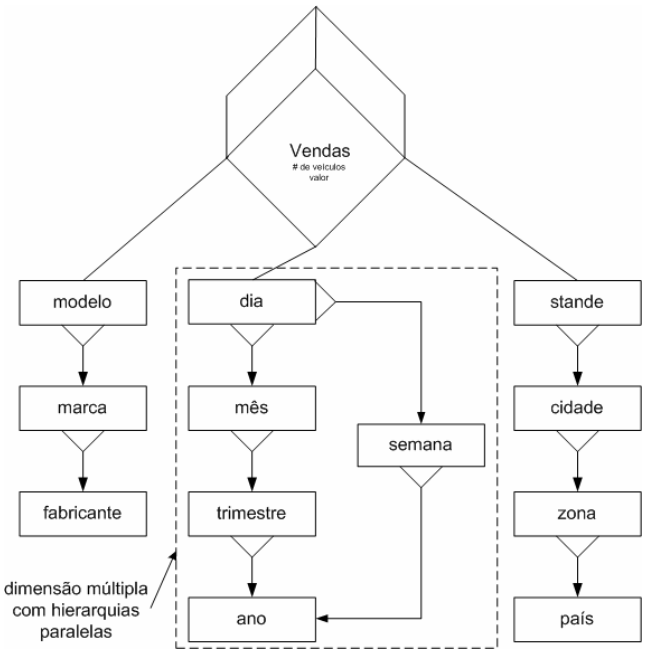


Figura 2.5. Modelo dimensional exemplificativo de um DW.

Normalmente, o utilizador inicia a sessão de consulta num ponto de entrada sob a forma de um relatório de negócio pré-definido, por exemplo, o total de vendas mensais, relativas ao primeiro trimestre de 2003, por país, dos veículos das marcas Mercedes e BMW, agrupadas por modelo. O formato da consulta está apresentado na Figura 2.6. Nela são apresentados os resultados das vendas através de uma visão multidimensional (MD), sob a forma de uma tabela - outro formato possível seria um gráfico correspondente. Algumas dimensões são restritas a um único valor (executando-se uma operação de *slicing*), no caso mostrado, a dimensão tempo: ano e trimestre. Estas dimensões podem chamar-se de "dimensões selecção" da consulta, mostradas na Figura 2.6, no canto superior esquerdo. Outros membros de dimensões (no caso, País, Marca, Modelo e Mês) vão ser mostrados distribuídos por cada um dos eixos (na tabela, linha para o primeiro, segundo e terceiro e coluna para o último), sendo então chamados de "dimensões resultado", já que o facto em análise vai ser agregado segundo os valores para cada um dos membros seleccionados nestas

dimensões e mostrado o resultado respectivo nas células da vista multidimensional, constituindo as denominadas "medidas resultado" da consulta.

Ano	2003					
Trimestre	1.º					
Sum of Veículos Vendidos			Mês			
País	Marca	Modelo	Jan	Fev	Mar	Grand Total
Espanha	BMW	320d	2144	2490	2691	7325
		525d	612	675	611	1898
		530d	1250	1403	1522	4175
		X5	902	941	951	2794
	BMW Total		4908	5509	5775	16192
	Mercedes	A170	662	740	777	2179
		C220	2072	2345	2571	6988
		C270	1553	1700	1962	5215
		SKL500	89	99	104	292
		SLK500	421	490	515	1426
	Mercedes Total		4797	5374	5929	16100
Espanha Total			9705	10883	11704	32292
Portugal	BMW	320d	156	186	202	544
		525d	46	60	63	169
		530d	8	9	10	27
		X5	49	61	62	172
	BMW Total		259	316	337	912
	Mercedes	A170	91	97	104	292
		C220	223	256	258	737
		C270	146	164	178	488
		CLK500	46	53	60	159
	Mercedes Total		506	570	600	1676
Portugal Total			765	886	937	2588
Grand Total			10470	11769	12641	34880

Figura 2.6. Resultado de uma interrogação multidimensional.

Prosseguindo a sua sessão, a interrogação seguinte será formulada, através da manipulação da estrutura da visão MD, por exemplo, fazendo a troca entre as medidas resultado e dimensão resultado, ou alterando a selecção de uma dimensão selecção. Um outro grupo de operações bastante típico consiste na alteração do nível de detalhe da consulta. Por exemplo, o utilizador, depois de conhecer as vendas num dado país, marca e mês, pode pretender conhecer com maior detalhe as vendas realizadas. Pode então efectuar uma operação denominada por *drill-down*, neste caso relativa à dimensão localização, que lhe permitirá detalhar as vendas relativas a um país, passando a exibi-las por cidade ou zona do país (Figura 2.7). Inversamente, a operação que corresponde à procura de dados mais gerais denomina-se por *roll-up*. Todas estas operações de consulta devem ser realizadas de forma rápida, dada a forma de diálogo típica da interacção de um utilizador com o sistema OLAP que se acabou de descrever.

Ano	2003						
Trimestre	1.º						
Sum of Veículos Vendidos				Mês			
País	Zona	Marca	Modelo	Jan	Fev	Mar	Grand Total
Portugal	ZC	BMW	320d	45	55	58	158
			525d	46	60	63	169
			X5	11	13	15	39
		BMW Total		102	128	136	366
		Mercedes	A170	23	24	23	70
			C220	57	61	63	181
			C270	54	63	66	183
			CLK500	27	33	39	99
		Mercedes Total		161	181	191	533
	ZC Total			263	309	327	899
	ZN	BMW	320d	54	60	66	180
			530d	4	3	4	11
			X5	12	14	12	38
		BMW Total		70	77	82	229
		Mercedes	A170	34	35	37	106
			C220	112	130	132	374
			C270	69	77	87	233
			CLK500	7	9	9	25
		Mercedes Total		222	251	265	738
	ZN Total			292	328	347	967
	ZS	BMW	320d	57	71	78	206
			530d	4	6	6	16
			X5	26	34	35	95
		BMW Total		87	111	119	317
		Mercedes	A170	34	38	44	116
			C220	54	65	63	182
			C270	23	24	25	72
			CLK500	12	11	12	35
		Mercedes Total		123	138	144	405
	ZS Total			210	249	263	722
Portugal Total				765	886	937	2588
Grand Total				765	886	937	2588

Figura 2.7. Resultado multidimensional de uma operação de *drill-down* sobre uma dimensão.

Apresenta-se seguidamente um resumo das operações OLAP mais comuns realizadas sobre um cubo com as dimensões cliente, produto e tempo. Na Figura 2.8 mostram-se as operações *roll-up* e *drill-down*, enquanto que na Figura 2.9 as operações *slice and dice* e *pivot*.

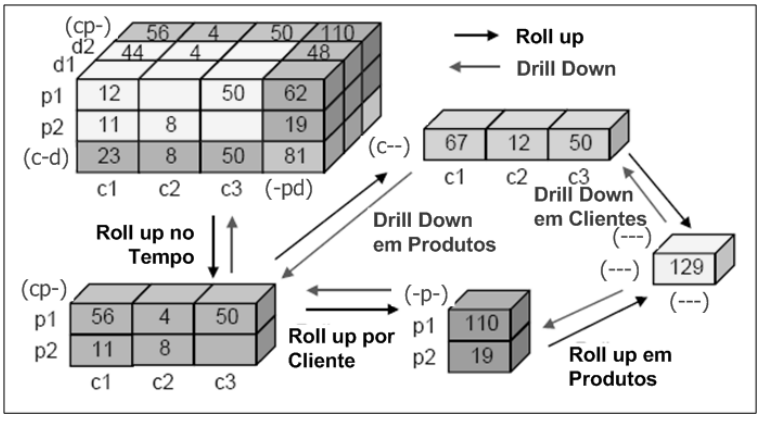


Figura 2.8. Representação das operações de *Roll-Up* e *Drill-Down* sobre um cubo.

Como se acabou de ver, o diálogo entre o utilizador e o sistema OLAP decorre de uma forma que pode ser descrita como uma sequência iterativa do tipo "uma resposta leva a uma nova questão". Assim, a demora na obtenção da resposta degrada a produtividade do utilizador. No limite, não só se assistirá a uma baixa produtividade, prejudicial para o negócio, mas a dois outros efeitos colaterais: a intercalação de outras tarefas com as operações de consulta e a consequente diminuição da concentração e capacidade de gerar novas hipóteses (com a possível perda do novo conhecimento resultante da confirmação ou negação dessas hipóteses), podendo, eventualmente, causar a simples não utilização do sistema OLAP.

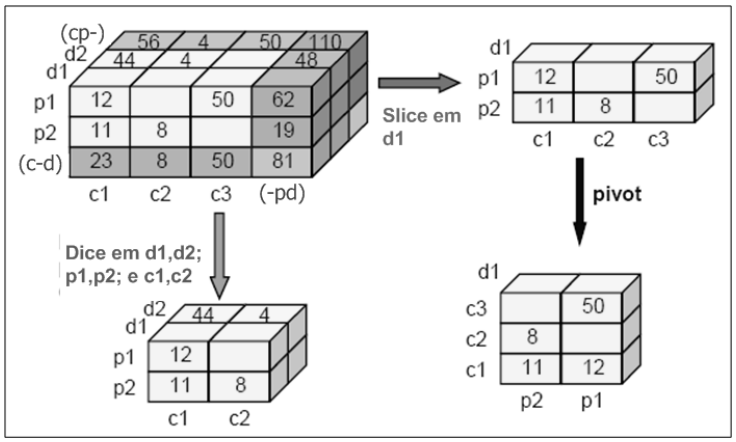


Figura 2.9. Representação das operações *Slice and Dice* sobre um cubo.

2.5 Impacto do Processamento Analítico nos Modelos de Negócio Reais

A possibilidade de navegar sobre dados correspondentes ao negócio, na sua forma nativa, confere aos agentes de decisão a capacidade de saber o que os dados efectivamente querem dizer. O exemplo apresentado na secção anterior mostra apenas um caso de utilização, não sendo, de modo algum, a única situação em que o processamento analítico pode ser utilizado. Para se perceber melhor a abrangência global do OLAP, num qualquer negócio, nada melhor do que avaliarmos a sua praticabilidade e poder em cada uma das áreas dum negócio. Para isso, será necessário, antes de mais, lançar mão de uma forma de representação conceptual de um qualquer negócio, algo que é possível utilizando os modelos introduzidos por Michael Porter [Porter, 1985].

Vai-se, aqui, mostrar a sua instanciação a um negócio de manufactura. Todavia, qualquer outro sector de actividade poderia, igualmente, servir de caso, já que este modelo é genérico.

A cadeia de valor interna (Figura 2.10), categoriza as actividades genéricas que adicionam valor numa organização. Estão divididas em "actividades primárias", aquelas directamente ligadas ao valor acrescentado criado pela unidade de negócio, tal como é vista no seu contexto de empresa pelos fornecedores e clientes, e "actividades de suporte", necessárias para facilitar, controlar e desenvolver o negócio ao longo do tempo, não adicionando valor de forma directa, mas somente através das actividades primárias.

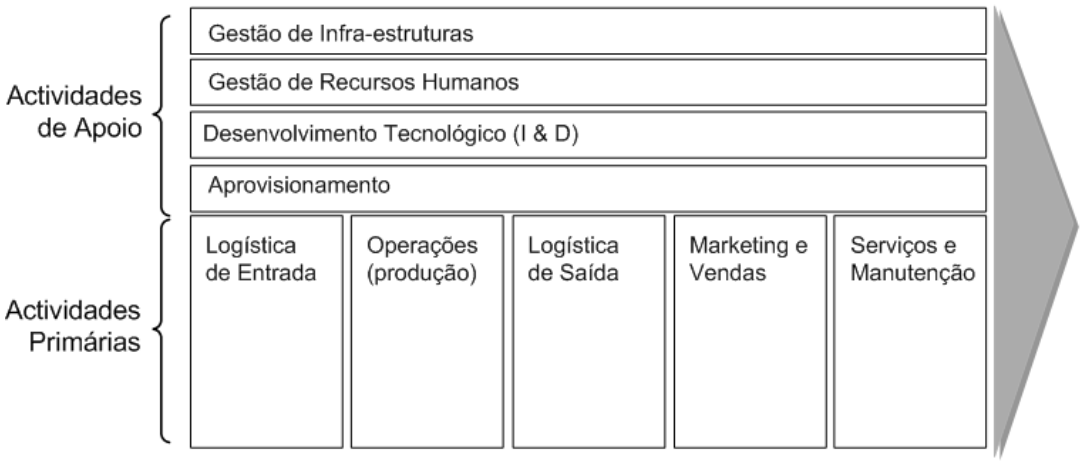


Figura 2.10. Cadeia de valor interna de uma indústria de manufactura.

A realização de cada uma das actividades gera valor, mas implica o consumo de recursos, devendo ser optimizada individualmente. Também o conjunto das suas actividades deve ser optimizado, para que o melhor desempenho total seja atingido. Assim, a identificação do valor gerado e recursos consumidos, para cada actividade, permite a prossecução do objectivo último da utilização deste modelo: a maximização da criação de valor, mantendo os custos controlados.

O conceito de cadeia de valor pode ser estendido para lá das fronteiras de cada organização individual e aplicado a toda a cadeia de aquisição e redes de distribuição. A entrega de um conjunto de produtos e serviços ao consumidor final mobiliza diversos actores, cada um gerindo a sua própria cadeia de valor. A sincronização do sector de negócio, a nível global, dessas cadeias de valor local, criam uma cadeia de valor estendida, que Porter denominou de "sistema de valor" (Figura 2.11). Imaginando o sistema de valor como um rio (cujo caudal será o fluxo de bens ou serviços, que vão sendo gradualmente transformados através da incorporação de valor em cada

um dos elos constituintes), percebe-se que este inclui as cadeias de valor dos fornecedores da empresa (e de todos os outros para montante), a própria empresa, os canais de distribuição da empresa e os clientes da empresa, estendidos estes para jusante, até aos consumidores finais.

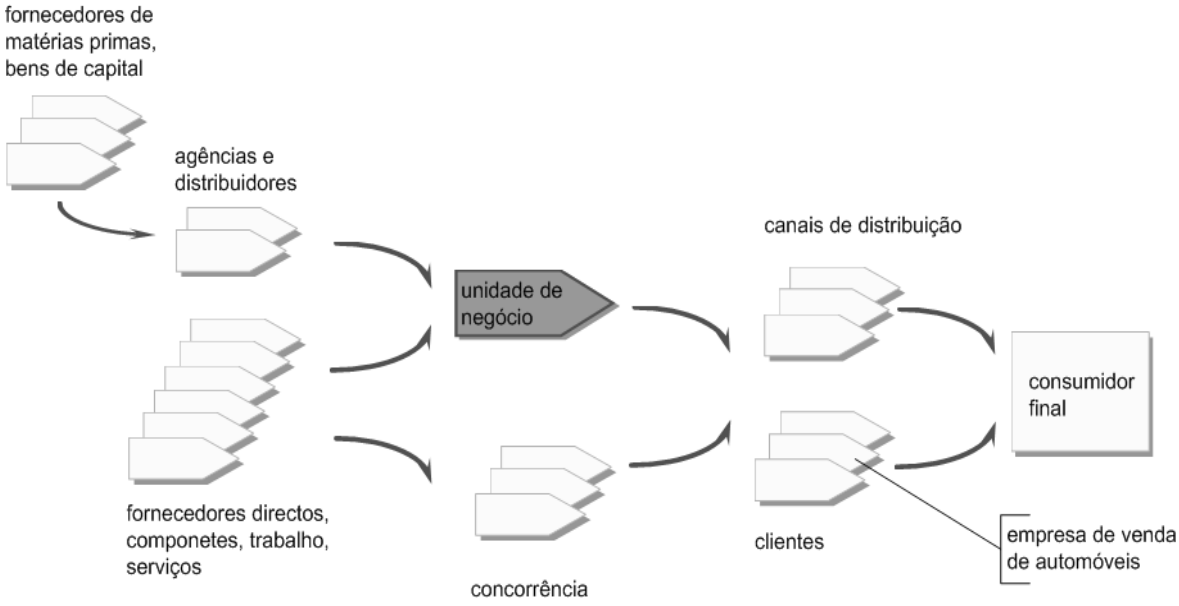


Figura 2.11. Sistema de valor de um sector de actividade de produção e comercialização de bens manufacturados.

A diferença entre o valor acrescentado e o consumo de meios será o ganho global do sector. Cada um dos intervenientes luta por uma fatia desse bolo, quer actuando internamente, quer facilitando e agilizando os processos de interface que modificam a competição e a transformam em partilha. Na realidade, o conhecimento das relações de força do sector de negócio, visto como um todo, pode permitir a adopção de estratégias que permitam uma melhoria global da sua posição (isto em termos de uma análise de forças concorrenciais [Porter, 1980], [Hunger & Wheelen, 2003]).

Os sistemas de informação, de um modo geral, podem ter um grande impacto, a vários níveis, na cadeia de valor interna e no sistema de valor. Ao nível desta última, pode actuar, por exemplo, na facilitação dos processos de interface entre os vários intervenientes (cite-se o processo de encomendas por meios automáticos, que pode propagar-se por toda a cadeia de jusante para montante). Está a falar-se, neste caso, de sistemas a nível operacional, mas a sua aplicação pode

estender-se ao nível de gestão e estratégico. É aqui que poderão actuar os sistemas de processamento analítico. A análise do comportamento de um conjunto (normalmente pequeno) de requisitos, devidamente mensurados, cuja satisfação assegure o sucesso do negócio (daí denominados de factores críticos de sucesso – (FCS)) que se traduzem em ganhos consideráveis para cada um e possivelmente para todos os intervenientes, tem uma intervenção relevante de sistemas de processamento analítico. Veja-se o exemplo da análise descrita na secção anterior. Na Figura 2.11 mostra-se o posicionamento da empresa que executa a análise no sistema de valor, neste caso ao FCS vendas de veículos, pois que a unidade de negócio em análise é uma empresa de comercialização de veículos automóveis. O conhecimento obtido relativo às vendas permitirá actuar, quer a nível de tomada de decisões com impacto na relação com os clientes (definição de novos pontos ou canais de venda), quer a nível de fornecedores, como a redução do seu poder ou partilha de informação, permitindo-lhe empreender acções atempadas no sentido de se dotarem de capacidade produtiva e de aprovisionamento adequados. Este conhecimento pode ser assim propagado para montante, a todos os elos da cadeia. Mas o impacto do OLAP pode também observar-se, e talvez com maior relevância, a nível de cada elo da cadeia do sistema de valor.

Observando a Figura 2.10, quase todas as actividades podem beneficiar do conhecimento obtido pelo processamento analítico, aqui focado na adopção de acções a nível de cada actividade e no seu relacionamento interno. O sistema OLAP de análise de vendas permite, por exemplo, empreender campanhas promocionais, novos serviços pós-venda, avaliar rotatividade de existências e sua rentabilidade; um comportamento algo anómalo pode ser um primeiro sintoma de um problema nascente ou de uma nova tendência (oportunidade). Na actividade de produção, a análise da qualidade e de interrupções na produção (devido a avarias ou descalibrações de máquinas) pode permitir a adopção de medidas pró-activas capazes de evitar ou limitar esses problemas. No sector de recursos humanos, a análise de produtividade dos funcionários e de faltas permite, por um lado, actuar no sentido de evitar o abstencionismo (medida de âmbito interno à própria actividade, embora com impactos em toda a empresa) e, por outro, uma vez que está ligada à actividade de produção, possibilita a adopção de medidas atempadas, no sentido de um planeamento mais correcto da produção.

Muitos outros exemplos poderiam ser acrescentados, já que o processamento analítico é parte integrante fundamental da denominada *Business Intelligence*. Esta pode definir-se como uma categoria ampla de aplicações e tecnologias capazes de permitir a obtenção de percepção acerca do negócio ou organização, com vista à compreensão dos activos em informação da empresa, para permitir que os decisores das empresas tomem melhores decisões. O termo implica um

conhecimento de todos os factores que afectam o negócio, sendo imperativo que o decisor tenha um profundo conhecimento acerca de factores como clientes, cadeia de fornecimento, competidores, parceiros de negócio, ambiente económico e operações internas. Veja-se, para finalizar esta secção, em que medida o processamento analítico pode contribuir para o conhecimento de cada um destes factores.

Relativamente a clientes, o processamento analítico permite conhecer os seus gostos. Há que adaptar o negócio às suas alterações: o seu conhecimento antecipado, muitas vezes conseguido através de análise aprofundada de singularidades, pode permitir o desenvolvimento de produtos ou serviços inovadores, à frente das alterações na procura dos clientes.

Quanto aos competidores, há que considerar que os objectivos são os mesmos: maximizar os lucros e a satisfação dos clientes. Para que a empresa seja bem sucedida, ela tem de estar um passo à frente dos competidores. O processamento analítico pode dizer quais as acções que os competidores estão a empreender, a fim de permitir a tomada de decisões de forma mais informada.

Os parceiros de negócio devem possuir a mesma informação estratégica, não devendo existir erros de comunicação que levem a ineficiências. Já foi discutido o impacto dos sistemas de processamento analítico na partilha de informação. Por exemplo, é agora comum permitir aos fornecedores o acesso aos níveis de inventário, métricas de desempenho e outros dados relativos à cadeia de fornecimento, de forma a colaborar na melhoria da sua gestão.

Já relativamente ao ambiente de negócio, é algo que atrás ainda não foi aflorado, quanto ao impacto possível dos sistemas de processamento analítico. Mas o estado da economia e dos seus factores chave são considerações importantes a levar em conta na tomada de decisões de negócio. O processamento analítico pode ser aqui uma grande mais-valia, já que, sendo o tempo uma dimensão omnipresente, a visualização da evolução temporal de muitos dos factores chave e a sua inspecção a vários níveis de abstracção mostram aspectos que podem ser determinantes na avaliação da oportunidade de um investimento, da expansão ou contracção das operações relativas ao negócio da empresa.

Finalmente, mas não menos importante, há um factor que é talvez aquele em que o processamento analítico terá uma maior aplicabilidade: a análise às operações internas. Estas constituem as actividades do dia-a-dia da empresa. Já se viu atrás: o processamento analítico pode aplicar-se a todas as actividades da cadeia de valor.

2.6 Materialização de Cubos

Nas secções anteriores, foi já, por várias vezes, aflorada a questão da necessidade da materialização do cubo e da impossibilidade da sua efectivação total para casos reais. Agora, vamos aprofundar a questão, procurando mostrar o problema, utilizando um caso de aplicação típico, retirado da base de dados [TPC-R, 2002], utilizada profusamente em testes de desempenho de sistemas de processamento analítico e de DW. Utilizando esse conjunto de dados teste e seleccionando a base de dados mais reduzida (1 GB) para as dimensões cliente, produto e fornecedor às quais foram adicionadas as hierarquias dimensionais cliente (c-n-r-all)[2], produto (p-t-all) e (p-s-all) e fornecedor (s-n-r-all) – ir-se-á gerar um cubo com 4x4x4=64 subcubos possíveis, uma vez que temos três dimensões, cada uma com quatro níveis na hierarquia. O cubo, de ora em diante designado por cubo A, está apresentado na Tabela 2.1, bem como o seu tamanho (em tuplos ou células).

Tabela 2.1. Cubo A: subcubos gerados com as dimensões cliente, produto e fornecedor.

Subcubo		Tamanho	Subcubo		Tamanho	Subcubo		Tamanho	Subcubo		Tamanho
1	cps	6,000,000	17	nps	5,000,000	33	rps	4,000,000	49	-ps	800,000
2	cpn	6,000,000	18	npn	5,000,000	34	rpn	4,000,000	50	-pn	800,000
3	cpr	6,000,000	19	npr	5,000,000	35	rpr	4,000,000	51	-pr	800,000
4	cp-	6,000,000	20	np-	5,000,000	36	rp-	1,000,000	52	-p-	200,000
5	css	5,000,000	21	nss	500,000	37	rss	2,500,000	53	-ss	500,000
6	csn	5,000,000	22	nsn	30,000	38	rsn	6,250	54	-sn	1,250
7	csr	5,000,000	23	nsr	6,250	39	rsr	1,250	55	-sr	250
8	cs-	5,000,000	24	ns-	1,250	40	rs-	25	56	-s-	5
9	cts	5,990,000	25	nts	800,000	41	rts	3,000,000	57	-ts	1,500,000
10	ctn	5,990,000	26	ntn	90,000	42	rtn	18,750	58	-tn	3,750
11	ctr	5,990,000	27	ntr	18,750	43	rtr	3,750	59	-tr	750
12	ct-	5,990,000	28	nt-	3,750	44	rt-	750	60	-t-	150
13	c-s	6,000,000	29	n-s	250,000	45	r-s	50,000	61	--s	10,000
14	c-n	2,500,000	30	n-n	625	46	r-n	125	62	--n	25
15	c-r	500,000	31	n-r	125	47	r-r	25	63	--r	25
16	c--	100,000	32	n--	25	48	r--	5	64	---	1

Este modelo dimensional é bastante simples. Procurou-se gerar um outro, um pouco mais complexo, a partir deste, pela adição de uma nova dimensão: tempo e respectiva hierarquia dimensional dia(d)-mês(m)-ano(y)-all. O tamanho do novo cubo OLAP supõe um ano de dados e as probabilidades de gerar uma célula são mostradas na Tabela 2.2.

[2] As iniciais dos nomes das hierarquias dimensionais em inglês foram mantidos, para manter a consistência com a norma TPC-R original, neste caso (costumer-nation-region-all), (product-type-all) e (product-size-all), (supplier-nation-region-all).

Tabela 2.2**.** Probabilidade de geração de células relativas à dimensão tempo.

Hierarquias	Subcubos Existentes no Lattice			
Nova Dim.	nível-baixo	1 Dim. Colapsada	2 Dim. Colapsada	Todas Dim. Colapsadas
d(day)	20%	40%	60%	100%
m(month)	60%	80%	90%	100%
y(year)	90%	95%	100%	100%
- (all)	100%	100%	100%	100%

Baseado nestas suposições, gerou-se um novo cubo com 256 subcubos (64x4 níveis da dimensão tempo), designado como cubo B (Tabela 2.3).

Os cubos A e B vão ser utilizados não só na discussão que se segue, mas também nos testes dos algoritmos desenvolvidos no âmbito do presente trabalho e a descrever em capítulos futuros.

Tabela 2.3. Cubo B: gerado adicionando a dimensão tempo ao cubo A.

Subcubo		Tamanho	Subcubo		Tamanho	Subcubo		Tamanho	Subcubo		Tamanho
0	cpsd	438,000,000	64	cpsm	43,200,000	128	cpsy	5,400,000	192	cps-	6,000,000
1	cpnd	438,000,000	65	cpnm	43,200,000	129	cpny	5,400,000	193	cpn-	6,000,000
2	cssd	365,000,000	66	cssm	36,000,000	130	cssy	4,500,000	194	css-	5,000,000
3	ctsd	437,270,000	67	ctsm	43,128,000	131	ctsy	5,391,000	195	cts-	5,990,000
4	npsd	365,000,000	68	npsm	36,000,000	132	npsy	4,500,000	196	nps-	5,000,000
5	cprd	438,000,000	69	cprm	43,200,000	133	cpry	5,400,000	197	cpr-	6,000,000
6	csnd	365,000,000	70	csnm	36,000,000	134	csny	4,500,000	198	csn-	5,000,000
7	ctnd	437,270,000	71	ctnm	43,128,000	135	ctny	5,391,000	199	ctn-	5,990,000
8	c-sd	876,000,000	72	c-sm	57,600,000	136	c-sy	5,700,000	200	c-s-	6,000,000
9	npnd	365,000,000	73	npnm	36,000,000	137	npny	4,500,000	201	npn-	5,000,000
10	nssd	36,500,000	74	nssm	3,600,000	138	nssy	450,000	202	nss-	500,000
11	ntsd	58,400,000	75	ntsm	5,760,000	139	ntsy	720,000	203	nts-	800,000
12	rpsd	292,000,000	76	rpsm	28,800,000	140	rpsy	3,600,000	204	rps-	4,000,000
13	cp-d	876,000,000	77	cp-m	57,600,000	141	cp-y	5,700,000	205	cp--	6,000,000
14	csrd	365,000,000	78	csrm	36,000,000	142	csry	4,500,000	206	csr-	5,000,000
15	ctrd	437,270,000	79	ctrm	43,128,000	143	ctry	5,391,000	207	ctr-	5,990,000
16	c-nd	365,000,000	80	c-nm	24,000,000	144	c-ny	2,375,000	208	c-n-	2,500,000
17	nprd	365,000,000	81	nprm	36,000,000	145	npry	4,500,000	209	npr-	5,000,000
18	nsnd	2,190,000	82	nsnm	216,000	146	nsny	27,000	210	nsn-	30,000
■ ■ ■											
46	r-nd	18,250	110	r-nm	1,200	174	r-ny	118	238	r-n-	125
47	-prd	116,800,000	111	-prm	7,680,000	175	-pry	760,000	239	-pr-	800,000
48	-snd	182,500	112	-snm	12,000	176	-sny	1,187	240	-sn-	1,250
49	-tnd	547,500	113	-tnm	36,000	177	-tny	3,562	241	-tn-	3,750
50	--sd	21,900,000	114	–sm	1,080,000	178	--sy	100,000	242	--s-	100,000
51	n--d	5,475	115	n--m	270	179	n–y	25	243	n---	25
52	rs-d	3,650	116	rs-m	240	180	rs-y	23	244	rs--	25
53	rt-d	109,500	117	rt-m	7,200	181	rt-y	712	245	rt--	750
54	r-rd	3,650	118	r-rm	240	182	r-ry	23	246	r-r-	25
55	-p-d	43,800,000	119	-p-m	2,160,000	183	-p-y	200,000	247	-p--	200,000
56	-srd	36,500	120	-srm	2,400	184	-sry	237	248	-sr-	250
57	-trd	109,500	121	-trm	7,200	185	-try	712	249	-tr-	750
58	--nd	5,475	122	--nm	270	186	--ny	25	250	--n-	25
59	r--d	1,095	123	r–m	54	187	r--y	5	251	r---	5
60	-s-d	1,095	124	-s-m	54	188	-s-y	5	252	-s--	5
61	-t-d	32,850	125	-t-m	1,620	189	-t-y	150	253	-t--	150
62	--rd	5,475	126	--rm	270	190	--ry	25	254	--r-	25
63	---d	365	127	––m	12	191	---y	1	255	----	1

Mas volte-se à questão central desta secção: se a disponibilização das agregações multidimensionais é interessante, porque não disponibilizá-las na sua totalidade?

É necessário olhar este problema de duas perspectivas:

- de um lado, os custos de interrogação (Ci);
- do outro, os custos de materialização e manutenção das agregações materializadas (*Cm*).

O primeiro custo é monotónico relativamente às agregações disponíveis: dado um conjunto I de interrogações e um conjunto M de agregações materializadas, a adição de uma nova agregação terá um efeito, em regra, benéfico para o custo de interrogação (no máximo não terá qualquer efeito, se a nova agregação não for utilizada por qualquer interrogação em I), mas nunca poderá aumentar o custo. Ou seja, os custos serão tão mais pequenos quanto mais pré-agregações adequadas estiverem disponíveis.

A materialização e manutenção das estruturas importa em espaço e tempo. Vimos já que o número de subcubos possíveis pode tornar-se muito grande, mas é relevante mostrar também a dimensão real de um cubo. Mesmo considerando o cubo A, bastante simples em termos de complexidade dimensional (e o mais pequeno), considerando a base de dados TPC-R, temos um número total de tuplos de 122.097.911. Para o cubo B, o número ascende a 12.853.175.791, e ainda estamos perante um esquema simples e uma base de dados pequena. Facilmente se percebe que, para dimensões dos dados base maiores e maior complexidade dimensional, o número de registos torna-se completamente incomportável. Mesmo sabendo da natureza "ambiente de só-leitura" de um sistema de DW (SDW), há que considerar as actualizações necessárias, já que as relações base são alteradas, em resultado das transacções e eventos ocorridos no mundo real, sendo necessário que essas alterações sejam reflectidas por toda a cadeia hierárquica de agregações, de forma a manter a consistência e evitar-se a obtenção de informação desactualizada.

Reflictamos na natureza dos custos envolvidos na materialização do cubo. Neste caso, ter-se-á:

1. o espaço para a sua efectivação;
2. o tempo de materialização inicial ou reconstrução; e
3. o tempo de actualização (refrescamento).

O primeiro é de natureza monotónica: mais agregações implicam sempre um maior espaço de armazenamento ocupado; cada nova actualização implica não só novos tuplos nas relações base, mas também muitos outros nos subcubos materializados, já que a dimensão tempo é

omnipresente, excepto se a granularidade mais fina considerada não for muito pequena (por exemplo mês). Os custos restantes são não-monotónicos [Bauer & Lehner, 2003]: a adição de uma nova agregação a um conjunto *M* prévio pode implicar uma diminuição de custos. Suponha-se a situação mostrada na Figura 2.12. Para simplificar, considere-se que a manutenção é realizada por geração integral dos subcubos a materializar. Vamos calcular o custo de cada distribuição M e M' (correspondente aos cenários 1 e 2, respectivamente).

O custo de materialização de M é de: 18000 (subcubo 1) + 3 x 6000 (subcubos 2, 3, e 4) + 12 (subcubo 5) = 36012; enquanto que o custo de materialização de M' é de: 18000 (subcubo 1) + 2 x 6000 (subcubos 2 e 4) + 2 x 600 (subcubos 3 e 5) + 12 (subcubo 6) = 31212 é menor do que o valor anterior, apesar de M'=M+(c-t).

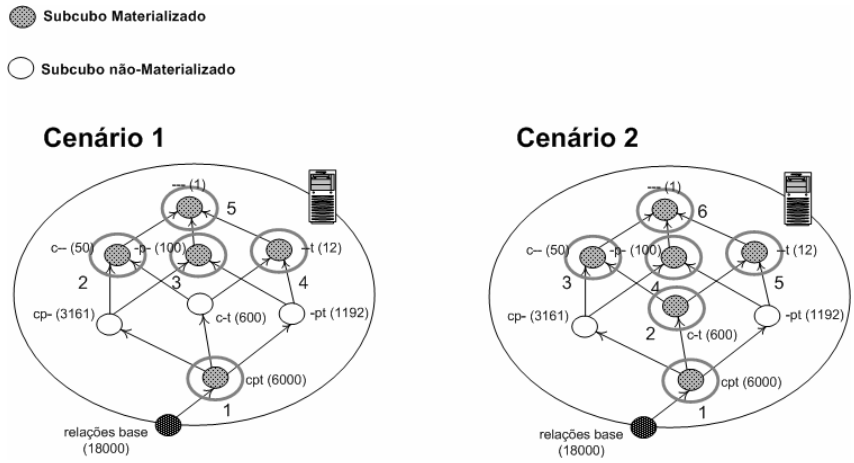


Figura 2.12. Cálculo de custos de manutenção para duas distribuições *M* de subcubos.

Este exemplo mostra que o custo de materialização é claramente não-monotónico. O custo de materialização é equivalente ao custo de manutenção por recriação dos subcubos a materializar, que podemos denominar como manutenção integral. O mesmo tipo de considerações e cálculos poderiam ser aplicados para o caso da manutenção incremental [Gupta, A. et al., 1993], [Griffin & Likkin, 1995], [Zhuge et al., 1995], onde podem ser aplicadas técnicas de manutenção incremental (por aplicação dos denominados deltas [Mumick et al., 1997]) para actualizar as vistas materializadas, sempre que as relações base são modificadas em resultado das alterações havidas nos sistemas operacionais ou outras fontes externas, alterações estas resultantes dos eventos ocorridos no ambiente de negócio. Em regra, os custos de manutenção incremental são bastante

mais pequenos do que os custos de manutenção integral. A razão é simples: as modificações têm um impacto limitado em muitos dos subcubos materializados.

Esta análise custo/benefício dos diversos factores intervenientes mostra que estamos perante um problema de natureza complexa. Em resumo: a natureza de *I* determina a utilidade das agregações em *M*; Ci(I, M) é monotónico com *M*; Cm(*M*) é não monotónico com *M*. Assim, dados os custos e o seu comportamento, importa, dado *I*, seleccionar as agregações *M* cuja existência se revelar mais benéfica: é o denominado problema de selecção de vistas ou subcubos a materializar.

2.7 Selecção de Subcubos: Optimização por Cálculo Simulado de Custos

A procura de soluções para o problema da selecção de vistas ou subcubos a materializar deverá assumir um carácter necessariamente aproximado e simulado. Senão, vejamos: o problema, já caracterizado como *NP-hard*, é de natureza combinatória, já que uma qualquer vista ou subcubo pode estar ou não materializado em *M*. Se o número de subcubos possíveis é muito grande, o número das suas possíveis combinações é muito maior. Além disso, mesmo considerando o cubo A, o número de possíveis *M* (subconjuntos de subcubos materializados) será de 2^{64}=1.84467E19. A procura da solução por pesquisa exaustiva do espaço de soluções implicaria um número igual de iterações, cada uma obrigando a um cálculo de custo. Mesmo supondo que cada iteração tivesse uma duração de 1 µs, o processo de optimização teria a duração de 584542 anos. Desta forma, pode concluir-se que o processo de optimização deverá procurar as zonas mais promissoras do espaço de soluções, para que assim possa ter um tempo de execução realizável. No entanto, este implica, em regra, um número relativamente elevado de iterações, o que introduz a questão do carácter simulado do cálculo de custos.

Para exemplificar, considere-se o caso do cálculo dos custos de interrogação. Para um conjunto de interrogações *Q*, sabe-se que o custo dependerá de *M*. Os custos de geração de um qualquer *M* são muito elevados, já que, mesmo considerando o cubo A (mostrado na Tabela 2.1), ter-se-á um número total de tuplos de cerca de 122 milhões. Assumindo como valor do número de tuplos em *M*, ½ do valor máximo, ter-se-iam ainda cerca de 61 milhões. Mesmo supondo que os custos de materialização fossem apenas os ligados ao armazenamento das vistas (o que não é verdade, pois que haverá a considerar os custos de busca e agregação) e que tivéssemos disponível um servidor

e base de dados capaz de armazenar 100000 registos.s^{-1}, teríamos um tempo de materialização de 610s, cerca de 10 minutos. Se fossem considerados os valores de busca e agregação, certamente maiores, já que a geração de um subcubo implica a utilização de um subcubo algo maior (suponha-se três vezes maior, em média), ter-se-iam mais 30 minutos, perfazendo um total de 40 minutos.

Considere-se um algoritmo simples de optimização: "*hill climbing*". Por ora, descreve-se o seu funcionamento de forma básica: uma população de *hill climbers* percorre o espaço de soluções, sendo permitidos movimentos que resultem em localizações de custo inferior. Diga-se, para simplificar, que cada *hill climber* procura um vale (mínimo) no espaço de soluções. Trata-se de um método de optimização por procura aleatória e bastante ineficiente, pois que se um *hill climber* cair num mínimo local, jamais conseguirá sair de lá e continuar a procura de uma melhor localização. Mas, para a discussão presente, é suficiente. Vamos supor que temos uma população de vinte *hill climbers* e que, para conseguirmos uma solução minimamente aceitável, são executadas 1000 iterações. Em cada iteração teremos vinte propostas de solução (uma distribuição *M* de subcubos a materializar), perfazendo um total de 20000 (1000x20) soluções. Cada uma destas implica um cálculo de custo, que permite aceitar ou não a deslocação do *hill climber* e avaliar a qualidade da solução. Ora, supondo as 20000 avaliações e considerando o custo de 40 minutos de materialização real, teríamos um valor de 555 (40x20000=800000min) dias! Pode concluir-se que o cálculo experimental dos custos de materialização de forma real é, simplesmente, inexequível. Desta forma, a simulação torna-se a única forma viável, desde que haja disponível um modelo adequado que permita emular o comportamento e as variáveis existentes no sistema real. Na posse deste modelo, poder-se-ão deduzir as fórmulas de cálculo de custos e, com elas, estimar os custos das simulações de *M* (soluções) propostas por um qualquer algoritmo de optimização.

As soluções serão obrigatoriamente aproximadas, dado o já discutido carácter explosivo do número de combinações de *M* possíveis, que inviabiliza uma pesquisa exaustiva. Em resumo, estamos assim perante um problema claro de optimização, onde temos um conjunto *S* de subcubos possíveis do cubo *C*, e relações de dependência *D* entre subcubos do *lattice*. Dado um conjunto de interrogações agregadas *I*, caracterizadas por uma frequência *f*, importa encontrar o subconjunto $M \in S$ que minimize o custo de interrogação *Ci*, que não será mais do que o somatório do custo de resposta a cada uma das interrogações $i \in I$, conforme mostrado na eq. 2.1, importando encontrar um *M* que minimize este custo.

$$Ci(I,M) = \sum_{i \in I} C(i,M).f_i \qquad \text{(eq. 2.1)}$$

Contudo, *M* importa em dois custos: espaço e tempo. O primeiro actua, de imediato, como um constrangimento ao processo de optimização. Considerando $T(s_i)$ o tamanho (em registos) do subcubo s_i, então o constrangimento anterior poderá ser definido como $\sum_{s_i \in S} T(s_i) \leq T_l$, em que T_l será o espaço de materialização disponível. Quanto ao segundo custo, pode ser tratado como um custo adicional ao custo de interrogação ou também como um constrangimento. Na primeira situação, calculado o custo utilizando a eq. 2.2, é utilizado em conjunto com o custo de interrogação, sendo procurada a minimização da soma dos dois custos. Já na segunda situação, um $M : Cm(M) > Cml$ (onde Cml será o custo de manutenção máximo admissível) deverá ser recusado ou tratado de outro modo, já que existe, neste caso, um constrangimento ao processo.

$$Cm(M) = \sum_{s_i \in M} Cm(s_i) \qquad \text{(eq. 2.2)}$$

Estes considerandos têm um reflexo imediato no processo de optimização e consequentes algoritmos. A primeira situação (a minimização conjunta de *Ci* e *Cm*) é, à partida, mais simples: uma comparação sucessiva de custos, procurando-se sempre um menor; já um constrangimento (normalmente resultante de uma imposição física de carácter restritivo), implica

1. a recusa da solução faltosa (ou a sua inadmissibilidade),
2. a sua correcção ou
3. a imposição de uma penalização.

No caso presente, a abordagem de tipo 1 implica que, a) a cada solução encontrada, seja efectuada a sua verificação e consequente não admissibilidade ou substituição, no caso de ser verificada a sua inconformidade face aos constrangimentos ou b) que o próprio processo de procura de soluções contenha em si mesmo uma qualquer heurística que evite a geração de soluções inválidas. Já numa abordagem do tipo 2, depois de gerada a solução, se verificada a sua inconformidade, importa disponibilizar uma heurística que permita a eliminação sucessiva de subcubos, até que a conformidade seja atingida.

Finalmente, a última abordagem: a imposição de uma penalização. Trata-se de um esquema só utilizável em técnicas de optimização onde se efectue a avaliação de uma função de elegância que

determine, de algum modo, o processo de optimização. É o caso, por exemplo, dos algoritmos genéticos, em que o processo de selecção é guiado pela adaptação ou elegância de cada solução (neste caso, o custo). Se uma dada solução violar um constrangimento imposto, é-lhe simplesmente aplicada uma coima, não mais do que um valor de um custo extra, que irá penalizar a sua adaptação e, assim, prejudicar a sua probabilidade de selecção e consequente transmissão de algumas das suas características à próxima geração, ainda que mantendo alguma possibilidade de isso acontecer, o que permite que algo de bom contido na solução não seja definitivamente perdido.

2.8 Caracterização das Soluções de Optimização de Estruturas Multidimensionais de Dados

O processo de optimização descrito de forma genérica na secção anterior inclui já o objectivo, o modelo, a lógica de selecção e restrições. Se fosse pretendido, poder-se-ia, decerto, efectuar uma classificação das propostas de solução para o problema da selecção de cubos segundo estes quatro referenciais. Contudo, estar-se-iam a esquecer duas dimensões, porventura mais relevantes, já que omnipresentes em quaisquer classificações das actividades humanas: o tempo e o espaço. Discuta-se, de imediato, a primeira.

Se há algo que, cada vez mais, parece ser inevitável, é a mudança. Inexoravelmente, o perfil de necessidades dos utilizadores altera-se, implicando uma progressiva desadequação dos subcubos materializados à carga de interrogações colocadas. Mas, por outro lado, já se sabe que a geração destas estruturas importa em custos muito elevados, especialmente se implicar uma recriação extensiva. Se, idealmente, este ajuste deveria ser efectuado a intervalos pequenos, é bom não esquecer que os custos de rematerialização serão incorridos a cada ajuste. Mesmo que intervalos mais curtos signifiquem modificações de extensão mais limitada, se subcubos de grandes dimensões estiverem em consideração, ter-se-ão custos muito elevados, porventura muito para além das janelas temporais de actualização disponíveis. Por outro lado, quando se utilizam perfis instantâneos, pode estar a incorrer-se num erro grosseiro: tentar uma sobreadaptação do sistema, procurando seguir tendências instantâneas e não perfis estabilizados e correspondentes a necessidades de longo prazo. As considerações anteriores não são válidas se o acompanhamento das necessidades de resposta a interrogações não implicar custos adicionais ou estes forem muito reduzidos. Mesmo que se verifique a sobreadaptação e as modificações não se revelem profícuas, as perdas não terão sido grandes.

Até ao momento, consideraram-se, na presente discussão acerca da temporalidade da recalibração das estruturas multidimensionais de optimização, duas variantes possíveis: intervalos algo alargados ou muito pequenos (quase em tempo real), ou seja $t \gg 0$ e $t \approx 0$; pensando no eixo de tempo de uma possível representação gráfica das soluções para o problema de selecção de subcubos, há ainda a temporalidade $t < 0$, implicando a introdução de um componente especulativo, em que se vai procurar prever as necessidades futuras, adequando atempadamente as estruturas multidimensionais a esse perfil de interrogações antevisto.

Em resumo, em termos da caracterização das soluções para ajuste das estruturas multidimensionais à alteração do perfil de interrogações, ter-se-á de considerar uma tripla abordagem:

1. As denominadas soluções estáticas, com $t \gg 0$, onde as recalibrações irão ocorrer a intervalos alargados. A nomeação "estática" (e.g. [Harinarayan et al., 1996], [Gupta & Mumick, 1999], [Lin & Kuo, 2004]) mostra a relativa estabilidade das estruturas nesta abordagem: só a intervalos longos serão aplicadas reconfigurações. Esta é a solução típica em estruturas multidimensionais de grandes dimensões, já que, como foi referido, implica custos muito elevados. Podem também classificar-se como abordagens reactivas, já que se assiste a uma reacção desfasada temporalmente em face das alterações do perfil de utilização.
2. As soluções dinâmicas, com $t \to 0$, pretendem acompanhar de perto as alterações do perfil de interrogações (e.g. [Scheuermann et al., 1996], [Kotidis & Roussopoulos, 1999]). Podem também denominar-se de soluções activas, já que se assiste à acção (adaptação) quase imediata em face das necessidades. Já foi referido que esta solução é praticável apenas quando os custos de reconfiguração são baixos:
   a) Quando são armazenados os resultados de interrogações colocadas pelos utilizadores, procurando a sua reutilização futura; neste caso os custos de manutenção são baixos, já que a geração das agregações é conseguida a custo zero (pois que aproveitados os incorridos na resposta às próprias interrogações). O esquema implica normalmente a utilização de uma *cache* na qual são armazenados os fragmentos de cubo, utilizando uma política de admissão e substituição de fragmentos, construída normalmente à volta de heurísticas de tipo temporal e/ou espacial.
   b) Se a reconfiguração for de extensão muito limitada, utilizando alguma "folga" na janela de refrescamento (sempre que o tempo total de manutenção é inferior ao

tempo máximo de manutenção admissível). Neste caso, esse tempo pode ser utilizado para geração de algum(ns) novo(s) subcubo(s) que o algoritmo de optimização mostre(m) ser adequado(s), atendendo à variação do perfil de interrogações.

3. As soluções pró-activas (e.g. [Belo, O., 2000], [Sapia, 2000], [Park et al., 2003]) com $t < 0$, pretendem não só uma reacção quase em tempo real, mas estar à frente no tempo: preparar as estruturas multidimensionais de optimização tendo em conta as necessidades futuras. Também este domínio de soluções conhece variantes:

   a) Utilizando caches com pré-busca, de proactividade temporal curta, procurando, por predição de uma ou duas interrogações em avanço, lançá-las antecipadamente e armazenar o resultado. Se a predição se revelar correcta, o sistema fornecerá imediatamente a resposta. Se a taxa de acertos for elevada, a melhoria sentida nos custos de interrogação poderá ser grande.

   b) Outras soluções que permitem que os algoritmos de admissão e substituição de caches incluam, nas suas heurísticas, uma avaliação da utilidade futura dos subcubos ou fragmentos, ou que façam uma análise prévia dos dados multidimensionais que permita avaliar da sua provável utilidade e propor possíveis interrogações aos utilizadores.

Quanto à segunda dimensão, ainda não discutida, é talvez de abordagem mais simples. A dimensão espacial das soluções de materialização dos cubos pode apenas assumir duas variantes: centralizada e distribuída, podendo esta última conhecer uma evolução no sentido de uma distribuição em larga escala.

A maioria das propostas de solução para o problema de selecção de cubos ou vistas a materializar endereça um repositório centralizado. É a solução clássica. Um cubo cuja distribuição *M* é armazenada num servidor e alvo dos posteriores processos de refrescamento. Fazendo um mapeamento na dimensão discutida anteriormente, dir-se-ia que a perspectiva estática será aqui território de eleição.

Já a distribuição das estruturas pode conhecer variantes, algumas consequência das soluções abordadas quanto à dimensão anterior. A abordagem activa e algumas pró-activas impõem a utilização de cache, que pode residir num ou vários servidores. De qualquer maneira, uma cache constituir-se-á quase sempre como um repositório mais, o que significa, de imediato, uma distribuição (ainda que limitada) das estruturas multidimensionais. Mas, também, facilmente se

percebe que esta solução pode ser estendida, considerando, não uma, mas várias caches, algo proposto em [Kalnis & Papadias, 2001]. Há apenas a considerar os custos de comunicação e a distribuição das interrogações no modelo de custos e prover a um mecanismo de admissão e exclusão de cache centralizado, semi-centralizado ou completamente distribuído [Kalnis & Papadias, 2001], [Stonebraker et al., 1994].

Considerando a abordagem mais tradicional, a materialização do próprio cubo pode ser feita, não num, mas em vários servidores. Estaremos perante a arquitectura já atrás denominada como M-OLAP, e que será abordada mais detalhadamente na próxima secção.

Esta proposta não será mais do que a migração das soluções de bases de dados distribuídas algo vulgares, com provas dadas em ambientes operacionais. Para ambientes informacionais, a sua aplicação será mais simples, já que estamos perante um ambiente de utilização dual e não interceptável, sendo dominante a utilização em ambiente de só-leitura. Todos os esquemas que asseguram a consistência da base de dados em presença de transacções distribuídas[3] são aqui desnecessários. As vantagens deste ambiente são conhecidas: capacidade de crescimento de armazenamento e processamento sustentado, maior disponibilidade a custos controlados. São consequência da conhecida relação de preço x desempenho: é mais barato um conjunto de vários servidores pequenos do que um servidor grande com a mesma capacidade agregada[4]. Uma solução M-OLAP implica, é claro, um aumento da complexidade da administração: há que lidar com uma nova dimensão – espaço. O clássico problema da selecção de cubos transforma-se agora no problema de selecção e alocação dos cubos. Não é suficiente seleccionar os cubos mais convenientes, mas importa também materializá-los no(s) nó(s) mais apropriado(s). Mas, se heurísticas ou algoritmos capazes de lidar com o novo problema estiverem disponíveis, as vantagens poderão ser muitas e a distribuição pode ser extensiva: uma sala ou campus cujos servidores estariam ligados por uma LAN (eventualmente de alta velocidade); ou ainda em várias destas instalações, e mesmo cubos residentes em clientes numa base *peer-to-peer* interligados por WAN heterogéneas [Kalnis et al., 2002a]. As consequências do aumento de complexidade de uma arquitectura deste tipo não serão dramáticas. Há apenas a considerar os efeitos dos custos de

[3] Nomeadamente aqueles que asseguram o critério da seriebilidade [Ozsu & Valduriez, 1997], que leva em consideração a propriedade do isolamento das transacções, assegurada pelo protocolo de bloqueio em duas fases, de que existem muitas variantes [Abbot & Garcia-Molina, 1992], [Lam & Hung, 1995], [Ulusoy & Belford, 1993].

[4] Uma consulta aos Benchmarks TPC-H [www.[7]], mostra que, por exemplo, a relação preço/QphH: 1) do servidor SunFire X4100 (o melhor para bases de dados de 100GB) é de 4.61 US$, 2) para o SunFire X4200 (o melhor para uma base de dados de 300 GB) é de 6.29 US$, 3) para o HP ProLiant DL585G1 (o melhor para bases de dados de 1000GB) é de 13.85 US$, 4) para o IBM eServer xSeries 346 (o melhor para bases de dados de 3000 GB) é já de 32.34 US$, finalmente 5) para

comunicação relativos às várias redes em presença e valores de capacidade de processamento e armazenamento de cada nó, algo que deverá ser reflectido no modelo de custos e equações respectivas, utilizados depois pelos algoritmos de estimativa de custos, que deverão incluir também considerandos relativos à paralelização de tarefas, já que a arquitectura M-OLAP é inerentemente paralela. Em termos da nossa classificação, poderemos considerar soluções centralizadas, distribuídas e fortemente distribuídas.

Para finalizar, importa sistematizar as propostas segundo a caracterização multidimensional apresentada. Não se pretende uma mostra exaustiva de todas as soluções, mas apenas referir as mais representativas, pelo seu grau de inovação quando foram apresentadas, pela sua importância ao abrirem uma nova linha de investigação ou família de soluções ou, ainda, pela sua eficácia na actualidade. Ainda assim, dada a extensão do número de soluções e número de dimensões identificadas, decidiu-se pela sua apresentação sob uma forma dupla, tabular e gráfica multidimensional, já que cada uma encerra virtudes e limitações. Esta apresentação resumida é depois complementada pela descrição algo pormenorizada das propostas reputadas como mais relevantes.

A representação tabular, dada a sua extensão, é apresentada em Anexo 1. Inclui as dimensões referidas atrás e, como tal, é mais exaustiva. Cada linha da tabela mostra uma dada solução, caracterizada segundo as dimensões, referindo o artigo e as características mais relevantes da proposta. Devido à exiguidade do espaço, empregar-se-ão abreviaturas para alguns membros das dimensões. Como dimensões, ter-se-á:

1. Objectivo, o alvo do processo de optimização: o custo de resposta às interrogações surge em destaque (minimização do custo de interrogações – $Min(Ci)$), pois que directamente relacionado com a satisfação dos utilizadores; também, e atendendo a que o tempo de refrescamento deve ser desejavelmente o mais curto possível, a soma dos custos de interrogação e manutenção pode constituir-se também como um objectivo mais alargado ($Min(Ci + Cm)$).
2. Lógica de selecção, relativa às heurísticas e outras lógicas, consubstanciadas nos algoritmos de optimização empregues para a prossecução do objectivo definido em 1. Muitos métodos de procura de soluções aproximadas foram sucessivamente utilizados,

o IBM System P5 575 (o melhor para bases de dados de 10000 GB) a mesma relação é de 47.00 US$. Informação obtida em 2006/10/03.

sendo os mais relevantes, constituindo-se assim como membros desta dimensão, os seguintes:

- algoritmos *greedy* (*Gr*);
- algoritmos evolucionários (mais especificamente genéticos (*Ge*) e genéticos co-evolucionários (*GeCoev*) e algumas hibridações(*GeH*));
- algoritmos de pesquisa aleatória e pseudo-aleatória, empregando *hill climbing* e *simulated annealing* (*Aleat*);
- algoritmos de optimização por enxame de partículas, na sua variante discreta (*ODiEP*) com variantes cooperativas (*ODiEP-Coop*) e multi-fase (*ODiEP-MF*);
- heurísticas específicas, típicas de controlo de caches (*HE*);
- uma combinação de algoritmos genéticos com heurísticas de pesquisa local, algo que se denominou de algoritmos miméticos (*Mi*) e outros, uma combinação de algoritmos genéticos, *hill climbers* e de enxame de partículas, algo que foi denominado de algoritmos *life cycle*, aqui uma implementação denominada algoritmo metamorfose (*Me*), onde o agente de busca se vai metamorfoseando em formas sucessivas, procurando explorar as virtudes de cada um dos algoritmos.

3. Tempo, periodicidade de recalibração, correspondendo ao intervalo temporal que medeia duas recalibrações das estruturas multidimensionais. Como se indicou atrás, teremos como membros desta dimensão: a) soluções estáticas (t>>0 - *E*), b) soluções dinâmicas (t≈0-*D*) e c) soluções pró-activas (t<0-*P*).
4. Constrangimentos, não mais do que limitações ou restrições severas, de ordem física, impostas ao processo de optimização. Estão relacionados com duas grandezas físicas quase omnipresentes: espaço (*E*) e tempo (*T*). O primeiro surge como a restrição espacial de armazenamento imposta à materialização dos cubos, que pode ser de natureza múltipla (no caso de OLAP distribuído). O segundo resulta da necessidade de actualização das agregações materializadas, consequência dos eventos ocorridos no mundo real (principalmente as transacções havidas nos sistemas operacionais). Supõe-se, neste caso, que o DWS terá uma utilização dual: um período em que estará disponível para responder a interrogações ou fonte de dados para outras aplicações e outro período devotado a actualizações (janela temporal de refrescamento ou de manutenção). O constrangimento de manutenção é, assim, não mais do que o período

de tempo disponível para o refrescamento do DW e demais agregações materializadas. Constitui-se também como uma possível restrição (ainda que algo elástica), mas que colide necessariamente com o período "útil" do DWS.

5. Espaço, correspondendo à distribuição das estruturas multidimensionais. Aplicação da velha máxima "*divide ut imperes*". Resume-se à clássica solução da engenharia: transformar um grande problema em vários pequenos problemas, mais manejáveis individualmente. No caso dos sistemas de processamento analítico, este surge em resposta ao crescimento do número de utilizadores e, especialmente, da dimensão física das próprias estruturas multidimensionais. Os membros desta dimensão são algo previsíveis, surgindo naturalmente as soluções centralizadas (*C*), distribuídas (D) e fortemente distribuídas (ou dispersas-*DD*). As implementações práticas relativas a cada membro desta dimensão podem assumir diversas formas arquitecturais. Estas serão descritas aquando da respectiva apresentação.
6. Natureza do modelo de custos, a simples caracterização do modelo utilizado para estimativa (cálculo) dos custos de interrogação e manutenção. O modelo de custos surge inevitavelmente baseado em grafos [Diestel, 2005] AND-OR ou OR [Gupta, H., 1997]. A maioria das propostas surgem ligadas ao modelo de custos dito linear (*L*), proposto inicialmente em [Harinarayan et al., 1996], que postula a dependência linear dos custos relativamente ao tamanho do subcubo utilizado. O modelo não linear (*NL*) admite a inclusão de não linearidades, permitindo assim um maior grau de abrangência em sistemas reais, já que, por exemplo, a existência de ordenações ou índices pode beneficiar a obtenção de um subcubo em detrimento de outro, mesmo utilizando o mesmo subcubo fonte. Qualquer cálculo de custo é baseado num modelo e, assim, este assume uma extrema relevância - um capítulo desta dissertação ser-lhe-á dedicado. De qualquer forma, apesar da sua relevância, a capacidade diferenciadora desta dimensão é baixa.

Quanto à representação gráfica multidimensional das soluções de optimização para estruturas multidimensionais de dados (Figura 2.13), apesar de algo limitada, já que está restrita a três dimensões, beneficia da clareza conferida pelo carácter visual. Além disso, como foram seleccionadas três dimensões – espaço, tempo e lógica de selecção - que introduzem, do nosso ponto de vista, um maior grau de diferenciação, continua a apresentar um elevado grau de informação. É com base nesta representação que será efectuada, já de seguida, a descrição das propostas mais relevantes, completando cada solução com os demais elementos caracterizadores

relativos às dimensões em falta. Refira-se que em anexo 2 é mostrada a mesma figura, mas algo mais completa, incluindo referências às propostas mais relevantes ou representativas de cada classe de soluções.

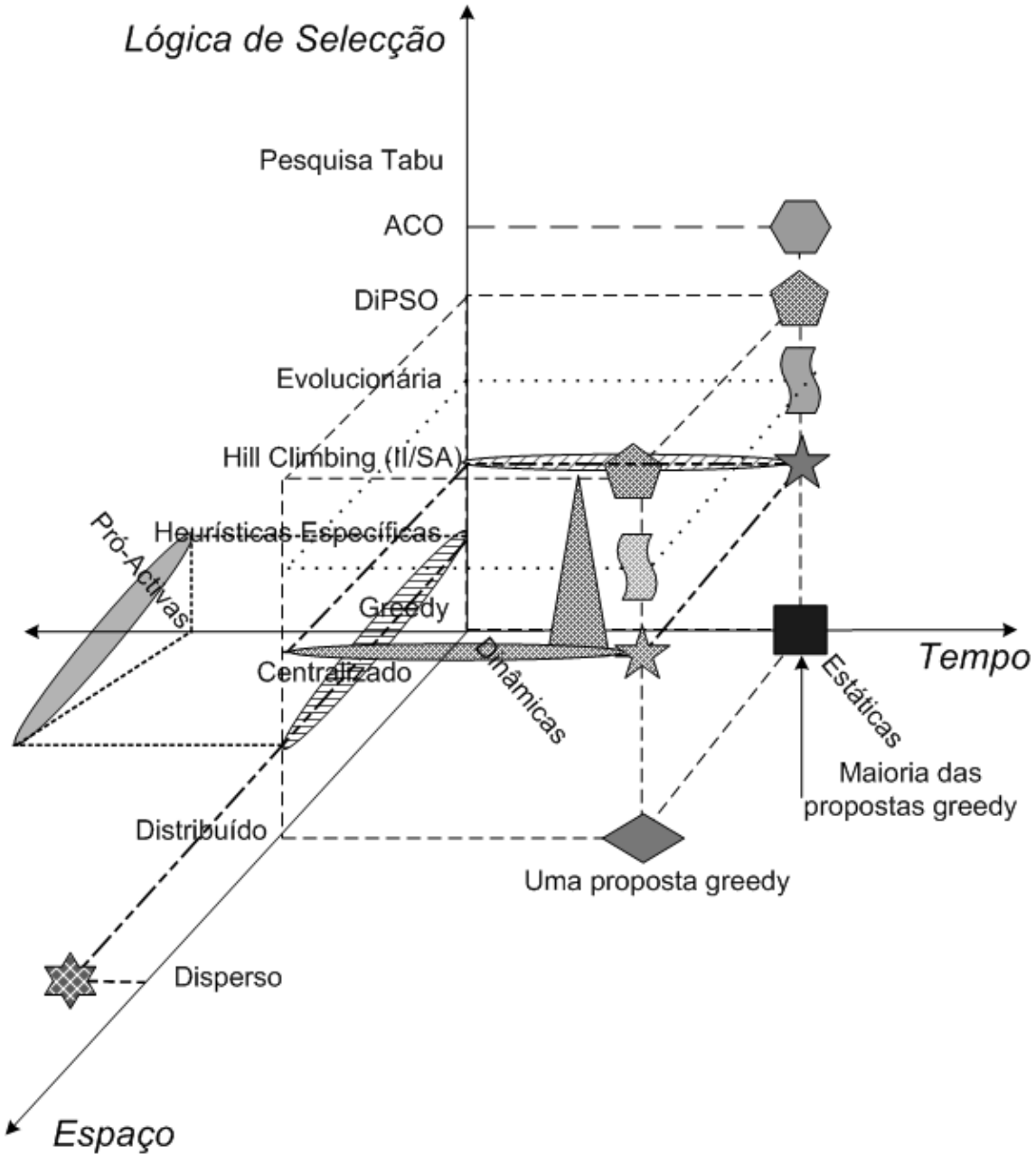


Figura 2.13. Caracterização tridimensional das soluções propostas para o problema de selecção de cubos.

Uma observação da Figura 2.13 revela que, quanto à primeira dimensão (espaço), há um sobrepovoamento de soluções endereçadas a estruturas centralizadas. Quanto à segunda dimensão (lógica de selecção), há a considerar uma maioria de propostas no domínio dos algoritmos *greedy*. A estas há a juntar soluções baseadas em heurísticas específicas, em algoritmos genéticos e selecção aleatória com pesquisa iterativa, por *simulated annealing* uma combinação de ambas. A optimização por enxame de partículas surge já no âmbito deste trabalho de doutoramento. Quanto à terceira dimensão (tempo), tem-se um claro predomínio das soluções estáticas, ainda que as soluções dinâmicas tenham também alguma relevância.

Consideremos, então, em primeiro lugar, a distribuição das estruturas. A maioria das soluções é focada na perspectiva centralizada, já que é a mais utilizada e também, até há muito pouco tempo, a única a ser implementada. Das muitas abordagens ao problema que poderiam ser referidas, seleccionaram-se algumas que passam a descrever-se, fundamentalmente utilizando duas lógicas de selecção: heurísticas *greedy* (representadas como um quadrado na Figura 2.13) e abordagens evolucionárias (representadas como um quadrilátero com as faces verticais curvas).

[Harinarayan et al., 1996] foi o primeiro trabalho a considerar o problema da selecção das vistas a materializar para suportar o processamento analítico de dados, tendo proposto um algoritmo *greedy* para solução do problema. Uma extensão a esta proposta é feita em [Gupta, H. et al., 1997] que inclui também a selecção de índices.

Apesar das características *greedy* dos algoritmos propostos, a sua escalabilidade não é fácil. Em [Baralis et al., 1997] é considerado o perfil de carga para a selecção das vistas. Dito de outra forma, se um conjunto de consultas relevantes especificadas pelo utilizador estiver disponível, então a exploração desta informação pode levar a uma significativa redução do espaço de soluções, dado que o número de consultas mais relevantes é extremamente pequeno quando comparado com o número de elementos do cubo de dados completo. Também procurando melhorar a escalabilidade de aplicação destes algoritmos, é proposta em [Shukla et al., 1998] uma modificação ao algoritmo *greedy* que selecciona as vistas de acordo com o seu tamanho. Esta mostra-se bastante mais eficiente, mantendo a mesma qualidade de outras propostas.

Por sua vez em [Soutyna & Fotouhi, 1997] é proposto um algoritmo de programação dinâmica para resolver o mesmo problema, que devolve o conjunto óptimo de vistas, dado um espaço disponível e frequência de consultas. Em [Shukla et al., 2000] é proposto um algoritmo multi-cubo, no qual o problema é considerado não baseado na suposição de que todas as agregações são calculadas a partir de um único cubo, mas para modelos de dados multi-cubo.

Em [Gupta, H., 1997] é proposta uma metodologia sistemática para a selecção das vistas a materializar, estendendo as consultas agregadas para uma forma mais geral, aquelas que possam ser representadas por um grafo AND-OR. Em [Gupta & Mumick, 1999] é considerado o constrangimento não só do espaço ocupado pelas vistas a materializar, mas também o seu custo de manutenção. Este mesmo problema é discutido em [Liang et al., 2001].

A forma de abordar o problema utilizando um grafo AND-OR é adequada para lidar com o problema da selecção de vistas de um modo geral, mas não é adequada para OLAP. A questão reside no facto deste grafo ser utilizado na avaliação de todas as possíveis alternativas na forma de executar uma interrogação, usando um conjunto de relações base, na presença de outras interrogações e vistas. Contudo, em OLAP, a natureza das agregações e de agrupamentos nas interrogações é tal que a dependência das vistas, considerando-as numa perspectiva do *lattice* (ou grafo de vistas OR), é a mais adequada, tal como foi proposto inicialmente em [Harinarayan et al., 1996], [Gupta, H., 1997] e [Gupta, H. et al., 1997].

Outras propostas seguiram uma abordagem indirecta. Em [Theodoratos & Selis, 1997] é proposto que todas as interrogações devam ser respondidas só pelas vistas materializadas, reescrevendo ou não as interrogações colocadas pelos utilizadores. Não é considerado o problema de constrangimento espacial, sendo propostos algoritmos que fazem a procura de soluções de forma exaustiva e heurística, modelando o problema como uma optimização no espaço de estados. Um outro modelo, chamado *Plano de Processamento por Vistas Múltiplas* (PPVM), é proposto em [Yang et al., 1997], explorando a existência de sub-expressões comuns na maioria das interrogações.

Uma abordagem mais geral, que procurou incorporar todas as questões tratadas até então foi proposta por [Theodoratos & Bouzeghoub, 2000], baseando-se mais uma vez em grafos AND-OR. Este autores procuraram acomodar todos os objectivos possíveis de concepção, tais como custo de processamento para resposta às interrogações e manutenção das vistas e constrangimentos como custos de espaço e manutenção. Já em [Jamil & Modica, 2001], é apresentada a concepção e implementação de um sistema de selecção de vistas para bases de dados multidimensionais.

Abordagens num outro domínio foram também intentadas e propostas, como, por exemplo, a pesquisa de soluções utilizando técnicas evolucionárias ou aleatórias. Assim, há algum trabalho dedicado à aplicação de algoritmos genéticos [Horng et al., 1999], [Zhang et al., 1999], [Zhang et al., 2001], [Lin & Kuo, 2004] ou pesquisa aleatória em [Kalnis et al., 2002b]. Em [Horng et al., 1999] é proposto um algoritmo genético para seleccionar o conjunto de vistas a materializar de forma a minimizar os custos de consultas, atendendo a constrangimentos de espaço e tempo de

manutenção e à frequência de utilização e de manutenção. A proposta de [Lin & Kuo, 2004] tem um objectivo semelhante, diferindo no esquema incluído para lidar com as soluções inviáveis, pois que violam o constrangimento. Se em [Horng et al., 1999] é utilizada uma função de penalidade que castiga qualquer solução impraticável, já em [Lin & Kuo, 2004] é proposto um esquema de reparação, sob a forma de um algoritmo de reparação *greedy*. As duas outras propostas, no domínio dos algoritmos genéticos, são, na realidade, abordagens híbridas, já que utilizam uma combinação de heurísticas e de algoritmos genéticos, conseguindo-se assim uma grande diminuição do grau de complexidade e consequente diminuição do tempo de execução.

A última proposta, já no domínio da pesquisa aleatória, representada como uma estrela de 5 pontas com sombreado compacto (Figura 2.13), utiliza algoritmos progressivamente mais eficientes (desde a pesquisa iterativa, por *simulated annealing* e uma combinação de ambos), especialmente interessantes para DW com elevada dimensionalidade, conseguindo soluções de qualidade próxima dos algoritmos *greedy*, num tempo três ordens de grandeza inferior [Kalnis et al., 2002b].

A solução distribuída só muito recentemente é que veio a terreiro [Bauer & Lehner, 2003] (Figura 2.13 como um losango), na qual o cubo irá ser distribuído por vários nós (podendo conter algum grau de replicação). É introduzido um *lattice* de agregações distribuído e um modelo linear estendido, considerando-se custos de interrogação para cálculo do benefício. Esta solução não inclui explicitamente custos de comunicação e processamento, apenas um factor adicional correspondente à razão entre os dois. É proposto um algoritmo *greedy*, denominado "*distributed node set greedy*" [Bauer & Lehner, 2003], que pode ser visto como uma extensão natural ao algoritmo greedy para um só nó. A avaliação experimental mostra uma vantagem do algoritmo proposto face ao algoritmo *greedy* para um só nó.

Todas as propostas acabadas de descrever são caracteristicamente estáticas, com excepção parcial da apresentada em [Kalnis et al., 2002b], que pode ser utilizada também numa abordagem dinâmica. Introduzindo a segunda dimensão da Figura 2.13, vão abordar-se algumas das propostas dinâmicas e pró-activas, uma vez que as estáticas acabaram de ser já descritas. A primeira proposta no domínio das propostas dinâmicas (mostradas como uma elipse sombreada a tracejado na Figura 2.13) talvez tenha surgido em [Scheuermann et al., 1996], sob a forma de um gestor de caches denominado WATCHMAN (*WArehouse inTelligent caCHe MANager*) que armazena em cache o resultado das consultas em conjunto com a expressão da consulta. As consultas seguintes podem ser satisfeitas pela cache se houver uma correspondência perfeita entre as expressões das

consulta, podendo utilizar-se um conjunto de algoritmos [Chen & Roussopoulos, 1994], [Gupta, A. et al., 1995] para testar a equivalência entre consultas, e, assim, alargar o número de consultas eventualmente satisfeitas pelo conteúdo da cache. Em [Scheuermann et al., 1996] são propostos dois algoritmos: um para a substituição de cache (LNR-C – *Least Normalized Cost Replacement*) e outro para a admissão em cache (LNC-A – *Least Normalized Cost Admission*). Ambos utilizam uma denominada "métrica de lucro", que considera para cada resultado de uma consulta não só a sua taxa média de referência, mas também o custo do recálculo de um resultado em conjunto com o tamanho do resultado. Tal como o sistema é descrito, depreende-se que a localização e instalação podem considerar-se bastante versáteis, visto que o gestor de cache é implementado como uma biblioteca de rotinas que podem ser ligadas com um qualquer gestor de *warehouse*. Assim, pode residir na mesma plataforma do DW, podendo a cache ser armazenada em memória ou em disco.

Em [Karayannidis & Sellis, 2001] é proposto um gestor de armazenamento específico para cubos de dados, nomeado SISYPHUS. A solução proposta é baseada na divisão regular do espaço multidimensional em *chunks* [Desphand et al., 1998], [Zhao et al., 1997]. Tem a vantagem de poder ser implementado sobre um gestor de armazenamento orientado a registos, proporcionando um conjunto de níveis de abstracção típicos em sistemas de gestão de armazenamento, mas modificados, adaptando-os às características do espaço de dados em OLAP (multidimensionalidade e hierarquias). O sistema de ficheiros SISYPHUS é nativamente multidimensional e suporta hierarquias, agrupando os dados de acordo com elas, sendo eficiente na utilização do espaço. Quando um resultado novo é recebido, este é colocado na cache nos *chunks* correspondentes. Ao ser necessário responder a uma nova interrogação, o sistema calcula o conjunto de *chunks* necessários à resposta, dividindo-os em dois grupos: os que estão presentes na cache e os que não estão. A resposta é depois construída pela busca dos primeiros na cache, sendo os restantes pedidos ao sistema subjacente (um qualquer repositório OLAP). Não é permitida agregação de resultados armazenados na cache. Isto implica que só são utilizados *chunks* do mesmo nível de agregação da consulta. Quanto a algoritmos de admissão e substituição, aqui são utilizados os mesmos que em WATCHMAN. O sistema SISIPHUS é mais propriamente um gestor de armazenamento, mas que pode ser implementado como uma cache, se for esse o objectivo.

Em [Kotidis & Roussopoulos, 1999] é apresentado outro gestor de caches OLAP: Dynamat. Nesta proposta são armazenados os denominados "fragmentos de cubo" que são resultados de consultas agregadas, com uma granularidade mais fina do que as vistas (podem incluir selecção de valores simples em algumas das dimensões). O sistema proposto tem duas fases de funcionamento: uma, denominada de "*on-line*", e outra, nomeada "janela de actualização".

Durante a fase *on-line*, quando uma interrogação chega, o sistema verifica se existe um fragmento de cubo relacionado na cache (não necessariamente ao mesmo nível de agregação da consulta), cujo repositório se denomina *pool* de vistas. Para isso, utiliza um localizador de fragmentos que os procura, pesquisando um outro componente da arquitectura, denominado de índice de directório. Se existir, o fragmento é utilizado para responder à consulta; caso contrário, as tabelas base são acedidas, sendo o resultado armazenado como um novo fragmento de cubo na cache. Claro que, como a cache não é de dimensão infinita, esta última operação pode implicar a violação do constrangimento espacial, sendo necessário seleccionar um fragmento a eliminar, utilizando uma métrica de benefício descrita um pouco mais abaixo.

Na fase de actualização, surge em cena o constrangimento temporal: dada uma janela de actualização de dimensão temporal fixa, o Dynamat deve excluir um conjunto de fragmentos que permitam que a actualização seja efectuada no tempo disponível. Uma métrica de avaliação de benefício é utilizada para seleccionar o fragmento da cache a eliminar, utilizável nas duas fases. A métrica é baseada em vários critérios: o tempo do último acesso (que pode ser considerado como uma política de substituição tipo *Least Recently Used* - LRU); a frequência de acessos ao fragmento (uma política de substituição tipo *Least Frequently Used* - LFU); o tamanho do fragmento, considerando que fragmentos grandes serão provavelmente mais úteis em consultas e que conduzem a um número menor de fragmentos na cache e consequentemente a um menor tempo de procura no directório (esta pode ser considerada como uma política tipo *Smaller Fragment First* – SFF). A combinação destes critérios conduz à chamada métrica *Smaller Penalty First* (SPF), que é directamente proporcional ao número de acessos e ao custo de recálculo do fragmento e inversamente proporcional à idade do fragmento, normalizado tendo em conta o tamanho do fragmento. O Dynamat é também um gestor dinâmico de armazenamento de vistas, que é proposto como uma cache OLAP, já que surge como um componente interposto entre o DW e o utilizador. De qualquer modo, não deverá colocar-se de lado a possibilidade de poder ser estendido a outros níveis da arquitectura dum DWS.

Em [Shim et al., 1999] é proposto mais um gestor de caches dinâmicas para sistemas de suporte à decisão. Esta solução inclui políticas de substituição de caches e gestão dinâmica do seu conteúdo e permite a resposta a consultas baseada não só em correspondência exacta a resultados armazenados (como no WATCHMAN e em parte no Dynamat), mas também utilizando fragmentos com correspondência não exacta, para consultas ditas canónicas [Shim et al., 1999]. O mecanismo que o permite é a utilização de um algoritmo de divisão de consultas, que vai transformar a consulta colocada, de forma a poder encontrar eventuais resultados armazenados de consultas que

possam satisfazê-la, utilizando um grafo dinâmico denominado "grafo de ligação de consultas", que representam os relacionamentos entre as consultas. A informação contida neste grafo é também utilizada no mecanismo de cálculo de lucro utilizado no algoritmo de admissão e substituição de resultados de consultas da cache LNC-RA, um derivado do LNC-R [Scheuermann et al., 1996], que atende à diminuição dos custos devidos à sua materialização, à taxa média de referência, à taxa média de actualizações, ao seu custo de actualização e ao seu tamanho. Aqui não é especificado explicitamente um qualquer enquadramento num DWS, aparecendo, no entanto, no diagrama arquitectural do gestor de cache, um componente de armazenamento denominado de "dados base", ao qual aparece ligado o gestor de cache. Assim, dada esta panorâmica, pode depreender-se a sua utilização como um novo componente da arquitectura, funcionando entre o DW e o utilizador, podendo assim residir num servidor dedicado, num DM ou ainda no próprio cliente.

Em [Kalnis & Papadias, 2001] é apresentada e avaliada experimentalmente uma arquitectura denominada de Servidores-Cache OLAP (SCO), que já fornece soluções de enquadramento a nível global, surgindo como uma extrapolação para os sistemas OLAP dos servidores *proxy*, que são largamente utilizados no ambiente WWW. Um SCO, contudo, não armazena páginas estáticas, mas armazena dinamicamente os resultados de consultas OLAP, tendo também capacidades computacionais, sendo, assim capaz de gerar novas agregações. Na arquitectura proposta, os SCO estão geograficamente dispersos e ligados através de uma rede arbitrária. Os clientes de dados OLAP não acedem ao DW directamente, mas sim através de um SCO, que, por seu turno, ou responde à consulta da sua cache local, ou a redirecciona para um dos seus vizinhos. Esta arquitectura proporciona uma maior disponibilidade dos dados OLAP a clientes remotos, um abaixamento de tráfego da rede e uma melhor escalabilidade, já que a adição de novos SCOs aumenta a capacidade de armazenamento e de processamento. Do ponto de vista do utilizador, a melhoria traduz-se num menor tempo de resposta. Esta proposta traz para o domínio OLAP um assunto relevante: o denominado *caching* activo em *proxies* Web [Cao et al. 1998]. Isto permite ao servidor *proxy* construir páginas dinamicamente, armazenando os dados em conjunto com uma *applete* para os manipular.

Se as propostas dinâmicas, atrás descritas, supunham uma arquitectura em três camadas, onde a cache surge na camada intermédia, a arquitectura SCO é mais geral, permitindo vários níveis de *cache* com a cooperação entre os seus diversos servidores, entrando também em consideração com o factor rede. A arquitectura do sistema segue o esquema típico de uma arquitectura

servidor-proxy, largamente utilizada para fazer o *caching* de páginas Web, na qual existem três camadas:

- Data Warehouse (DW), que permite eventualmente vários DWs distintos;
- Servidores-Cache OLAP (SCO), ligados aos DWs, e interligados através de uma rede arbitrária;
- cliente, onde os utilizadores acedem aos DWs de forma indirecta, colocando as interrogações ao SCO.

Cada SCO contém uma cache que armazena resultados de interrogações, cujas capacidades computacionais lhe permitem agregar os resultados armazenados localmente para responder a uma nova interrogação. A intuição base desta arquitectura é que algum SCO na rede será capaz de satisfazer um qualquer pedido de um cliente, poupando assim recursos de rede e de computação que seriam incorridos se a resposta fosse proporcionada pelo DW.

O cerne desta arquitectura é a distribuição e a consequente cooperação; há uma distribuição de dados e de processamento, que permitem a resposta a interrogações. Há que decidir como saber quem vai responder a interrogações e quem decide onde armazenar (em que cache SCO) um novo resultado de uma interrogação obtido de um DW. Em resumo, e dada a versatilidade da arquitectura, esta pode materializar-se de diversas formas. As mais relevantes são porventura três:

- Arquitectura centralizada, em que um sítio central tem informação sobre o estado de todos os SCOs. Este sítio será responsável pela criação de todos os planos de execução de consultas e decidir se e onde quaisquer resultados de interrogações serão armazenados.
- Uma arquitectura semi-centralizada, onde continua a haver um sítio central responsável pelo planeamento de execução das interrogações, mas cada SCO controla localmente a sua cache.
- Uma arquitectura autónoma, onde não teremos qualquer sítio central. Todas as decisões são locais.

Tendo sido avaliados comparativamente os três tipos de arquitecturas, e de forma intuitiva, poder-se-á dizer que, porventura, o primeiro permitirá custos de execução mais baixos e melhor utilização dos SCOs, devido a um conhecimento do estado global do sistema. Essa é a abordagem utilizada na maioria dos SGBDs distribuídos e será, em OLAP, a adequada para instalações

Intranet, onde a infra-estrutura pertence à mesma organização, especialmente se a distribuição típica (DMs) não é viável, dada a natureza dinâmica e altamente imprevisível dos padrões de consultas. No outro extremo da arquitectura, a autonomia total será adequada para grandes ambientes não estruturados (por exemplo Internet), proporcionando maior escalabilidade e disponibilidade do sistema. A solução intermédia permite naturalmente estabelecer um compromisso entre as duas soluções extremas.

Ainda no domínio das soluções dinâmicas em ambiente fortemente distribuído, importa referir a proposta de [Kalnis et al., 2002a] (Figura 2.13), a estrela de seis pontas, sombreada. Aqui é proposta uma arquitectura PeerOLAP, constituída por um número possivelmente grande de clientes simples (vulgares PCs), cada um contendo uma cache, com os resultados mais úteis, clientes estes conectados por uma rede de comunicações, eventualmente heterogénea. Se uma interrogação não puder ser respondida localmente (i.e. utilizando o conteúdo da cache do computador onde a interrogação foi colocada), ela é propagada pela rede até que um *peer* tenha na sua cache a resposta pretendida. Uma resposta pode igualmente ser gerada utilizando resultados parciais (fragmentos) existentes em várias caches. Desta forma, PeerOLAP actua como uma grande cache distribuída que estende os benefícios da utilização das caches tradicionais no lado do cliente. Este sistema é completamente distribuído e pode auto-reconfigurar-se de forma a acompanhar as variações do perfil de carga, diminuindo assim os custos de interrogação.

Para terminar a descrição de algumas propostas relativas à dimensão temporal de recalibração, estão em falta as pró-activas (ver a Figura 2.13 a elipse com sombreado contínuo). Caches com *prefetching* ou restruturação (com recálculo dinâmico dos subcubos futuramente necessários) são formas possíveis de soluções desta classe de propostas [Belo, O., 2000], [Sapia, 2000]. Também a inclusão de avaliação de utilidade futura de subcubos ou fragmentos na política de admissão e substituição de cache constitui uma proposta englobada nesta categoria de soluções [Park et al., 2003].

A primeira proposta [Belo, O., 2000] surge como um assistente pessoal – um agente de software típico – dotado de capacidades de actualização dinâmica das "*CubeViews*" mais requeridas por um ou um grupo que partilhe um conjunto de fragmentos de cubos de dados num ambiente distribuído. A arquitectura proposta e mecanismos de aprendizagem, na qual sobressai a unidade de aprendizagem e o motor de inferência, são responsáveis pelo processo de aquisição de conhecimento e pela selecção de regras sobre as necessidades dos utilizadores e decisão da melhor forma de reestruturar os cubos, permitindo a redefinição das vistas mais acedidas num

tempo razoável. Desta forma, operações de *pivoting*, *slice and dice* e consultas a novas "*CubeViews*" podem ser propostas ao utilizador tendo sido pré-preparadas *on-the-fly* as estruturas de disponibilização das consultas adequadas. Tudo isto será feito através da análise de uma base de conhecimento relativa à história de interacção com o utilizador, agrupada por eventos, analisada temporalmente, constituindo um cubo do perfil do utilizador.

Nesta proposta, parece estar implícita a existência de duas categorias principais de acções: 1) o recalculo *on-the-fly* de valores agregados, correspondentes a possíveis operações de *pivoting* ou *slice and dice*, na qual poderemos considerar que haverá a predição da ou das próxima(s) operação(ões) do utilizador; para tal, vão calcular-se e propor-se as "*cubeviews*", estando-se em presença de uma proactividade de profundidade curta; 2) a restruturação de "*CubeViews*" para novas operações de consulta, onde a profundidade poderá considerar-se média ou longa.

Em resumo, pode entender-se o assistente pessoal proposto como de função dual: efectua o recálculo e a restruturação de "*CubeViews*" e propõe as próximas acções ao utilizador, minimizando a latência percebida pelo utilizador. Uma e outra função têm como base o conhecimento do perfil de necessidades do utilizador. Esta última função é, aliás, alvo de muita investigação em vários trabalhos em sistemas de recomendação, de que [Mitchel et al., 1994] é um exemplo.

A segunda proposta [Sapia, 2000] denominada PROMISE (*Predicting User Behavior in Multidimensional Information Systems Environments*), utiliza um mecanismo preditivo baseado num modelo Markoviano de 1.ª ordem. Estabelece a exequibilidade da solução em duas vertentes:

- A primeira, ao analisar a carga de um sistema OLAP real, mostra que um pedido tem uma grande probabilidade de estar "perto" do pedido anterior.
- A segunda mostra a existência de um período longo entre dois passos navegacionais, dito de "tempo de consideração", aquele que se segue a uma resposta de uma interrogação e termina com a colocação da próxima; é o tempo que o utilizador leva a analisar a resposta à sua interrogação anterior, até formular nova hipótese e colocar a nova interrogação. Ora, este tempo permite que se possa efectuar um *prefetching* da próxima interrogação.

Desta forma, e criando modelos Markov como um formalismo para modelar o processo de análise OLAP (o comportamento interactivo), mas utilizando abstracção de padrões no que é chamado de protótipo de estrutura de interrogação, é possível propor um algoritmo de predição, o qual vai ser

utilizado para efectuar o *prefetching*. Efectivamente, são criados dois modelos de Markov independentes por cada dimensão: um, o denominado modelo estrutural, relativo à estrutura das consultas (podendo ser utilizado para predizer os níveis de selecção, a granularidade resultado e as medidas resultado da próxima consulta); outro, chamado de modelo baseado em valor, que contém informação acerca de caminhos de navegação típicos na dimensão (pode ser utilizado para predição de valores de restrição da consulta). Em termos práticos, a proposta parece não contemplar propriamente uma cache, mas apenas o mecanismo de *prefetch* de profundidade curta. De qualquer forma, trata-se de uma primeira abordagem explícita de mecanismos de *prefetch* em OLAP.

Quanto à terceira proposta [Park et al., 2003] apresenta um gestor de caches baseado na utilidade dos resultados de interrogações para a reescrita e processamento de outras interrogações. A política de admissão e substituição dos resultados OLAP na cache considera o benefício não só em termos das interrogações lançadas recentemente, mas também atendendo às interrogações esperadas no futuro. Também aqui é utilizado um mecanismo de predição tal como em [Sapia, 2000], mas não utilizado para *prefetching*, mas sim para a inclusão de interrogações futuras na política de gestão da cache, denominada *lowest-usability-first-future*. Este mecanismo de predição é baseado na exploração dos relacionamentos semânticos entre as sucessivas interrogações de uma sessão OLAP, derivados da natureza interactiva e navegacional das interrogações OLAP, utilizando um modelo probabilístico.

Já no que respeita à terceira dimensão, a primeira família, que poderemos denominar de "heurísticas *greedy*", é aquela que mereceu um maior número de propostas de solução. Logo em [Harinarayan et al., 1996], ela foi iniciada, sob a forma de um algoritmo GSC (*Greedy under Space Constraint*), que introduz a heurística base de toda a família: o conceito de benefício e o início com um conjunto de subcubos vazio, ao qual se vai adicionar aquele que, em cada fase, se revelar como o mais benéfico em termos da diminuição do custo de resposta ao conjunto de interrogações colocadas. Muitas outras propostas poderiam ser referidas, que foram acrescentando várias heurísticas à heurística base descrita, alargando a sua aplicação e introduzindo novos constrangimentos (como é o caso do tempo de manutenção [Gupta & Mumick, 1999]). Na actualidade, há nesta família quatro propostas dominantes: os algoritmos *greedy* de árvore invertida e A* [Gupta & Mumick, 1999], e os algoritmos *greedy* de duas fases e *greedy* integrado [Liang et al., 2001] - uma análise comparativa utilizando um modelo de custos não linear e descrição pormenorizada pode ser encontrada em [Yu et al., 2004].

A segunda família traz para o domínio da selecção das vistas ou subcubos os algoritmos genéticos [Holland, 1992]. Na sua essência, um algoritmo genético procura capitalizar directamente a forma como se processa a evolução, sabendo-se que esta é um método robusto e com sucesso para adaptação em sistemas biológicos. Estes algoritmos podem pesquisar espaços de hipóteses, contendo partes inter-actuantes complexas, onde o impacto de cada parte nas hipóteses genéricas de elegância possa ser difícil de modelar. Eles começam com uma população aleatória (grupos de cromossomas); em cada geração são seleccionados pais e gera-se uma descendência, utilizando operações análogas aos processos biológicos, tipicamente, as operações de cruzamento e mutação. Adoptando o princípio da sobrevivência do mais apto, todos os cromossomas são avaliados utilizando uma função de adaptação para determinar a qualidade de cada solução, que é então utilizada para decidir que indivíduos (e respectivos cromossomas) vão ser eliminados ou propagados. Quanto mais elevada for a adaptação, maior é a probabilidade do cromossoma respectivo ser utilizado na geração da nova população. O processo é repetido sucessivamente até que um determinado critério de finalização seja satisfeito. O cromossoma com o maior valor de adaptação, na última população, dá a solução. Uma descrição geral dum algoritmo genético simples pode ser consultada em [Goldberg, 1989]. Ao fim de um número reduzido de gerações, conseguem chegar a soluções, senão melhores do que as dos algoritmos *greedy*, pelo menos algo comparáveis [Horng et al., 1999], [Zhang et al., 2001], [Lin & Kuo, 2004].

Já as propostas no domínio da pesquisa aleatória procuram dar resposta ao tempo de execução muito longo dos algoritmos *greedy*, sem aplicação em DW de elevada dimensionalidade. Em [Kalnis et al., 2002b] é proposta uma heurística sob a forma de um reservatório (com um tamanho correspondente ao constrangimento espacial), à qual são adicionadas vistas (previamente ordenadas), até que o constrangimento temporal seja violado. A selecção e eliminação das vistas pode ser efectuada através de três tipos de pesquisas: melhoramento iterativo (II), que mais não é do que um *hill climber*, *simulated annealing* (SA) e optimização em duas fases (2PO), uma combinação das duas anteriores, dando origem a três algoritmos. Estes são aplicados ao problema da selecção de vistas sob constrangimento espacial num DW centralizado, mostrando o último conseguir soluções de qualidade comparável ao algoritmo GSC em tempos três ordens de grandeza inferiores (para um DW com 15 dimensões).

Uma outra proposta [Maniezzo et al., 2001], utiliza um método de optimização inspirado na vida, neste caso, o comportamento de uma colónia de formigas, apropriadamente designado como *Ant Colony Optimization* (ACO) [Dorigo et al., 1996]. As formigas depositam uma substância química (chamada feromona) no solo que percorrem. Esta substância influencia a escolha que elas fazem

ao seleccionarem o caminho a seguir: um determinado caminho é seleccionado com maior probabilidade por uma formiga se ele tiver maior quantidade de feromona. Este comportamento implementa um mecanismo autocatalítico (realimentação positiva). A deposição de feromona, que cria um caminho, pode ser visto como uma forma simples de comunicação indirecta, conhecida como *stigmergy*. A proposta aborda o problema mais restrito da selecção de fragmentos de vistas a materializar num DW, também designado como problema de fragmentação vertical, descrito em [Munneke et al., 1999]. Utiliza ANTS [Maniezzo, 1999], uma técnica da classe de optimização por colónia de formigas. A ideia principal, subjacente a todos os algoritmos ACO, é a pesquisa paralela efectuada por várias *threads* computacionais, todas baseadas numa estrutura dinâmica de memória que incorpora informação acerca da eficácia dos resultados obtidos anteriormente. Uma formiga é definida como um agente computacional simples que iterativamente constrói uma solução para o problema a resolver. O comportamento colectivo emerge da interacção das diversas *threads* de procura, tendo provado ser eficaz na solução de problemas de optimização de carácter combinatório.

Também no domínio dos algoritmos inspirados na vida, há a referir a optimização por enxame de partículas, dado que utilizada no âmbito desta dissertação, sendo portanto a base de alguns dos esquemas de optimização propostos. O conceito do enxame de partículas teve origem numa simulação de um sistema social simplificado, mais especificamente um bando de aves ou cardume de peixes. A sua aplicação em optimização é proposta em [Kennedy & Eberhart, 1995] e [Eberhart & Kennedy 1995]. O algoritmo de optimização por enxame de partículas (PSO) simula a capacidade de processar conhecimento das sociedades animais mais evoluídas. Aqui, tem-se um conjunto de agentes simples, as partículas, que voam num espaço multidimensional, mapeado no espaço de soluções do problema a resolver. Desta forma, cada posição da partícula corresponde a uma solução para o problema. Como qualquer sistema dinâmico, a partícula tem, em cada instante, uma posição e uma velocidade. A velocidade é depois modificada por actuação de uma aceleração, onde intervêm dois tipos de informação: o conhecimento relativo ao seu próprio sucesso (a melhor posição atingida) e o conhecimento do sucesso colectivo (a melhor posição conseguida por uma qualquer partícula do enxame). Esta alteração da velocidade e, consequentemente, da localização da partícula, procura levá-la a localizações correspondentes a melhores soluções para o problema. Há duas versões principais do algoritmo PSO: a versão inicial, que usa valores reais, podendo assim denominar-se contínua (onde o espaço é contínuo) e a versão binária ou discreta (onde o espaço é discreto – a posição da partícula em cada dimensão é

0 ou 1), proposta em [Kennedy & Eberhart, 1997], vocacionada para a solução de problemas de optimização de carácter combinatório.

2.9 Como Assegurar a Escalabilidade do Sistema

O sucesso do processamento analítico levou a uma procura sucessivamente crescente dos seus serviços: o número de utilizadores cresce e cada vez mais áreas de negócio são englobadas. Também o volume dos próprios dados cresce (em resultado da dimensão temporal e do crescimento das próprias organizações). Torna-se necessário haver hardware sucessivamente mais poderoso e de custo exponencial; a própria administração da informação torna-se uma tarefa quase impossível. Portanto, é necessário um sistema de processamento analítico mais escalável, nas suas várias vertentes. Várias soluções arquitecturais e conceptuais foram sendo propostas, para as quais as soluções de optimização acabadas de descrever foram gradualmente endereçadas.

As respostas aos problemas colocados pela procura crescente aos dados de interesse analítico podem ser englobadas em duas grandes categorias de soluções. A primeira, de actuação local, almeja a melhor adequação de cada nível da infra-estrutura informacional: acções que podem ser denominadas de "intra-nível", já que procuram a optimização de recursos dentro de cada nível. É o caso, por exemplo, das soluções para o problema da selecção de vistas a materializar no DW, ou a distribuição dessas vistas por vários nós computacionais. Já a segunda actua a nível global, procurando atrair novos conceitos consubstanciados em novos componentes arquitecturais, providenciando à sua inclusão harmoniosa no todo da arquitectura informacional, através da necessária coordenação entre os vários níveis da arquitectura, podendo assim denominar-se de actuações "inter-nível" e globais. Na prática, assiste-se a uma extensão do próprio conceito de *Data Warehousing*, ao postular a separação dos sistemas operacionais dos de suporte à decisão, aqui separando funções pela inclusão de novos níveis na arquitectura do DWS. Alguns exemplos deste tipo de acções serão a inclusão de *Data Marts* (DMs), as caches de armazenamento de resultados de interrogações, procurando a sua reutilização futura, a hierarquização destas, e soluções de coordenação e visão global, forçando à inclusão de *middleware* apropriado.

Até aqui, mostrou-se apenas como as soluções surgiram em resultado da pressão da procura, mas, na realidade, um outro actuador foi também determinante: a história. Na verdade, se recuarmos um pouco, a própria génese do conceito de *Data Warehousing* resultou da confluência da evolução

histórica dos sistemas de informação e da sua aparente impossibilidade de adaptação, em face das necessidades de informação das organizações. A pressão no sentido da eficiência dos sistemas operacionais e a sua continuada urgência levaram à criação de uma entidade altamente especializada e autista, em grande medida incapaz de mudar em consequência das alterações do meio (as emergentes necessidades informacionais). Na verdade, muitas tentativas de extracção de dados dos sistemas operacionais, para tratamento externo, esbarravam com o seu isolamento relativamente aos seus pares (com a consequente panóplia de incongruências e inconsistências) e da sua quase impossível alteração (ambas consequência da história do seu desenvolvimento). Também a perturbação causada pela nova função, que tentava ser-lhe acometida, era lesiva da sua tarefa primordial: acompanhamento e suporte dos processos de negócio. Esta confluência de factores concernentes à história da evolução dos sistemas operacionais ditou a emergência do *Data Warehousing*: a procura de uma resposta rápida às necessidades de informação do negócio, com um mínimo de interferência no suporte aos próprios processos operacionais.

A maior ou menor premência na obtenção de informação, muitas vezes consequência do atraso da introdução do *Data Warehousing* (e sua oportunidade), ditou também o seu processo de introdução, a sua arquitectura de suporte e respectiva evolução. Também é preciso não esquecer os factores atrás referidos, resultantes da procura crescente de informação analítica. Assim, a arquitectura de um DW pode assumir várias configurações possíveis, não só por imperativos de desempenho e administração, mas também por questões ligadas à própria história do seu desenvolvimento e operação nas organizações. Assim, se num grande DW se torna difícil (dir-se-ia impossível) a adaptação às necessidades individualizadas de cada utilizador, sendo igualmente de grande complexidade a sua administração, o que vai obrigar à sua distribuição, também o inverso pode suceder: tendo-se um DM, ao qual vão sendo acrescentados outros, em algum momento assiste-se à necessidade de criar uma perspectiva global. Estas situações mostram as várias classes de arquitecturas de DWS que podem ser consideradas: os denominados DW empresariais (a perspectiva centralizada), os DMs, os DW empresariais distribuídos e a perspectiva híbrida, que pode incluir uma amálgama de cada uma destas. De seguida, vamos apresentar uma panorâmica de cada uma destas, mostrando em que medida procuraram responder aos problemas colocados.

A adopção atempada do conceito nascente permitiu um desenvolvimento harmonioso e concepção do DW: é a abordagem tradicional e aquela em que a maioria das pessoas pensa quando se fala em DW. Nesta, os dados da empresa estão armazenados numa única base de dados, com um único modelo de dados. Dada a perspectiva centralizada, permitem correlacionar informação entre quaisquer sectores ou actividades de uma empresa, proporcionando aos decisores dados

consistentes e integrados de todas as áreas mais relevantes do negócio. Parece ser uma abordagem do maior interesse do ponto de vista da disponibilização dos dados, mas que pode, também, gerar problemas quanto à sua escalabilidade: quando o número de fontes de dados é grande e diversificado e o número de utilizadores é igualmente grande, esta torna-se de difícil implementação, podendo causar o falhanço de todo o projecto. Além disto, mesmo se a implementação e instalação é bem sucedida, dado o DW ser uma estrutura "viva", o seu crescimento é inevitável. Muitas das soluções de optimização (tendentes à selecção das agregações a materializar) que, na secção anterior, foram caracterizadas como estáticas e centralizadas (por exemplo [Gupta & Mumick, 1999], [Liang et al., 2001], para esquemas de baixa dimensionalidade, [Lin & Kuo, 2004], [Zhang et al., 2001] para esquemas de média dimensionalidade e [Kalnis et al., 2002b] para esquemas de elevada dimensionalidade) endereçam este tipo de DW. Surgem depois problemas relacionados com a sua manutenção, optimização e, acima de tudo, questões relativas à limitação da janela de refrescamento: o tempo necessário à extracção, preparação e integração dos novos dados no DW. Neste caso, a preparação das estruturas de optimização de acesso necessárias torna-se progressivamente maior, à medida que o DW vai crescendo, o que, no limite, torna tendencialmente nulo o tempo em que o DW fica disponível para consultas.

As limitações acabadas de evidenciar depressa conduziram a novas soluções. Percebeu-se que a tradicional solução do "*divide ut imperes*" teria de ser adoptada: em grandes DWs o número de sumarizações tornou-se demasiado elevado. Isto tornou o DW difícil de manejar e administrar, além de implicar um tempo de refrescamento muito grande. Mas o facto, observável, de cada departamento da empresa estar interessado em apenas uma parte de todos os dados existentes no DW, permitiu a criação de estruturas mais leves, o que alivia a carga imposta sobre a "janela de *down time*" disponível para refrescamento do DW, durante a qual este fica normalmente, indisponível para consultas. Todavia, ao mesmo tempo, muitas outras organizações pretendem instalar DWs, requerendo resultados rápidos, ainda que sectoriais. Estes factores conjugados motivaram o surgimento de um novo conceito: os DMs, que podem ser considerados como um *downsize* dum DW, um DW departamental. Estes, uma vez que são justificados por duas realidades opostas, podem ser de dois tipos: 1) aqueles que existem em conjunto com um DW central (abordagem centralizada - Figura 2.14–à esquerda); e 2) aqueles que resultam da criação, por cada departamento, de um DM próprio (abordagem descentralizada - Figura 2.14-à direita).

Neste último tipo, dado não existir o DW que proporcione uma visão global dos dados, deverá ser dotado de uma camada de *middleware* que se encarregará dessa função, assegurando que não

haverá ilhas de informação em cada departamento e inconsistências não desejadas, e ainda que a comunidade de decisores não fique limitada a visões parciais da empresa. Este tipo de DM também é denominado de DW Empresarial Distribuído de Área Local. Aqui, as soluções de optimização estáticas de índole espacial centralizada, acabadas de referir atrás, a propósito das arquitecturas tradicionais do DW, continuam a ser adequadas à selecção dos cubos mais benéficos.

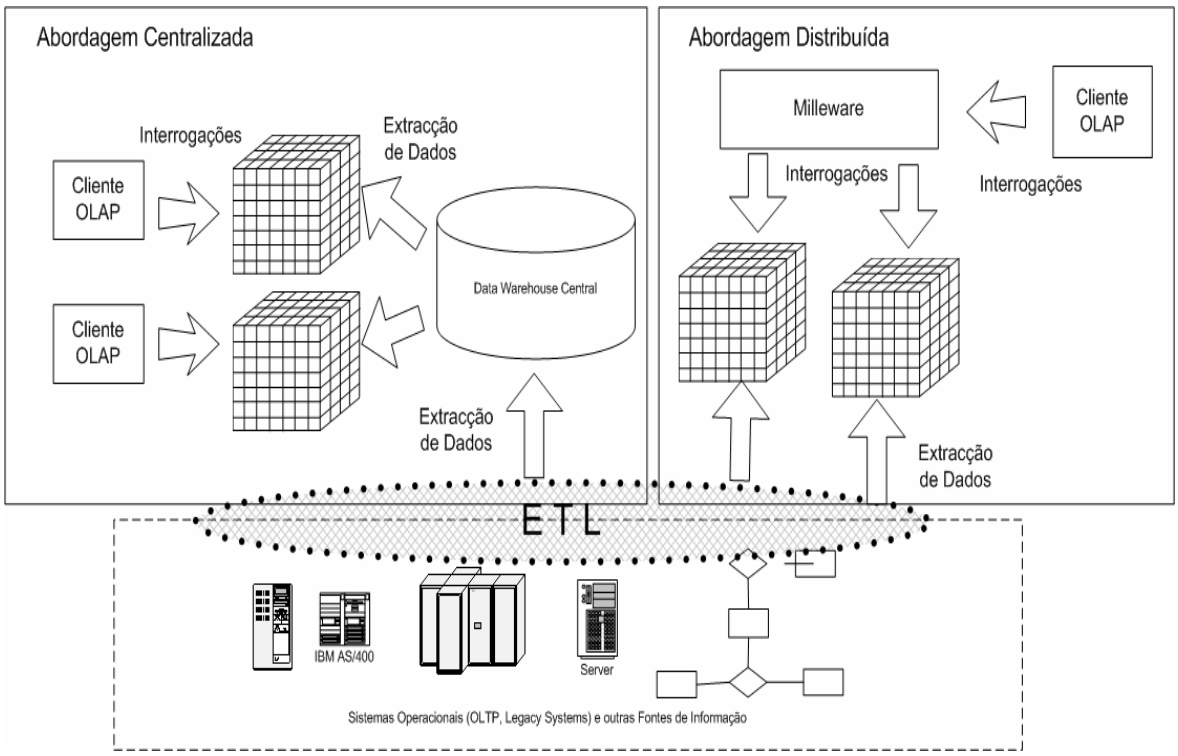


Figura 2.14. Duas abordagens possíveis para os Data Marts.

Se a inclusão de DM na arquitectura de um DWS se constitui como uma actuação inter-nível, uma acção interna, ao nível do repositório base (se aí se encontrarem materializadas as vistas correspondentes) ou no DM, pode ser procurada: a distribuição do cubo por vários nós, aproveitando o efeito de utilização de plataformas de hardware correntes, gera uma arquitectura OLAP multi-nó (M-OLAP)(Figura 2.15).

A arquitectura OLAP multi-nó é caracterizada por ser constituída por um conjunto de *n* nós servidores OLAP (NSO) interligados por uma rede de comunicações, que pode ser heterogénea, sem ser obrigatório que esteja completamente conectada, embora seja esta a situação considerada

no exemplo da Figura 2.15. Estes servidores serão capazes de responder a interrogações colocadas por uma comunidade de utilizadores, sendo caracterizados, obviamente, por um espaço de armazenamento e uma capacidade de processamento. Desta forma, o problema da selecção e alocação de cubos é constrangido pelo espaço de armazenamento por nó [Bauer & Lehner, 2003], havendo que considerar a capacidade de processamento de cada nó e o perfil das interrogações distribuídas, além da possibilidade de paralelização das tarefas, que pode ser conseguida devido ao paralelismo inerente à arquitectura [Loureiro & Belo, 2006f], [Loureiro & Belo, 2006i]. Esta característica será especialmente desejável para o processamento das operações relacionadas com o período de actualização (refrescamento) [Loureiro & Belo, 2006g], [Loureiro & Belo, 2006h], [Loureiro & Belo, 2006d], [Loureiro & Belo, 2006b], permitindo uma maior rapidez (assim uma menor janela de actualização) e a possibilidade de aplicar uma abordagem dinâmica. Ainda relativamente à arquitectura M-OLAP, facilmente se percebe que as facilidades de comunicação estendem as dependências entre os subcubos do *lattice* para além das fronteiras de cada nó. No próximo capítulo (secções 3.2 e 3.3) serão descritos dois modelos de custos e respectivos algoritmos de cálculo que podem ser considerados para esta arquitectura.

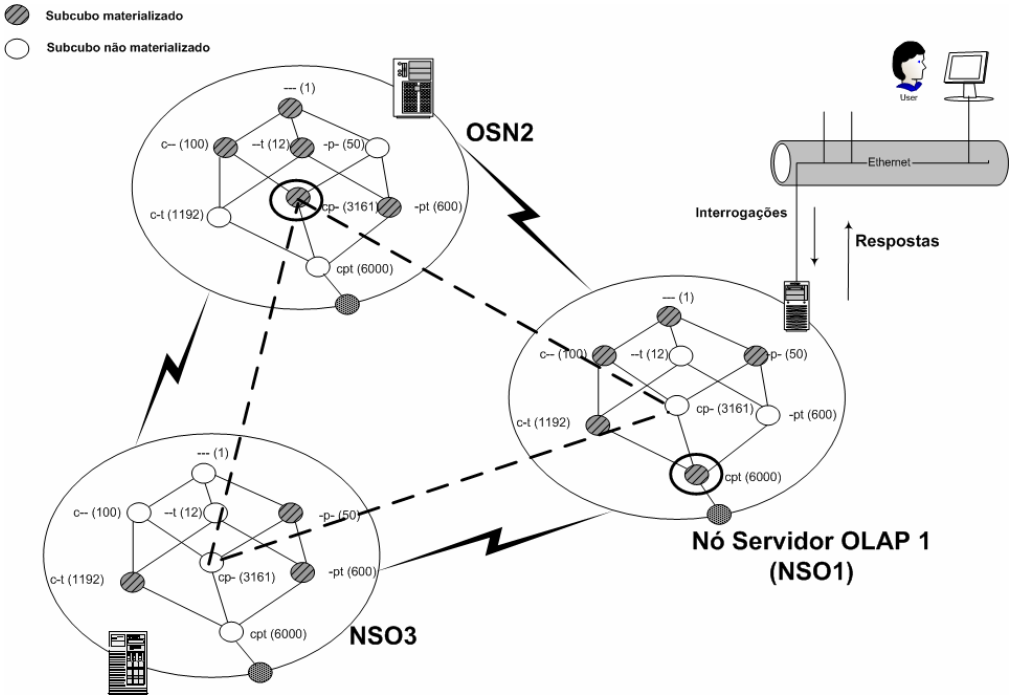


Figura 2.15. Exemplo de arquitectura OLAP multi-nó (M-OLAP).

A solução que podemos observar na Figura 2.14-b, se alargada geograficamente, vai constituir o denominado Data Warehouse Empresarial Distribuído, cuja arquitectura é apresentada na Figura 2.16, também chamado de Data Warehouse Federado ou Data Mart Empresarial [Informatica, 1997]. As vantagens desta abordagem são claras: os DMs individuais podem ser criados e geridos de forma flexível, mas sempre permitindo uma visão consolidada da informação da empresa na sua globalidade. O reverso da medalha é patente na complexidade adicional, relativa à gestão do esquema global e no processamento das interrogações, distribuído pelos vários DMs. Facilmente se percebe que quanto mais global a distribuição do DW maior será a importância do aumento de acesso local através de replicação de dados, o que vem mostrar a transição para o que poderá ser designado de DW Híbrido.

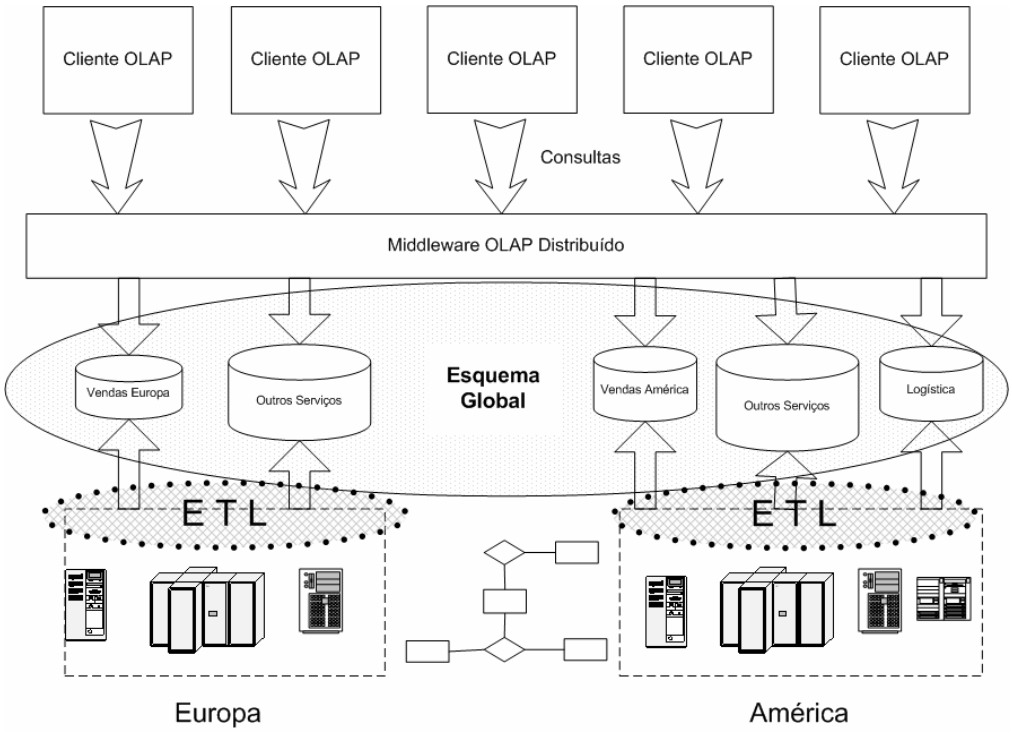


Figura 2.16. Arquitectura de um Data Warehouse Empresarial Distribuído.

Nesta arquitectura poderemos ter uma combinação de algumas das arquitecturas apresentadas anteriormente, a que poderão ser acrescentadas novas camadas de armazenamento ou processamento a nível mais local. Veja-se a importância da sua existência. No estado actual da tecnologia, o acesso a dados remotos é condicionado pelos custos bastante elevados a nível de

comunicações. Isto faz pensar em replicar algumas estruturas, procurando aproximar os dados da sua utilização. Esta replicação poderá assumir a forma de uma ou mais camadas de servidores intermédios, e mesmo de estruturas localizadas nos próprios clientes. Como se pode imaginar, esta diversidade permite constituir arquitecturas bastante complexas e distintas. No limite, poder-se-á ter algo como o que se mostra na Figura 2.17. A vantagem da inclusão das novas camadas de servidores OLAP intermédios reside na possibilidade de adaptação activa (dinâmica), como o apresentado em [Scheuermann et al., 1996], [Kotidis & Roussopoulos, 1999], [Roy et al., 2000], [Kalnis et al., 2002a] ou pró-activa [Belo, O., 2000], [Sapia, 2000], [Park et al., 2003] das estruturas multidimensionais aí residentes, ao perfil de carga dos clientes.

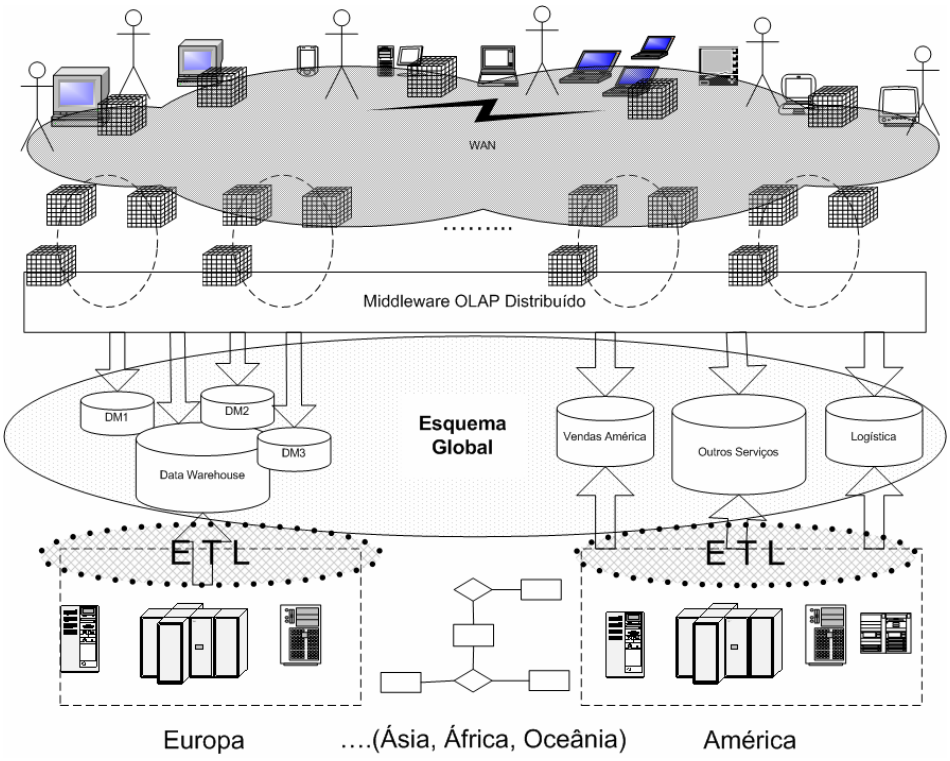


Figura 2.17. Data Warehouse Híbrido Global Distribuído.

Capítulo 3

Modelos e Algoritmos de Estimativa de Custos

3.1 Modelo Linear Centralizado

O modelo linear centralizado foi apresentado pela primeira vez em [Harinarayan et al., 1996] e é suporte de muitas das propostas algorítmicas vocacionadas à selecção de vistas a materializar para repositórios centralizados.

Suponhamos um esquema relacional simplificado de um DW com apenas três dimensões e sem quaisquer hierarquias, cuja tabela de factos representa as vendas de produtos:

```
Fact_Vendas (Cliente, Produto, Fornecedor, Vendas)
```

Um esquema como este, embora simples, serve no entanto para responder a várias interrogações, podendo o decisor avaliar as vendas segundo as diversas perspectivas de análise: cliente, fornecedor e produto. Lançando mão do conceito de cubo introduzido na secção 2.2, poder-se-á considerar que as vendas podem ser conceptualizadas na forma do cubo de dados, consistindo em três dimensões: CLIENTE, PRODUTO e FORNECEDOR. Cada célula conterá o valor das vendas efectuadas a um determinado cliente, de um dado produto e adquiridos a um determinado fornecedor, vendas estas correspondentes a um único conjunto de valores para as três dimensões. O cubo de dados correspondente é mostrado na Figura 3.1 (idêntica a uma das ilustrações da Figura 2.2 - aqui repetida para mostrar o conceito de agregação multidimensional e dependências no *lattice*), que inclui apenas cinco produtos, quatro clientes e três fornecedores.

Se for necessário saber as vendas de cada produto por fornecedor, teremos de efectuar uma agregação ao longo da dimensão cliente, relativamente aos diferentes produtos e fornecedores. Em SQL teríamos algo do género:

```
SELECT Produto, Mês, SUM (Vendas) AS Vendas_Totais
FROM Fact_Vendas
GROUP BY Produto, Fornecedor
```

Esta agregação corresponderia à "fatia" mostrada na Figura 3.1 a sombreado. O subcubo correspondente à agregação pode ser representado como (*-pf*), onde o símbolo "-" (cujo significado foi já discutido na secção 2.2) representa a situação nas quais o atributo da dimensão (neste caso, cliente) não está presente na cláusula GROUP BY. A agregação segundo a combinação destas três dimensões leva à obtenção dos oito subcubos possíveis (2^d), cujas relações podem ser modeladas através do *lattice* já apresentado na secção 2.2 e que se apresenta, novamente, na Figura 3.2, modificado para atender às dimensões em causa e incluir, para cada subcubo, o tamanho respectivo e frequência de utilização. O tamanho apresentado corresponde ao número de células não nulas do subcubo em consideração.

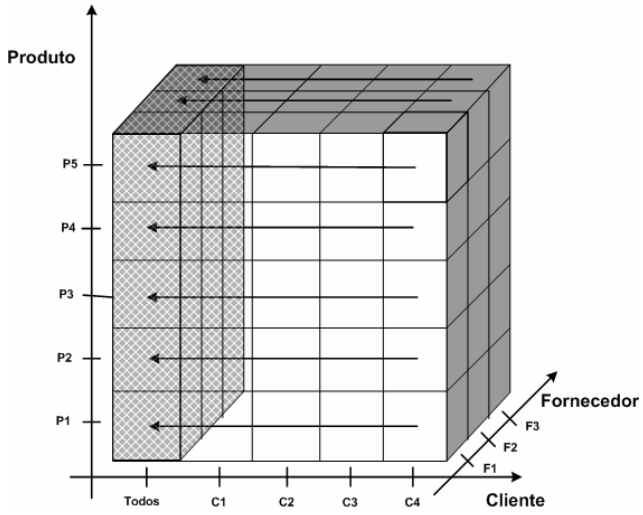


Figura 3.1. Cubo relativo às vendas de produtos, com dimensões produto, cliente e fornecedor.

A ligação entre quaisquer dois subcubos no *lattice* representa a relação de dependência entre eles. A resposta a uma interrogação relativa às vendas por cliente, dadas as relações entre os subcubos

mostrados na Figura 3.2, pode ser respondida pelos subcubos (*cpf*), (*cp-*), *(c-f)* ou (*c--*) a custos sucessivamente mais baixos, 6M, 6M, 5M e 0.1M, respectivamente. Os mesmos considerandos se poderiam aplicar relativamente a quaisquer dois subcubos colocados num dado caminho entre os dois vértices (*cpf*) e (---). Poder-se-á dizer que $c_i \preceq c_j$ (o subcubo c_i é dependente de c_j), se qualquer interrogação que possa ser respondida por c_i o possa ser também por c_j, onde $\preceq$ representa a relação de dependência (derivado-de, ser-calculado-de). Voltando ao exemplo, $(cp-) \preceq (cpf)$ e $(c--) \preceq (cp-) \vee (cpf)$. Deve notar-se que esta relação de dependência não se verifica no sentido inverso $c_j \not\preceq c_i$, ou seja, nem todas as interrogações respondidas por c_j o serão por c_i. Por exemplo, (*cp-*) na Figura 3.2 não pode responder à interrogação "mostrar as vendas agrupadas por cliente, produto e mês".

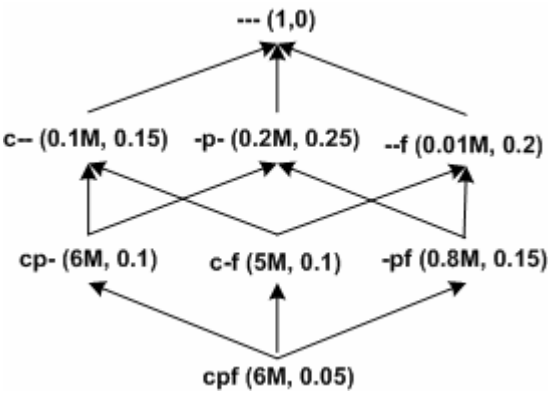


Figura 3.2. *Lattice* correspondente às dimensões cliente, produto e fornecedor.

3.1.1 Modelo de Custos e Fórmulas de Cálculo

As relações de dependência entre subcubos impõem uma ordenação parcial dos subcubos. Então, dado um *lattice* $(L, \preceq)$, onde L é o conjunto possível de vistas e $\preceq$ as respectivas relações de dependência, e um subcubo c_i de um subconjunto $M \subseteq L$, a relação antecessor é definida como $Anc(c_i, M) = \{c_j \mid c_j \in M \ \wedge \ c_i \preceq c_j\}$, ou seja, o antecessor de c_i será um qualquer subcubo c_j pertencente a *M* capaz de responder a qualquer interrogação que possa ser respondida por c_i, ou

seja que c_i dependa de c_j. A relação inversa, denominada descendente, pode ser definida como $Desc(c_i, M) = \{c_j \mid c_j \in M \ \wedge \ c_j \preceq c_i\}$.

Em [Harinarayan et al., 1996] foi demonstrado que o tempo de execução da resposta (*Rt*) a uma interrogação variava quase linearmente com o tamanho do subcubo *S* utilizado para sua resposta, ou seja $Rt = m^* \mid S \mid + c$, onde *m* é a razão do tempo de resposta à interrogação e $\mid S \mid$ é o tamanho do subcubo *S* utilizado para lhe dar resposta, e *c* o custo fixo, a sobrecarga correspondente à execução de uma interrogação de tamanho desprezível. Esta variação linear determinou o nome dado ao modelo (de custos linear) e, por outro lado, a possibilidade de desenvolver todo um conjunto de algoritmos de cálculo de custos e outros tendentes à sua minimização.

Como o objectivo é responder a uma qualquer interrogação da forma mais rápida possível, há que usar o subcubo que implique um custo mínimo, o seu antecessor mínimo, que pode ser definido como $Manc(c_i, M) = \min_{c_j \in Anc(c_i)} \mid c_j \mid$, já que $Ci(C_i) = \propto \mid C_j \mid$, ou seja, o custo de uma qualquer interrogação é proporcional ao tamanho do subcubo utilizado para a sua resposta. Similarmente, pode definir-se o chamado maior descendente como $Mdesc(c_i, M) = \max_{c_j \in Desc(c_i)} \mid c_j \mid$.

Voltando ao exemplo do *lattice* apresentado na Figura 3.2, os antecessores de (*c*--) são (*cp*-), (*c-f*) e (*cpf*). Já o antecessor mínimo é (*c-f*), visto que o tamanho é 5M registos, ao passo que, para os outros dois, tem-se um tamanho de 6M registos.

Considere-se agora o exemplo da Figura 3.3. Os custos de resposta à interrogação *i1* $Ci(i_1, M) = \min\{\mid (--f) \mid, \mid (cpf) \mid\}$ ou, generalizando,

$$Ci(c_i, M) = Manc(c_i, M) = \min_{c_j \in Anc(c_i)} \mid c_j \mid \qquad \text{(eq. 3.1).}$$

Os custos de resposta ao conjunto de interrogações $I = \{i_1, i_2, ..., i_n\}$ com frequências $F = \{fi_1, fi_2, ..., fi_n\}$, será $\sum_{i=1}^{n} fi_i . Ci(i_i, M)$ ou $\sum_{i=1}^{n} fi_i . Manc(c_i, M)$, já que uma interrogação agregada i_i é representada pelo correspondente subcubo c_i. Finalmente, usando a eq. 3.1, ter-se-á:

$$Ci(I,M) = \sum_{i=1}^{n} fi_i \min_{c_j \in Anc(c_i)} |c_j| \qquad \text{(eq. 3.2).}$$

Conforme atrás foi referido (secção 2.6), os custos de manutenção revelam um comportamento não monotónico, podendo o próprio processo de manutenção ser levado a efeito de duas formas distintas: por recriação completa (denominada manutenção integral - *from scratch*) ou aplicando técnicas de manutenção incremental. Esta maior complexidade e variedade imporá, necessariamente, uma maior complexidade do próprio modelo e equação de cálculo. Para simplificar, vai aqui considerar-se que apenas vão ser inseridos novos tuplos na(s) tabela(s) factos das relações base. Esta suposição faz sentido, já que os dados armazenados no DW estão em expansão contínua, sendo as modificações muito limitadas. Além disso, assume-se que as inserções não violam quaisquer constrangimentos de integridade referencial no esquema em estrela.

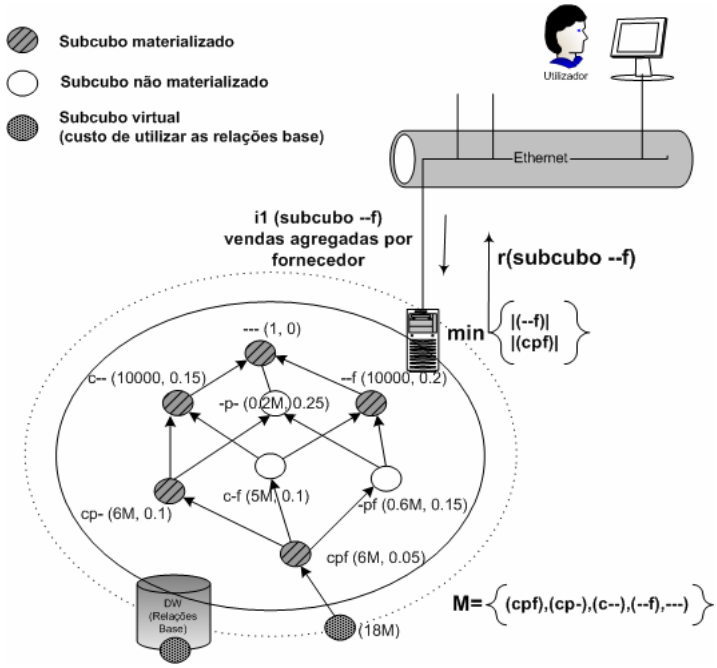


Figura 3.3. Custos correspondentes à resposta à interrogação do tipo "vendas por fornecedor".

Considerando a primeira abordagem à manutenção (integral), os custos de manutenção de um subcubo a materializar são análogos aos custos de resposta a uma interrogação similar. Ou seja,

$Cm(c_i, M) = |\, Manc(c_i, M) \,| = \min_{c_j \in Anc(c_i)} |\, c_j \,|$. Se *fm* representar a frequência de actualizações no período em consideração, teremos um custo de manutenção obtido a partir de:

$$Cm(M) = fm. \sum_{i=1}^{n} \min_{c_j \in Anc(c_i, M')} |\, c_j \,| \qquad \text{(eq. 3.3)},$$

em que *M* será o resultado final da materialização dos sucessivos subcubos, podendo considerar-se a existência de um *M´* que constituirá, em cada passo, o conjunto dos subcubos já materializados (considerados para custos) e que irá sendo adicionado dos sucessivos subcubos entretanto já refrescados, estando, portanto, no início vazio e devendo no final ser igual a *M*, ou seja,

$$M^{'} \subseteq M \,|\, \begin{cases} M_0^{'} \leftarrow \varnothing \\ M_{i+1}^{'} \leftarrow M_i^{'} \cup \{c\} : c \in M \setminus M_i^{'}; i = \{0, ..., \#M\} \end{cases}.$$

A segunda abordagem à manutenção resulta em custos bastante menores, pois que a extensão das células (ou tuplos) das estruturas multidimensionais afectadas pelas alterações é, em regra, limitada. Também os deltas[6] [Mumick et al., 1997] gerados são, normalmente, muito menores do que os correspondentes subcubos. Esta redução de impacto das alterações pode, neste modelo simplificado, ser aproximado pela inclusão de um peso adicional (*em*), não mais do que um factor correspondente à extensão do impacto das alterações em cada subcubo e nos custos de geração do próprio delta. Desta forma, a eq. 3.3 ficará:

$$Cm(M) = fm. \sum_{i=1}^{n} \min_{c_j \in Anc(c_i, M')} |\, c_j \,|.em \qquad \text{(eq. 3.4)}.$$

[6] Deltas ou tabelas de sumarização delta representam as alterações líquidas à tabela de sumarização (ou subcubo) correspondente, devidas às alterações ocorridas nas relações base (p. ex. tabela factos). Na prática, cada delta é gerado calculando as agregações correspondentes a um dado subcubo, relativas à alterações ocorridas nas relações base. Este delta pode depois ser utilizado para actualizar o subcubo correspondente ou ainda ser agregado para calcular outros deltas a usar para actuualizar outros subcubos. Esta última situação é normalmente vantajosa (em relação ao cálculo dos outros deltas utilizando as alterações havidas nas relações base) já que o delta origem, sendo já uma agregação, deverá implicar o acesso a um menor de tuplos do que as alterações havidas nas relações base.

3.1.2 Algoritmo de Estimativa de Custos de Interrogação

O cálculo dos custos de interrogação é bastante simples, e resume-se a não mais do que uma aplicação directa da eq. 3.2. Como dados necessários ao cálculo tem-se: 1) $(L, \preceq)$, os subcubos, tamanhos e relações de dependência (informação relativa ao próprio cubo); e 2) o perfil de interrogações, aqui representado na respectiva frequência. Para facilitar a especificação de $(L, \preceq)$, e dada a possibilidade de se estar a lidar com cubos complexos, optou-se por incluir um serviço de geração e disponibilização das dependências, utilizando informação acerca de cada subcubo (o seu código), e relativa às dimensões e hierarquias respectivas (suportando hierarquias paralelas). Assim sendo, o algoritmo usa, como dados de entrada:

- os subcubos do *lattice*, sendo cada um codificado usando a notação (xyz...) que tem vindo a ser utilizada para designar cada subcubo, em que cada símbolo representa o nível da dimensão e hierarquia a que é efectuado o *group-by* e que deverá ter valores distintos em cada dimensão, mas que podem ser repetidos em dimensões diferentes;
- o tamanho de cada subcubo (em tuplos), dado o modelo de custos linear;
- a frequência das interrogações, *F*.

Assim, e atendendo à eq. 3.2, o algoritmo deverá, para cada interrogação, procurar o antecessor mínimo em *M*. Dada a previsível utilização intensiva do algoritmo, o seu tempo de execução deve ser o mais reduzido possível. Neste sentido, a pesquisa do antecessor mínimo vai ser disponibilizada por um serviço que usa uma matriz de adjacência [Cormen et al., 2001], relativa às correspondências do tipo (interrogação → possíveis subcubos a utilizar). A natureza do problema mostra uma sobreposição de subproblemas: um mesmo subcubo pode ser utilizado para resposta a várias interrogações. Por outro lado, o cálculo das relações de dependência pode ser efectuado uma vez e armazenado para utilização posterior. Assim, o emprego de programação dinâmica [Bertsekas, D., 2000] foi determinante para reduzir o tempo de execução do algoritmo. O método utiliza, para a solução de um problema, um processo de três passos que, instanciados ao problema em causa, podem ser assim descritos:

1. o problema (encontrar o antecessor de custo mínimo) foi dividido em problemas mais simples (cálculo dos possíveis antecessores de um qualquer subcubo e, de entre estes, o de custo mínimo);

2. cada um dos subproblemas foi resolvido optimamente (encontrado o custo de cada par subcubo - subcubo utilizável), ou seja calcularam-se para cada subcubo, todos os possíveis antecessores;
3. para resolver o problema original, basta depois, de entre os possíveis antecessores (já conhecidos), procurar o de custo mínimo (abordagem denominada de *memoization*).

Na verdade, a solução para cada um dos subproblemas é utilizada repetidamente, dada a possível utilização do mesmo subcubo como antecessor de muitos outros e devido à natureza altamente iterativa dos algoritmos de optimização, onde a estimativa de custos vai ser utilizada. A inclusão desta tecnologia permitiu um aumento da eficiência do tempo de execução de três ordens de grandeza, segundo testes efectuados. Desta forma, ao saber-se qual a interrogação, a simples inspecção desta matriz permite logo conhecer o respectivo antecessor mínimo. No Algoritmo 3.1 apresenta-se uma descrição formal do algoritmo de estimativa de custos de interrogação.

```
Input: L // Lattice com todos os subcubos, dependências e tamanhos
       Fi=( Fi1, Fi2,...., Fin) // Distribuição das interrogações
       M=(Cu1, Cu2...., Cun) // Subcubos materializados
Output:C(I,(M,Fi))// custo de resposta às interrogações I de freq. Fi,
       dado M
1. Inicialização: C(I) ← 0;
2. Calcular e totalizar custo de resposta a cada uma das
    interrogações colocadas
   Para cada interrogação i, fazer:
    Ci ← ∞; // inicializar custo de resposta a i com um valor máximo
    Para cada antecessor Ca=Anc(i), fazer:
     Canc(i,M) ← |Ca|; // custo de resposta pelo antecessor de i
     Se Ci > Canc(i,M) então Ci ← Canc(i,M);
    C(I) ← C(I) + Ci; //totalizar custo das sucessivas interrogações
3. Devolver Resultado:. Return C(I)
```

Algoritmo 3.1. Algoritmo de estimativa de custos de interrogação.

Dada a complexidade e simplificação conseguida com a inclusão do serviço de geração das dependências entre subcubos no *lattice*, e para uma melhor compreensão deste algoritmo, importa discutir alguns detalhes da sua concepção.

Na prática, o que se pretende fazer é carregar uma matriz com a informação dos subcubos capazes de responder a uma dada interrogação, ou dos antecessores do seu subcubo correspondente. Como o algoritmo deverá suportar dimensões com hierarquias paralelas, considere-se o *lattice* mostrado na Figura 3.4. Se for colocada uma interrogação agregada por

região e tipo de produto (rt), os subcubos capazes de lhe responder serão (rp), (ct) e (cp), ou seja, um qualquer subcubo que tenha na primeira dimensão a própria dimensão/hierarquia ou uma mais abaixo nessa dimensão (r ou c), aplicando-se o mesmo tipo de heurística para a segunda dimensão (neste caso, valores possíveis, t ou p). Outros exemplos possíveis podem ser:

1. para a interrogação relativa às vendas por tipo de produto (-t), os subcubos respondentes possíveis serão (-t) - o subcubo correspondente e um seu antecessor, (rt), (-p), (rp), (ct) e (cp), ou seja, um qualquer subcubo que tenha na primeira dimensão um qualquer membro dessa dimensão, e, na segunda dimensão, t ou p, sendo igualmente válida a heurística atrás deduzida;
2. para a interrogação (--), qualquer subcubo é elegível (um qualquer membro em qualquer dimensão);
3. para responder à interrogação agregada por região e produto, só os subcubos (rp) e (cp) são elegíveis, confirmando mais uma vez a heurística enunciada.

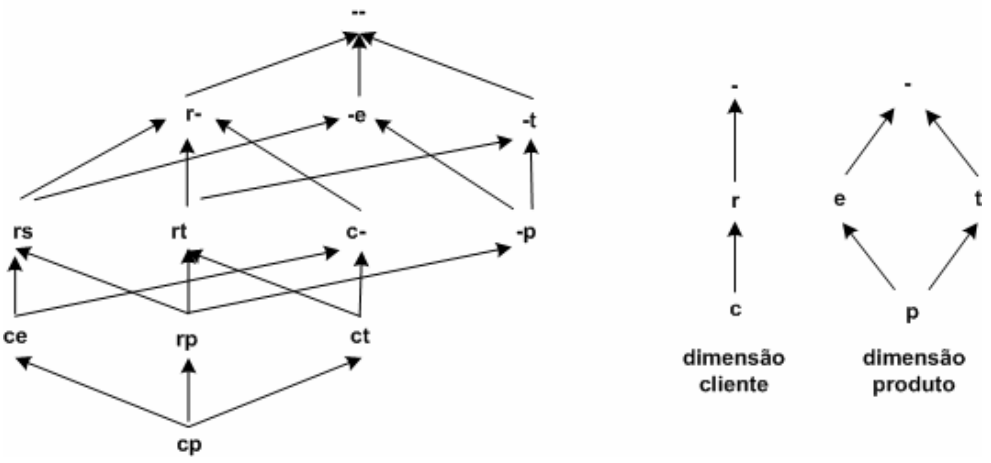


Figura 3.4. *Lattice* relativo às dimensões cliente e produto e respectivas hierarquias.

Assim, em resumo, os subcubos antecessores de um determinado subcubo poderão encontrar-se:

1. procurando outros que, em cada dimensão, tenham um membro igual ou de nível mais baixo na hierarquia nessa dimensão;
2. só os subcubos que satisfaçam a condição anterior em todas as dimensões, é que serão os seus antecessores.

No Algoritmo 3.2 apresenta-se a descrição formal deste algoritmo.

```
Input: L // Lattice com todos os subcubos
       EDW // dimensões e hierarquias do esquema multidimensional
Output:MDep:{I_i ≤ C_j} // conjunto das dependências: matriz com os subcubos
       // capazes de responder a cada interrogação agregada
1. Inicialização: MDep ← ø; // limpar conjunto de dependências
2. Procurar antecessores para cada subcubo e carregar MDep com resultado
   Para cada subcubo i, fazer:
    // Carregar Mt[j][k] (uma matriz transitória de correspondências) com os subcubos
    // antecessores [k] do subcubo em análise, para cada dimensão [j]
    Mt[j][k] ←false; // inicializar a matriz transitória
    Para cada dimensão j, fazer:
     Para cada subcubo k, fazer:
      Para cada nível l de hierarquia na dimensão j, fazer:
       // a dimensão j pode ser uma das hierarquias paralelas dessa dimensão
       Se caracter (CodSubcubo i[j])=caracter correspondente a membro da
        dimensão j do nível k ou inferior), Então
        Mt[j][k] ←true;  // o subcubo k, na dimensão j, qualifica como antecessor
          do subcubo i
    // Em Mt[j][k] tem-se agora o valor true se o subcubo k
    // qualificar para o subcubo i, relativamente à dimensão j
    // Mas o subcubo k só é um antecessor de i se qualificar para todas as dimensões
    // Fazer o AND para todas as dimensões e no final carregar MDep com o resultado
    Para cada dimensão j \0, fazer:
     Para cada subcubo k, fazer:
      Mt[0][k] ← Mt[0][k] AND Mt[j][k];
    Para cada subcubo k, fazer:
     Dep[i][k] ← Mt[0][k];
3. Devolver Resultado:Return MDep
```

Algoritmo 3.2. Algoritmo de geração de dependências.

3.1.3 Algoritmo de Estimativa de Custos de Manutenção

Comparando as eq. 3.2 e 3.3, as semelhanças são notórias. Supondo, numa primeira abordagem, que vai ser considerado um esquema de manutenção do tipo integral, o algoritmo será bastante semelhante ao Algoritmo 3.1. Há, contudo, que ter em atenção que materializar M é o objectivo. Definindo M' como o conjunto dos subcubos já efectivamente materializados (considerados para custos) e passíveis de ser utilizados para gerar outros subcubos ainda a materializar, inicialmente M' estará vazio, sendo sucessivamente acrescentado de subcubos até que $M' \leftarrow M$. A característica de não-monotonicidade dos custos de manutenção vai também implicar que a ordem de materialização determine o seu valor. Se a ordem pode ser determinada exteriormente (como é o caso do cálculo dos custos de manutenção, quando são utilizadas heurísticas construtivas no

problema da selecção de cubos, tipo *greedy*, por exemplo), o oposto é igualmente válido: é o caso dos algoritmos evolucionários, nos quais os sucessivos M não têm qualquer correlação obrigatória. Como o algoritmo a desenvolver deverá ser de utilização genérica, já que será utilizado em algoritmos de optimização baseados em heurísticas várias, deverá ser capaz de seleccionar uma ordem de materialização.

Conhecidas as relações de dependência entre subcubos no *lattice*, e sabendo que estas determinam os subcubos a usar para materializar outros, é forçoso concluir que estas poderão fornecer a base para possíveis heurísticas a utilizar. Assim, poder-se-á usar uma ordem determinada pelo nível dos subcubos no *lattice*, iniciando nos níveis mais baixos; olhando para o *lattice* apresentado na Figura 3.4, ter-se-ia: (cp), (ce), (rp), (ct), (rs), (rt), (c-), (-p), (r-), (-e), (-t), (--).

Outra heurística, porventura mais eficaz, consiste na utilização de um esquema tipo *greedy*, empreendendo, em cada estágio, uma pesquisa do subcubo a materializar que maior benefício trouxer em termos das futuras materializações. Esta heurística implicará, decerto, um custo computacional muito superior, o que poderá ser lesivo da desejada eficiência do algoritmo, devendo assim ser de utilização cautelosa (implicando uma análise cuidadosa em termos de custos X benefícios).

Pode-se também, simplesmente, usar uma ordem especificada pelo utilizador, que pode, inclusivamente, ser objecto de alguma afinação prévia. Pesadas as alternativas, optou-se pela utilização de uma heurística do terceiro tipo, pela sua simplicidade e provável eficiência. As outras alternativas, possivelmente mais eficazes, serão investigadas aquando do ataque de problemas mais complexos, como será o caso das arquitecturas M-OLAP nas quais o paralelismo introduz um maior grau de liberdade e heurísticas mais elaboradas poderão trazer uma mais-valia. Em resumo, o algoritmo de estimativa de custos de manutenção poderá ser descrito da seguinte forma:

1. inicialmente $M' = \varnothing$, ou seja, não há qualquer subcubo efectivamente materializado;
2. para cada subcubo C_i em M, numa ordem especificada exteriormente, procurar em M' o seu antecessor mínimo, ou usar as relações base, se não houver em M' um qualquer subcubo capaz de lhe dar resposta;
3. adicionar C_i a M e considerar os custos em termos dos custos de manutenção totais;
4. repetir 1 e 2 até que $M' = M$.

A descrição formal do algoritmo de estimativa de custos de manutenção é apresentada em Algoritmo 3.3.

```
Input: L // Lattice com todos os subcubos, dependências e tamanhos
     EDW // dimensões e hierarquias do esquema multidimensional
     MDep // conjunto das dependências: matriz com os subcubos
          //capazes de responder a cada interrogação agregada
     M=(Cu1, Cu2,...,Cun)// Subcubos a materializar ou actualizar na ordem
          // especificada
Output: C(M) // custo de manutenção de M

1. Inicialização:
   C(M)← 0; // inicialização do custo de manutenção
   M'←Ø; // no início não há qualquer subcubo efectivamente materializ.ou actualizado
2. Calcular e totalizar custo de manutenção de cada subcubo a materializar
   Para cada subcubo c em M ∉ M', fazer: // para cada subcubo em M
                                 // ainda não materializado ou actualizado
    Se Não Existir Ant(c) em M' então // se não existe um antec. já materializ.
     Cm ← custo correspondente a geração usando as relações base
    Senão // se houver um antecessor em M', calcular o antecessor de custo mínimo
     Cm ← ∞; // inicializar custo de antecessor com um valor máximo
     Para cada antecessor Ca=Anc(c) ∈ M', fazer:
      Cant(c,M')←|Ca|; // custo de manutenção de c pelo antecessor Ca
      Se Cm > Cant(c,M') então Cm ← Canc(c,M');
   M' ← M' ∪ c // c passa a efectivo
   C(M) ← C(M) + Cm; //totalizar custo de manutenção dos sucessivos subcubos em M
3. Devolver Resultado: Return C(M)
```

Algoritmo 3.3. Algoritmo de estimativa de custos de manutenção para uma arquitectura OLAP centralizada.

3.2 Modelo Linear Distribuído

Basicamente, este modelo é uma extensão do modelo descrito anteriormente, sendo o suporte à optimização das soluções arquitecturais de distribuição intra-nível (secção 2.9), mantendo a característica linear do modelo base.

A ideia da distribuição das estruturas emerge, naturalmente, da migração para o mundo informacional das vantagens obtidas do seu uso em sistemas operacionais: a diminuição da complexidade, o evitar de estrangulamentos, a deslocação dos locais de distribuição para mais perto do consumo e, naturalmente, a redução de custos, já que um servidor OLAP de grande capacidade (de armazenamento e processamento) tem um custo bastante superior a vários servidores de capacidade agregada idêntica [Loureiro & Belo, 2006b]. Em resumo, a distribuição das estruturas multidimensionais surge em resposta à necessidade de escalabilidade OLAP.

Relembrando e pormenorizando o que discutiu anteriormente, nesta arquitectura, tem-se um conjunto de *N* nós-servidores OLAP (*NSO*), cada um com um espaço de armazenamento (*Es*) e capacidade de processamento (*Cp*) determinado. A sua arquitectura simplificada pode ver-se na Figura 3.5. Nesta figura, são mostrados três nós NSO e o nó 0 (onde residem as relações base – um DW, por exemplo). Neste modelo, teremos não só relações de dependência intra-nó, descritas e discutidas no modelo apresentado na secção anterior (típicas do *lattice* de dependências), mas também as relações de dependência inter-nós que permitem a resposta a uma interrogação, utilizando um subcubo existente num outro nó, a um custo adicional, correspondente aos custos de comunicação envolvidos.

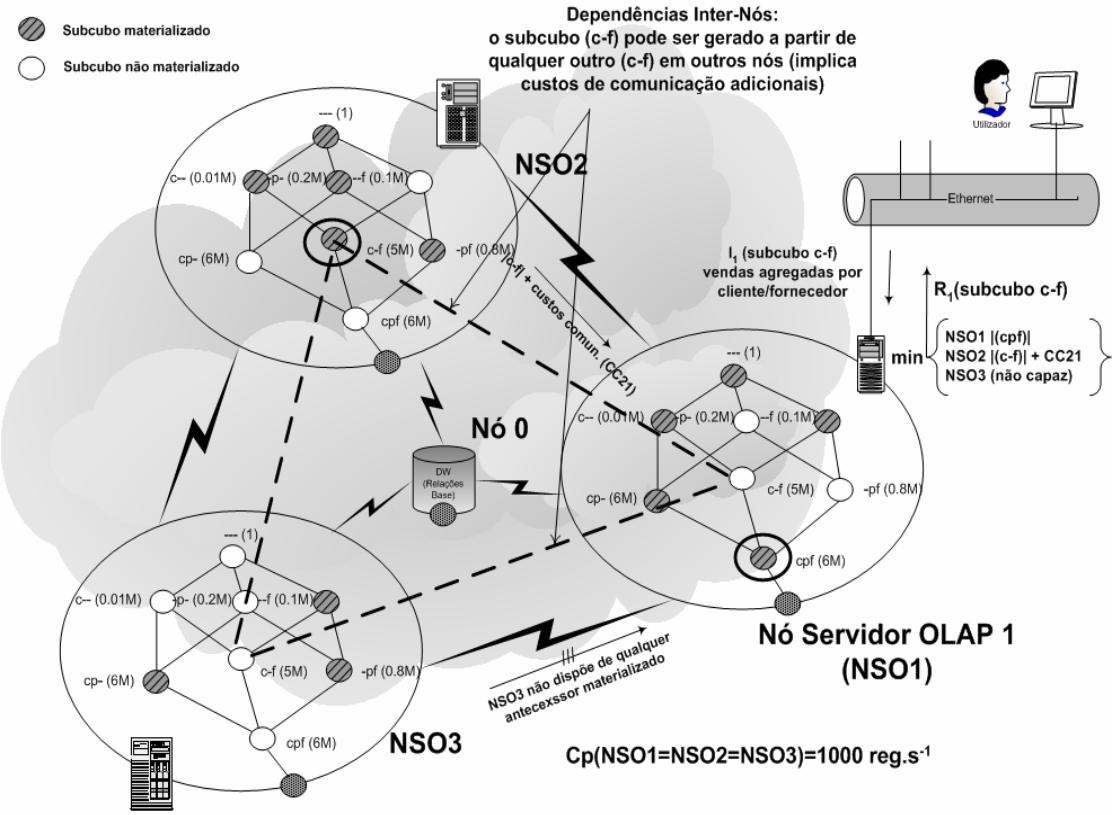


Figura 3.5. Arquitectura OLAP Multi-Nó (M-OLAP) e dependências inter-nó.

No caso da Figura 3.5, está-se a supor uma rede local a interligar os diversos nós. Esta arquitectura pode, no entanto, assumir duas variantes e algumas combinações, consoante a área física abrangida: 1) os nós estarão localizados numa pequena área (edifício ou campus); 2) nós em localizações fisicamente distantes; ou 3) uma combinação das duas anteriores.

3.2.1 Modelo de Custos Distribuído e Fórmulas de Cálculo

Na arquitectura M-OLAP que irá ser aqui discutida, está a considerar-se uma variante do tipo 1 referido no final da secção anterior. A rede de comunicação que interliga os diversos NSOs pode assumir diversas formas, mas aqui será caracterizada por um conjunto de parâmetros genéricos como os definidos em [Walrand & Varaiya, 2000]:

- o débito binário (DB) ou largura de banda, relativo ao número de bits (líquido ou total) que podem atravessar o canal de comunicação por unidade de tempo (em bps),
- a latência (*La*), valor do tempo que medeia o pedido de comunicação e o seu início efectivo e
- o tamanho de pacote (Tp).

Para as variantes 2 e 3 atrás referidas, outros parâmetros deveriam ser considerados como [Walrand & Varaiya, 2000], [Huang & Chen, 2001]: atraso em trânsito (AT), tempo de estabelecimento de uma ligação (TEL), utilizado em conexões em modo de não-ligação, e tempo gasto no estabelecimento de um canal virtual (TECV), típicos para redes tipo WAN. Assim, para áreas restritas, o custo de comunicação de transmissão de dados entre os nós *i* e *j* (CC_{ij}) podem ser assumidos como lineares e dados por:

$$CC_{ij} = Np * Sp / BD + La \qquad \text{(eq. 3.5)},$$

na qual *Np* representa o número de pacotes de dados a transferir e *Sp* o tamanho de cada pacote.

A existência dos canais de comunicação vai gerar uma nova rede de dependências, atrás nomeadas de dependências inter-nós - apresentadas na Figura 3.5 a tracejado forte – que, neste caso, são apenas as relativas ao subcubo (*c-f*). Estas dependências vão existir entre todos os subcubos de todos os nós (se a malha da rede for completamente conectada), gerando o *lattice* de agregações distribuído [Bauer & Lehner, 2003] (Figura 3.6). O conceito de *lattice* de agregação é agora alargado de forma a capturar a distribuição pelos vários NSOs. Desta forma, além das setas que representam as dependências (de agregação) intra-nó, outras foram introduzidas para considerar os canais de comunicação relativos ao cenário distribuído, por exemplo entre os nós (*cpf*) e (*cp*-) no *lattice* da Figura 3.6 nos vários NSOs, que correspondem à dependência inter-nó relativa aos subcubos (*cpf*) e (*cp*-). O *lattice* completamente expandido deveria incluir outros conjuntos de setas, um para cada nó, que, para simplificar a figura, foram omitidos e apenas representados pelas linhas tracejadas.

Veja-se agora como utilizar estas dependências estendidas para resposta a uma interrogação. Considere-se o caso apresentado na Figura 3.5. Um utilizador, conectado ao NSO1, pretende conhecer o total das vendas agregadas por cliente/fornecedor, ou seja, em termos de agregações, coloca a interrogação (*c-f*). O subcubo (*c-f*), ou um qualquer outro que, através de agregações, possa gerá-lo, poderão ser utilizados, independentemente de estarem materializados no NSO1 ou em outro nó qualquer. Desta forma, a resposta à interrogação poderá ser fornecida por (*cpf*) de NSO1 ou (*c-f*) de NSO2, havendo neste último caso um custo adicional correspondente aos custos de comunicação entre NSO2 e NSO1. Como o objectivo será, em regra, a minimização de custos, desde que disponível um modelo de custos, um qualquer agente, que pode residir em cada um dos nós, poderá decidir acerca do redireccionamento ou não de uma qualquer interrogação. Neste caso, já que em NSO2 reside a resposta directa, deverá ser este o nó respondente, excepto se os custos de comunicação ultrapassarem a diferença para os custos de busca e agregação a incorrer com a utilização do subcubo (*cpf*). Em resumo, os custos de resposta a uma interrogação serão a soma: 1) dos custos de busca (e agregação) - *Csa* - do subcubo utilizado na resposta e 2) dos custos de comunicação (*Ccom*), se o nó respondente for diferente do nó onde é colocada a questão. Ou seja, $Ci(i_i) = Csa(Anc(c_i)) + Ccom(c_i)$, atendendo a que $i_i = c_i$.

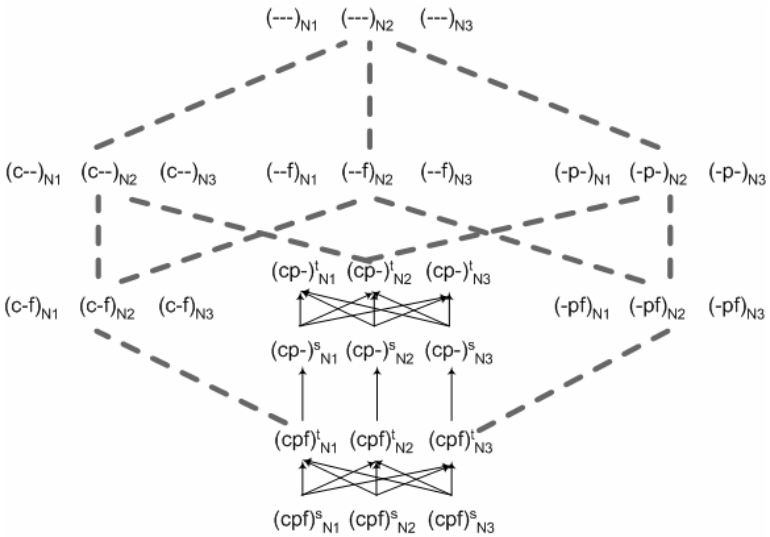


Figura 3.6. Lattice de agregação distribuído, com as dependências intra e inter-nó [Bauer & Lehner, 2003].

C*sa* é de natureza idêntica ao discutido anteriormente na secção 3.1.1. Assim, lançando mão da eq. 3.2, ter-se-á:

$$Ci(I,M) = \sum_{i=1}^{n} fi_i \min_{c_j \in Anc(c_i)} (|c_j| + Ccom(c_i)) \qquad \text{(eq. 3.6).}$$

Contribuindo para os custos de manutenção de um subcubo (c_i) num qualquer nó, é importante ter em conta os dois custos considerados na eq. 3.6, mas também um outro relativo aos custos de integração no nó destino (*Cint*). Similarmente, para os custos de manutenção, e utilizando a eq. 3.4, ter-se-á:

$$Cm(M) = fm.\sum_{i=1}^{n} \min_{c_j \in Anc(c_i, M')} ((|c_j| + Ccom(c_i)) + Cint(c_i)).em \qquad \text{(eq. 3.7).}$$

Tente-se então aplicar a eq. 3.6 ao cálculo do custo da resposta à interrogação indicada na Figura 3.5. O custo de resposta, utilizando o subcubo (*cpf*) do NSO1, é $Ci(i_1, M) = |(cpf)| = 6M$, uma vez que que os custos de comunicação são nulos. Contudo, se for considerado o subcubo (*c-f*), residente em NSO2, ter-se-á um custo $Ci(i_1, M) = |(c-f)| + Ccom((c-f))_{NSO2 \to NSO1}$. Supondo, para simplificar, que a rede não obriga a um tamanho definido de pacote, ter-se-á $Ccom()_{NSO2 \to NSO1} = \frac{vol\_dados\_a\_tranf}{DB} + La$. Aplicando esta equação em cima e supondo que a rede que interliga os vários nós M-OLAP tem um DB=1Gbps e La=50ms, obter-se-á:

$$Ci(i_1, M) = |(c-f)| + \frac{|(c-f)|}{DB} + La = 5M + \frac{5M.(tam\_tuplo)}{1G} + 50ms\,.$$

Cada tuplo ou célula conterá, em regra, um valor (vendas, custo, quantidade, média, etc.), valor este que será necessário transmitir. Um valor deste tipo ocupa tipicamente 8 bytes, que poderá ser acrescido de mais 2 bytes para considerar alguma sobrecarga protocolar de comunicação. Poderá assim considerar-se que cada tuplo terá um tamanho de 10 bytes. Voltando ao cálculo, teremos: $Ci(i_1, M) = 5M + \frac{5M.10.8}{1G} + 50ms$. Ou seja, tem-se um primeiro termo em unidades tuplos e os segundo e terceiro termos em unidades temporais. Há que encontrar uma forma de converter uma na outra. Voltando ao modelo de custos linear descrito em 3.1.1, na equação $Rt = m*|S| + c$, m representa efectivamente o tempo de execução da interrogação por cada tuplo do subcubo

utilizado para lhe dar resposta. Ou seja, *m* será o custo de processamento de cada tuplo. O inverso será o número de tuplos que o servidor conseguirá processar em cada unidade de tempo, uma grandeza que poderá ser designada como *Cp* (capacidade de processamento). Conhecendo esta grandeza, poder-se-á converter um custo (em tuplos) num custo em unidades temporais, tal como é pretendido. Por outro lado, uma arquitectura M-OLAP tem vários NSOs, podendo cada um ter uma capacidade de processamento (e espaço de armazenamento) diferenciado. Assim, a especificação de *Cp* para cada nó e a sua inclusão na equação de custos permitirá esta generalização do modelo. Retomando o cálculo entretanto interrompido, e supondo uma capacidade de processamento do NSO1 de 10K tuplos.s^{-1} ter-se-á:

$$Ci(i_1, M) = \frac{5M}{10K} + \frac{5M.10.8}{1G} + 50ms = 500 + 0.4 = 500 + 0.4 + 0.05 = 500.45s$$

Generalizando, o custo de resposta a um conjunto de interrogações será calculado por:

$$Ci(I, M) = \sum_{i=1}^{n} fi_i \min_{c_j \in Anc(c_i)} \left(\frac{|c_j|}{Cp} + Ccom(c_i)\right) \qquad \text{(eq. 3.8)},$$

na qual

$$Ccom(c_i)_{NSO_x \to NSO_y} = \frac{|c_i|.10.8}{DB} + La \qquad \text{(eq. 3.9)},$$

se para a rede considerada o tamanho do pacote for variável.
De contrário, para o cálculo dos custos de comunicação, é aplicada a eq. 3.5, em que

$$Np = trunc\left(\frac{|c_i|.10.8}{Sp}\right) + 1 \qquad \text{(eq. 3.10)}.$$

Para finalizar, e considerando que uma interrogação inclui, em regra, cláusulas de restrição, pode estar em questão a busca e agregação de apenas uma parte do subcubo (se disponíveis os índices apropriados). Também os custos de comunicação serão afectados, já que apenas um fragmento de subcubo será transportado. Esta situação pode ser considerada pela inclusão de um novo peso – a extensão de interrogação (*ei*). Desta forma, a eq. 3.8 ficará, finalmente,

$$Ci(I, M) = \sum_{i=1}^{n} fi_i.ei_i \min_{c_j \in Anc(c_i)} \left(\frac{|c_j|}{Cp} + Ccom(c_i)\right) \qquad \text{(eq. 3.11)}.$$

3.2.2 Algoritmos de Estimativa de Custos de Interrogação

Para cálculo de custos de interrogação, poder-se-á, como em 3.1.2, efectuar a aplicação directa da equação de cálculo de custos, neste caso, a eq. 3.11. Mas, já que se está a lidar com uma arquitectura M-OLAP, inerentemente paralela, pode também conceber-se um algoritmo em que seja simulada a execução paralela das diversas tarefas. Assim, serão apresentados dois algoritmos de cálculo de custos de interrogações: o algoritmo ACCIESST (um acrónimo de *A*lgoritmo de *C*álculo de *C*ustos de *I*nterrogação com *E*xecução *S*equencial *S*imulada de *T*arefas), que supõe a execução sequencial das interrogações colocadas para o cálculo de custos de interrogação, abreviado para ASCII, e o algoritmo ACCIEPST (acrónimo para *A*lgoritmo de *C*álculo de *C*ustos de *I*nterrogações com *E*xecução *P*aralela *S*imulada de *T*arefas), ambos vocacionados para utilização em arquitecturas M-OLAP.

No Algoritmo 3.4, apresenta-se o processo de estimativa de custos de interrogação [Loureiro & Belo, 2006b], uma transposição directa da equação 3.11.

```
Input: L // Lattice com todos os subcubos, dependências (MDep) e tamanhos
       Ei=(Ei_{1,1...n},...,Ei_{n,...n}) // Extensão de utilização dos subcubos em cada
         // nó/subcubo, correspondente às cláusulas de restrição das interrogações
       Fi=(Fi_{1...n,1},...., Fi_{1...n,n}) // Frequência de interrogações nos vários nós
       M=(Cu_{1...n,1},...., Cu_{1...n,n}) //subcubos materializados nos nós da arq. M-OLAP
       Cp=(Cp_1... Cp_n.) // Capacidade de Processamento de cada nó
       X=(X_1... X_n.) // Conexões de comunicação e parâmetros respectivos
Output:   C(I, (M,Fi,Ei,P,X)) // custos de resposta às interrogações I, dados Fi,
       // Ei, M, P e X

1. Inicialização: C(I) ← 0;
2. Cálculo e totaliz. dos custos de resposta a cada uma das interrogações:
   Para Cada nó, Fazer:
    Para Cada interrogação i, Fazer:
     Ci ← ∞; // inicializar o custo a um valor muito elevado (um máximo admissível)
     Para Cada antecessor Ca=Ant(i, M), Fazer: // o antecessor pode residir no nó
              // onde foi colocada a interrogação ou outro qualquer nó, dado o lattice
              // estendido (criado pelas dependências inter-nó)
       Ci(Ca, M) ← (scan + aggr. + comm. costs);  // custo de resposta à
         // interrogação I, se o antecessor Ca for utilizado,
         // calculado pelas eq. 3.9, 3.10 e 3.11.
       Se Ci>Ci(Ca, M) Então Ci ← Ci(Ca, M); // cálculo do antecessor mínimo de i
      Próximo Ca
     // Totalizar o custos das interrogações sucessivas
     C(I) ← C(I)+Ci;
    Próximo i
   Próximo nó
3. Devolver Resultado: Return C(I)
```

Algoritmo 3.4. Algoritmo sequencial de cálculo de custos de interrogação (ASCCI).

Neste algoritmo supõe-se que as interrogações vão ser respondidas de uma forma sequencial. Isto significa que o paralelismo inerente da arquitectura M-OLAP não é utilizado.

Embora possa parecer um pouco que este algoritmo é inútil, de facto, os algoritmos de optimização, onde poderá ser utilizado, tentam minimizar os custos de interrogação totais: mesmo que os custos calculados sejam maiores do que os reais, como o objectivo é a sua minimização, o algoritmo poderá ser utilizado, desde que a relação dos custos estimados / custos reais seja mantida (ou com uma aproximação razoável). Assim, embora se saiba, à partida, que os custos aqui calculados deverão ser bastante superiores aos reais, admitindo a condição enunciada, beneficiar-se-á da vantagem de estar a utilizar-se um algoritmo menos complexo, portanto, mais eficiente e, consequentemente, mais escalável.

Um cálculo mais realista impõe a utilização dos vários NSOs disponíveis numa arquitectura M-OLAP capazes de dar resposta, em paralelo, às interrogações colocadas. Dessa forma, concebeu-se também um algoritmo com simulação de execução paralela de tarefas, já que, numa situação real, o sistema tenderá a utilizar todos os recursos disponíveis (Algoritmo 3.5). Basta olhar para esse algoritmo para se perceber (pelo seu tamanho) a sua muito maior complexidade (face ao Algoritmo 3.4). Na verdade, a simulação da execução paralela impõe um controlo da paralelização das tarefas (especialmente a alocação de tarefas a cada um dos NSO e resolução de conflitos). Dada a sua complexidade, optou-se por se discutir alguns conceitos e esquemas utilizados na sua concepção. O Algoritmo 3.5 supõe um lote com as interrogações e tenta alocá-las a NSOs de uma forma paralela, tentando utilizar o paralelismo inerente à arquitectura M-OLAP. Para cálculo dos custos, o algoritmo utiliza o conceito de *janela de execução*, adoptada e descrita em [Loureiro & Belo, 2006g]. A janela de execução é uma versão simplificada do processamento típico de tarefas num *pipeline* [Hennessy & Patterson, 2002]. O conceito de janela de execução será clarificado na secção 3.2.3. Simplificadamente poderá dizer-se que é uma forma de dividir o tempo em intervalos sucessivos, em que as tarefas são executadas de uma forma paralela e cujo valor do intervalo temporal (o custo máximo de um qualquer conjunto de tarefas que formam uma transacção) é depois utilizado para fazer o cálculo dos custos totais de resposta às interrogações.

Como o tempo de execução do algoritmo é uma característica muito importante, dada a sua possível utilização intensiva, heurísticas simples deverão ser preferidas, pois que, decerto, permitirão que a correspondente implementação do algoritmo seja mais rápido. Com este propósito, foram adoptadas um conjunto de opções de concepção:

1. não será tentada a utilização de qualquer função de "*look-ahead* de interrogações", de forma a efectuar uma reordenação das interrogações, sempre que a próxima interrogação tenha de ser processada num NSO já ocupado;
2. se uma situação de ocupação se verificar, só será procurado um antecessor alternativo que possa ser processado num NSO ainda livre na janela de execução corrente;
3. se a estratégia anterior não for bem sucedida, o algoritmo conforma-se simplesmente com a situação (um ou mais NSOs ficarão desocupados na janela corrente), algo semelhante à existência de "bolhas" (*stalls*) [Hennessy & Patterson, 2002], quando se fala em processamento em *pipelines* de microprocessadores.

```
Input: L // Lattice com todas as combinações de granularidades, dependências (MDep) e
tamanhos
       Ei=(Ei1,1...n,...,Ein,...n // Extensão de utilização dos subcubos em cada
          // nó/subcubo, corresp. às cláusulas de restrição em I
       Fi=(Fi1...n,1,...., Fi1...n,n,I)// Distribuição de frequências de interrogação
       M=(Su1...n,1,...., Su1...n,n) // Alocação de subcubos materializados aos NSOs
       P=(P1... Pn.) // Capacidade de processamento de cada NSO
       X=(X1... Xn.) // Conexões de comunicação e seus parâmetros
Output: C(I, (M,Fi,Ei,P,X)) // custos de resposta às interrogações I, dado M, Fi,
    // Ei, P e X
 1. Inicialização: C(I) ← 0, nWindow← 0; // reset dos custos de interrogação e do
       // número da janela de processamento corrente
  1.1. Carga do lote: carrega o lote com todas as interrogações a processar;
 2. Processamento das interrogações:
  Para Cada interrogação i do lote, Fazer:
   2.1. Encontrar um nó e subcubo n_s materializado em M, o antecessor a
    utilizar para responder a i:
    // procura um antecessor que seja o de menor custo ou aquele com o custo mais
    // baixo a seguir (quando está à procura de antecessores alternativos, o
    // antecessor é um par nó e subcubo
    n_s ← próximo antecessor mínimo(i) ∈ M;
    custoProc ← custoBusca/Agr(n_s) // custo de produção de s(i) de n_s no nó n,
     // onde s(i) é o subcubo correspondente a i
    custoCom ← custoCom(s(i)),n → n(i) // custo de transporte de s(i) de n
     // para n(i), o nó onde foi colocada i
    2.1.1. Se não há nenhum antecessor, usar as relações base:
     n ← 0; // o nó 0 é o nó base
     custoProc ← custoBase // custo de processamento base definido
     custoCom ← custoCom(s(i),0 → n(i) // custo de transporte de s(i)
      // desde o nó base até n(i)
   2.2. Aloca o processamento de i ao nó n OU tenta encontar um antecessor
   alternativo OU fecha a janela corrente se não houver qualquer nó
   desocupado capaz de processor I a partir de um qualquer antecessor
   a um custo admissível
```

Algoritmo 3.5. Algoritmo ACCIEPST M-OLAP.

Algoritmo ACCIEPST M-OLAP (continuado da página anterior).

```
Se Encontado um antecessor, mas busyNode[n]=true, Então
    // foi encontrado um antcessor, mas está localizador num nó já ocupado:
    // tentar encontrar um antecessor alternativo
    encontrar_antec_altern = true
    Repetir 2.1.;
Se anc(i)=Ø E encontar_antec_altern=false E busyNode[0]=false, Então
    // aloca o processamento de I ao nó base (nó 0) e adiciona os custos
    // à janela corrente
    busyNode[0] ← true, valUsoProc[0] ←custoProc;
    valUsoCom[0,n(i)] ←valUsoCom[0,n(i)]+custoCom;
    Processa Próxima Interrogação;
 Senão Se anc(i)=Ø E encontar_antec_altern=false E busyNode[0]=true, Então
   Process. A: // a interrogação corrente só pode ser respondida pelas relações base
    // (nó 0), mas este nó já está ocupado: nada mais pode ser feito para resolver o
    // conflito; a janela corrente tem de ser fechada e reprocessar esta interrogação
    corrWinProcCusto ← max(custoProc[qq. nó] // calcula o máximo custo de
     // processamento num qq. nó
    corrWinComCusto ← max(custoCom[qq. nó → qq. nó] // calcula o custo máximo de
     // comunicação de um qq. link
    busyNode[qq. nó] ← false; // todos os nós são libertados (estado=desocupado)
    nWindow++; // incrementa o número da janela de processamento
    // actualizar os custos de interrogação totais
    C(I) ←C(I) + (corrWinProcCusto + corrWinComCusto)*F(i)*E(i) // F(i) e E(i)
     // são a freq. e extensão da interrogação
    Reprocessar a interrogação Corrente i;
   Senão Se encontar_antec_altern=true E nenhum Is Found, Então
    // nada mais pode ser feito, fechar a janela de processamento corrente e
    // reprocessar a interrogação corrente i
    Executar o processamento como Process. A;
   Senão Se encontar_antec_altern=true E encontrou um E custo admissível,
    Então
   Process. B: // encontrou um antecessor a processar num nó desocupado a um custo
    // admissível: processar a interrogação i
    busyNode[n] ← true, valUsoProc[n] ←custoProc;
    valUsoCom[n,n(i)] ←valUsoCom[n,n(i)]+custoCom;
    Processar Próxima Interrogação;
   Senão Se encontar_antec_altern=true E encontrou um E custo n/ admissível,
    Então
    // encontrou um antecessor a processar num nó desocupado mas o seu custo é
    // superior ao admissível
    Executar o processamento como em Process. A; // fechar a janela corrente e
     // reprocessar a interrogação corrente
   Senão Se encontar_antec_altern=false E encontrou um, Então
    Executar processamento como em Process. B; // processar a interrogação
     // corrente e alocar o nó respondente
 2.3. Acumula o custo da última janela de processamento // todas as
    // interrogações foram processadas, acumular custos
   corrWinProcCusto ← max(custoProc[qq. nó]; // calcula o máximo custo de
    // processamento de num qq. nó
   corrWinComCusto ← max(custoCom[qq. nó → qq. nó]; // calcula o custo máximo
    // de comunicação de um qq. link
   C(I) ←C(I) + (corrWinProcCusto + corrWinComCusto)*F(i)*E(i);
3. Devolver o Resultado::
 Return C(I) // devolve o custo estimado das interrogações
```

Possivelmente, esta última heurística configura uma abordagem pessimista de solução para o problema, mas, decerto, mais optimista e próxima da realidade do que a execução sequencial das tarefas suposta no Algoritmo 3.4, onde um só NSO é utilizado em cada janela de execução.

Em resumo, o que atrás se referiu, significa que em cada janela de execução, se uma interrogação for alocada ao NSO *x*, e esse mesmo NSO for o eleito para processar a próxima interrogação, o algoritmo não procura qualquer (futura) interrogação ainda não respondida que possa ser alocada num nó livre. Contudo, para a nova interrogação, tenta encontrar um antecessor alternativo (um subcubo residente num NSO *y* que esteja ainda livre), ainda que implicando um custo superior para a resposta à interrogação.

Acrescente-se que o algoritmo não aceita um qualquer antecessor (senão, no limite, as relações base poderiam ser utilizadas), mas o valor do custo a incorrer deverá estar dentro de um limiar (um parâmetro definidor, um factor aplicado ao valor da janela de execução corrente). Isto significa que o algoritmo aceitará apenas um antecessor alternativo se o seu custo não for muito superior ao custo total de processamento da janela corrente. Desta forma, a utilização de subcubos alternativos implica um custo adicional bastante limitado, já que utiliza tempo já gasto para processar outra(s) interrogação(ões) em nós já alocados, fazendo trabalhar um NSO que, de outra forma, poderia ficar desocupado.

3.2.3 Algoritmos de Estimativa de Custos de Manutenção

Numa arquitectura distribuída OLAP, cada nó é caracterizado por uma capacidade de armazenamento máxima (aqui sem uso, só de interesse em algoritmos de selecção e alocação de cubos, como um constrangimento para a selecção de *M*) e uma capacidade de processamento *Cp*. Como referido anteriormente, qualquer conexão é caracterizada, neste trabalho, por um débito binário e uma latência. O *lattice L* é caracterizado pelos subcubos e suas dependências, tamanho, frequência e extensão de actualização de cada subcubo. Na sua essência, o cálculo dos custos de manutenção de uma distribuição de subcubos *M* (a materializar nos vários NSOs) pode resumir-se como a soma da manutenção de todos os subcubos em *M*. Se o conjunto dos subcubos realmente actualizados está inicialmente vazio e os subcubos forem progressivamente adicionados um a um (criando-se *M'*), como é o caso dos algoritmos de selecção de cubos *greedy*, o cálculo do custo de manutenção é fácil, já que, para cada novo subcubo, conhece-se o *M'* actual. Contudo, esta simplicidade é aparente: já que só é válida para abordagens centralizadas (ver secção 3.1.3).

Numa arquitectura distribuída, a sua aplicação conduz ao cálculo de um custo pessimista, porque é suposta uma execução sequencial (sem usar o paralelismo inerente da arquitectura M-OLAP).

Na prática, uma solução mais realista pode ser a criação de um novo plano de execução, caindo-se numa abordagem mais genérica, na qual *M* é conhecido, sem qualquer ordem pré-estabelecida de geração ou de actualização dos subcubos em *M*. Nesta situação, têm de ser utilizadas heurísticas de forma a construir a ordem de actualização dos subcubos, tendo-se em conta um conjunto de princípios orientadores, nomeadamente:

1. o paralelismo da arquitectura M-OLAP vai ser, sempre que possível, utilizado;
2. no início, as actualizações (ou relações fonte) residem nas relações base, aí estando as sementes do processo de refrescamento;
3. os subcubos ou deltas [Mumick et al., 1997] são calculados no nó base, migrados depois para o(s) nó(s) destino(s);
4. os subcubos ou deltas gerados em 2) devem ser seleccionados criteriosamente, pois que, como são gerados no nó base, não é possível tirar partido do paralelismo da arquitectura;
5. o processo de refrescamento pode prosseguir, utilizando os subcubos actualizados (ou gerados) para actualizar (ou gerar) outros, de uma forma paralela;
6. o processo de refrescamento é dividido em vários passos, sendo o custo total de manutenção a soma dos custos incorridos em todos eles;
7. os custos de manutenção podem ser utilizados em algoritmos de selecção de cubos de duas maneiras: a) para permitir a avaliação da "elegância" (adaptação) de uma distribuição *M* de subcubos (tipicamente adicionada dos custos de interrogação); e b) para permitir a rejeição (ou correcção obrigatória) de uma solução de materialização *M*, se implicar a violação de um constrangimento temporal.

Para limitar a complexidade, o algoritmo não tentará efectuar a transferência para outros nós de qualquer possível processamento de subcubos ou deltas. Isto significa que todas as agregações necessárias serão executadas no próprio nó onde os dados fonte habitarem. Uma abordagem semelhante será adoptada relativamente às comunicações: não será feita qualquer tentativa de procurar vias alternativas, se uma dada conexão estiver congestionada.

Já que várias possíveis heurísticas se perfilaram como base de concepção do algoritmo de estimativa de custos a desenvolver, optou-se por criar três algoritmos, cada um baseado numa forma distinta de propagar as alterações que ocorreram nas relações base. Desta forma, permitiu-

se a execução de uma avaliação comparativa de que se fará uma breve descrição e discussão de resultados (secção 3.2.4).

Também a execução paralela torna o cálculo dos tempo de execução (custos) algo mais complexo, de acordo com o princípio referido no ponto 5) da enumeração anterior. O processo de manutenção é dividido em janelas de execução sucessivas, onde podem decorrer várias tarefas. Para cada janela de execução, o algoritmo usa a máxima utilização de um recurso para a estimativa do custo de manutenção.

A apresentação formal de cada um dos algoritmos vai ser complementada com uma descrição informal que explicará alguns detalhes de concepção e implementação. Todos os algoritmos têm em comum uma concepção baseada em heurística *greedy* e uma execução em duas fases (origem do acrónimo base Greedy2F), sendo a primeira comum ao primeiro e segundo algoritmos, e que se descreve já a seguir.

Fase um: o algoritmo vai procurar todos os subcubos em M (de qualquer dos nós) que só possam ser calculados utilizando as relações base (que habitam no nó 0). Então, esses subcubos (ou deltas) são gerados e enviados para o(s) nó(s) destino. Os custos de execução serão a soma: dos custos de processamento do nó 0 (a produção dos subcubos ou deltas), do máximo do custo de comunicação em qualquer conexão de comunicação e do máximo dos custos de processamento em qualquer dos nós destino (correspondendo à actualização ou integração dos subcubos).

No algoritmo Greedy2FSH [Loureiro & Belo, 2006g], a segunda fase utiliza uma heurística simples para seleccionar a sequência de propagação das alterações; seguindo uma sequência hierárquica (daí o seu acrónimo SH). Esta sequência hierárquica é executada usando subcubos nível a nível, começando no nível de hierarquia mais baixo (maior detalhe) e actualizando um seu descendente mais replicado; o algoritmo procurará encontrar um subcubo a processar em cada nó disponível na arquitectura. O processo é repetido com a geração (ou actualização) de subcubos nível a nível, até que o subcubo de nível mais elevado seja gerado. A versão formal do algoritmo é mostrada em Algoritmo 3.6.

A segunda fase do algoritmo Greedy2FSH pode ser descrita como:

1. O processamento das operações ligadas à manutenção nesta fase está dividido em janelas sucessivas, cujo conceito é descrito já no próximo parágrafo. Em cada janela, o algoritmo usa subcubos já actualizados (ou gerados) – subcubos efectivos – começando com subcubos a um nível mais baixo (maior detalhe), pesquisando todos os nós e

procurando outros subcubos, nível a nível, que não tenham sido ainda actualizados (ou gerados), que os subcubos efectivos possam ser utilizados para actualizar. Se forem encontrados vários subcubos passíveis de ser gerados (ou actualizados) pelo mesmo subcubo efectivo, será eleito o subcubo mais replicado. Os subcubos ou deltas são gerados (sendo o processamento alocado aos nós respectivos) e enviados para os nós destino, sendo considerados os custos de pesquisa, agregação e integração para os nós respectivos e de comunicação para as conexões utilizadas na transferência dos subcubos ou deltas. Ao fechar cada janela, será considerado como o seu custo, a soma do custo máximo num qualquer nó (geração, integração ou ambos) e o custo máximo em qualquer conexão, sendo registadas as tarefas realizadas e custos, que constituem o plano de manutenção.

2. Repetir o passo 1 até que todos os subcubos em *M* sejam actualizados (ou gerados).
3. Devolver Custo(M) e fica também disponível o plano de manutenção.

Para terminar a discussão dos mecanismos e conceitos utilizados neste algoritmo e nos dois seguintes, importa clarificar o conceito de janela de execução. Esta não é mais do que uma divisão temporal, correspondente ao tempo de execução de um conjunto de operações de manutenção (geração de subcubos ou deltas, a sua transmissão e integração nos nós destino). Podemos ver o conceito de janela de execução como um esquema simplificado, uma alternativa à utilização de conceitos ligados à gestão de processamento tipo *pipeline*, sendo o seu uso motivado pela impossibilidade de pensar na capacidade de processamento e comunicação como um reservatório de recursos, sem ter de se lidar com conflitos, quer em termos dos próprios recursos, quer em termos da disponibilidade dos subcubos (ou deltas) já efectivamente actualizados (ou gerados). De facto, não se pode usar um subcubo para actualizar outro antes do primeiro ter sido efectivamente actualizado, impondo a utilização de conceitos ligados à gestão de projectos, dadas as interdependências entre tarefas.

O conceito de janela de execução permite, assim, a exploração do paralelismo inerente à arquitectura M-OLAP, sem obrigar à utilização dos mecanismos complexos ligados ao processamento tipo pipeline. O esquema é provavelmente menos rigoroso, mas é bastante intuitivo, pois que, cada janela corresponde efectivamente ao conjunto de operações correspondentes à carga possível dos vários nós da arquitectura M-OLAP. Além disso, corresponde à própria sequência temporal da propagação das actualizações.

```
Input: L;// Lattice com todas as combinações de granularidade, dependências e tamanhos
       E=(E1,...,En); M=(Cu1...n,1,...., Cu1...n,n); // extensão de actualizações
         // e distribuição de subcubos a materializar em todos os nós
       P=(P1... Pn); X=(X1... Xn); // Capacidade de processamento dos nós,
          //conexões de comunicação e suas características
Output:C (M, E,P,X), P(M, E,P,X); // Custo de manutenção de M, dados E, P e X
        // e respectivo plano de manutenção
Begin
 1. Inicialização
  C(M) ← 0 // inicialização do custo de manutenção de M
  M' ← { }, P(M) ← { } // no início não há qq. subcubo efectivamente materializado
     // (ou actualizado) e plano de manut. vazio
  UgenN ← 0; UintegN ← 0; Ulink ← 0 // inicialização da capacidade de processamento
     // usada na geração, integração e uso de comunicação em links
 2. Processamento da Fase 1:
  P(M) ← "Fase 1, Window 0" // Início da fase 1: na fase 1 materializa ou
     // actualiza os subcubos que não têm qq. antecessor em M
  Para Cada cn ∈ {M, n} ∉ M', fazer: // para cada subcubo ainda não actualizado
     // residente em qq. nó
    Se Não Existir Ant(cn) Em M, fazer: // se não existir um qq. antecessor de cn
     // em M, calcula cn utilizando as relações base
     M' ← M' ⋃+ cn // cn passa a efectivo;
     UgenN(nó base) ← UgenN(nó base) + proc. utiliz. na geração do subcubo
        // ou delta
     UintegN(nón) ← Uinteg(nón) + proc. no nó n // (nó destino onde
        // ocorre a integração)
     Ulink ← Ulink+capacidade de comunicação utilizada da conexão(nó 0 → nó n);
     P(M) ← P(M) + cn  // gerar cn ou delta de cn e exportá-lo para o nó n
  Próximo cn
  C(M) ← UprocN(nó base) + max(UintegN(nón)) + max(Ulink)
 3. Processamento da Fase 2:
  // Fase 2: na segunda fase, em cada nó, o algoritmo procura por subcubos
  // já efectivamente actualizados segundo uma ordem hierárquica directa
  // (de subcubos de granularidade fina, para subcubos de granularidade mais elevada)
  // que possam ser utilizados para actualizar (ou gerar outros)
  Window ← 0; P(M) ← "Fase 2, Window"+Window // início de segunda fase, window=0
  Enquanto M ≠ M' // enquanto houver subcubos ainda não actualizados (ou gerados)
     Para Cada cn ∈ {M', n} numa ordem hierárquica directa // para cada subcubo
      // já actualizado em qq. nó na ordem especific.
      Para Cada n // para cada nó
        Para Cada nível hierárquico h // procura subcubos nível a nível
         Para Cada cnn ∈ {M, n} ∉ M ∈ h // para cada subcubo cnn ainda
          // não actualizado em qualquer nó nn e do nível h
          Se cnn É Descendente(cn) Então// procura descendentes de cada subcubo
           // já actualizado
           csel ← max(num(Descendent(cn))); // selecciona o descendente(cn)
            // mais replicado
           // o subcubo mais replicado é alocado e processado (considerado
           // para custos) e passa a efectivo
           M' ← M' ⋃ csel ; // altera o estado do subcubo csel para actualizado
           UgenN(n) ← UgenN + cap. de proc. usada no nó n (na geração de csel )
            // e totaliza os custos de geração
           UintegN(nn) ← UintegN + cap. de proc. usada no nó nn // totaliza
            // custos de integração (de csel nos nós destino)
```

Algoritmo 3.6. Algoritmo Greedy de Duas Fases com Pesquisa Hierárquica.

Algoritmo Greedy de Duas Fases com Pesquisa Hierárquica (continuado da página anterior)

```
            Ulink ← Ulink + capacidade de comunicação usada do link (se n ≠ nn)
             // totaliza os custos de comunicação
            P(M) ← P(M) + cn + cnn  // gera cn ou correspondente delta no nó n
             // e exporta-o para os nós nn
          Próximo cnn
        Próximo h
      Próximo n
    Próximo cn
    C(M) ← C(M) + max(UgenN(nó n) + UintegN(nónn) + Ulink); // totaliza o custo
     // máximo da janela
    Window ← Window + 1; // incrementa o número de janela
    Para Cada n, nn
      UgenN(n) ← 0;
     UintegN(nn) ← 0; Ulink(n,nn) ← 0; // limpa todos os totalizadores de custos
      // intermédios, quando muda de janela
    Próximo cn, cnn
  Fim Enquanto
 4. Devolver Resultado:
  Return C(M) e P(M); // devolve o custo e plano de manutenção
End
```

Já o algoritmo Greedy2FPR [Loureiro & Belo, 2006b] (Algoritmo 3.7), utiliza uma outra heurística para seleccionar a sequência de propagação das actualizações (ou gerações). Renuncia à pesquisa hierárquica de subcubos já efectivos e separação em níveis dos subcubos (ou deltas) produzidos (como em Greedy2FSH) e, em seu lugar, procura simplesmente por eventuais subcubos replicados, tentando reutilizar um máximo de custos de actualização (daí o seu nome PR – pesquisa de replicação).

O passo 1 da segunda fase deste algoritmo, descrito já a seguir, substitui o correspondente do algoritmo anterior:

> Em cada janela, o algoritmo procura por subcubos (já actualizados), para alocar um a cada nó da arquitectura M-OLAP, que possam ser utilizados para actualizar (ou calcular) outros subcubos ainda não actualizados (ou gerados). Se um subcubo (ou delta) pode ser utilizado para materializar ou actualizar vários outros, o algoritmo vai eleger aquele que é capaz de gerar ou actualizar o subcubo mais replicado, permitindo assim uma reutilização máxima dos custos de geração. Os subcubos ou deltas são gerados e enviados para os nós destino, sendo adicionados os custos de busca e agregação, de comunicação e integração. O custo calculado para a janela de execução será a soma do custo máximo incorrido num qualquer nó e o custo de comunicação máximo ocorrido numa qualquer conexão.

```
Input: L;// Lattice com todas as combinações de granularidade, dependências e tamanhos
       E=(E1,...,En); M=(Cu1...n,1,...., Cu1...n,n); // extensão de actualizações
         // e distribuição de subcubos a materializar em todos os nós
       P=(P1... Pn); X=(X1... Xn); // Capacidade de processamento dos nós,
          //conexões de comunicação e suas características
Output:C (M, E,P,X), P(M, E,P,X); // Custo de manutenção de M, dados E, P e X
        // e respectivo plano de manutenção
Begin
 1. Inicialização
  C(M) ← 0 // inicialização do custo de manutenção de M
  M' ← { }, P(M) ← { } // no início não há qq. subcubo efectivamente materializado
     // (ou actualizado) e plano de manut. vazio
  UgenN ← 0; UintegN ← 0; Ulink ← 0 // inicialização da capacidade de processamento
     // usada na geração, integração e uso de comunicação em links
 2. Processamento da Fase 1:
  P(M) ← "Fase 1, Window 0" // Início da fase 1: na fase 1 materializa ou
     // actualiza os subcubos que não têm qq. antecessor em M
  Para Cada cn ∈ {M, n} ∉ M', fazer: // para cada subcubo ainda não actualizado
     // residente em qq. nó
   Se Não Existir Ant(cn) Em M, fazer: // se não existir um qq. antecessor de cn
     // em M, calcula cn utilizando as relações base
     M' ← M' ⋃+ cn // cn passa a efectivo;
     UgenN(nó base) ← UgenN(nó base) + proc. utiliz. na geração do subcubo
        // ou delta
     UintegN(nón) ← Uinteg(nón) + proc. no nó n // (nó destino onde
        // ocorre a integração)
     Ulink ← Ulink+capacidade de comunicação utilizada da conexão(nó 0 → nó n);
     P(M) ← P(M) + cn  // gerar cn ou delta de cn e exportá-lo para o nó n
  Próximo cn
  C(M) ← UprocN(nó base) + max(UintegN(nón)) + max(Ulink)
 3. Processamento da Fase 2:
  // Fase 2: na segunda fase, em cada nó, o algoritmo procura por subcubos
  // já efectivamente actualizados, seleccionando aquele que vá gerar um outro
  // com o maior número de replicações
  Window ← 0; P(M) ← "Fase 2, Window"+Window // inicializa a segunda fase
    // com window=0
  Enquanto M ⋂ M' ≠ M // enquanto existirem subcubos ainda não actualizados
    //(ou gerados)
    Para Cada n // para cada nó
     Para Cada cn ∈ {M', n} // para cada subcubo já actualizado em cada nó
      Para Cada cnn ∈ {M, n} ∉ M' //  para cada subcubo cnn  ainda não actualizado
        // em qualquer nó nn
       Se cnn é Descendente(cn) Então // procura descendentes para cada subcubo
         // já actualizado
         csel ← max(num(Descendente(cn))); // selecciona o descendente(cn)
           // mais replicado
      Próximo cnn
      // o subcubo mais replicado é alocado e processado (considerado para custos)
      // e passa a efectivo
      M' ← M' ⋃ csel ; // altera o estado do subcubo csel para actualizado
      UgenN(n) ← UgenN + cap. de proc. usada no nó n (na geração de csel ) // e
        //  totaliza os custos de geração
```

Algoritmo 3.7. Algoritmo *Greedy* de Duas Fases com Pesquisa de Replicações.

Algoritmo *Greedy* de Duas Fases com Pesquisa de Replicações (continuado da página anterior)

```
          UintegN(nn) ← UintegN + cap. de proc. usada no nó nn // totaliza custos
            // de integração (de c_sel nos nós destino)
          Ulink ← Ulink + capacidade de comunicação usada do link (se n ≠ nn)
            // totaliza os custos de comunicação
          P(M) ← P(M) + c_n + c_nn  // gera c_n ou corr. delta no nó n e exporta-o para
            // os nós nn
      Próximo c_n
    Próximo n
    C(M) ← C(M)+max(UgenN(nó n)+UintegN(nó_nn)+Ulink) // totalize o custo máximo
      // da janela
    Window ← Window +1; // incrementa o número da janela
    Para Cada n, nn
      UGenN(n) ← 0; UintegN(nn) ← 0; Ulink(n,nn) ← 0; // limpa todos
        // os totalizadores de custos intermédios, quando muda de janela
    Próximo n, nn
  Fim Enquanto
  4. Devolver Resultado:
    Return C(M) e P(M); // devolve o custo e plano de manutenção
End
```

O terceiro algoritmo, denominado Greedy2FPB (Algoritmo 3.8), tem uma primeira fase que é um refinamento da fase 1 do Algoritmo 3.6 ou do Algoritmo 3.7 e uma fase 2 em que é utilizada uma heurística completamente diversa.

A fase 1 tenta tirar vantagem do paralelismo, gerando subcubos ou deltas endereçados a nós inactivos em Greedy2FSH ou Greedy2FPR. Na fase 2, utiliza a métrica de benefício típica dos algoritmos de selecção de cubos (como em "*GSC - Greedy under Space Constraint*" [Harinarayan et al., 1996], daí o nome PB (pesquisa de benefício). Neste algoritmo vai ser utilizado um benefício absoluto, sem qualquer relação com o espaço de materialização, já que este não está em causa. Para cada um dos subcubos ainda não actualizados, o algoritmo vai seleccionar seguidamente aquele cuja utilidade futura para actualizar (ou gerar) outros subcubos seja maior. O tratamento de custos também é algo refinado, já que usa, como unidade atómica de custos, a transacção, compreendida esta como o conjunto correspondente à geração, transmissão e integração de cada subcubo ou delta. Cada janela vai ter um custo imputado igual à transacção de custo máximo.

```
Input: L;// Lattice com todas as combinações de granularidade, dependências e tamanhos
       E=(E1,...,En); M=(Cu1...n,1,...., Cu1...n,n); // extensão de actualizações
         // e distribuição de subcubos a materializar em todos os nós
       P=(P1... Pn); X=(X1... Xn); // Capacidade de processamento dos nós,
          //conexões de comunicação e suas características
Output:C (M, E,P,X), P(M, E,P,X); // Custo de manutenção de M, dados E, P e X
        // e respectivo plano de manutenção
Begin
 1. Inicialização
  C(M) ← 0 // inicialização do custo de manutenção de M e ocupação dos nós
  M' ← { }, P(M) ← { } // no início não há qq. subcubo efectivamente materializado
     // (ou actualizado) e plano de manut. vazio
  UgenN ← 0; UintegN ← 0; Ulink ← 0 // inicialização da capacidade de processamento
     // usada na geração, integração e uso de comunicação em links
 2. Processamento da Fase 1:
  P(M) ← "Fase 1, Window 0" // Início da fase 1: na fase 1 materializa ou
   // actualiza os subcubos que não têm qq. antec. em M
  Para Cada cn ∈ {M, n} ∉ M' // para cada subcubo ainda não actualizado
   // em qualquer nó
   Se Não Existir Anc(cn) Em M // se não houver qq. antecessor de cn em M,
    // calcula cn utilizando as relações base
    M' ← M' ∪ cn // cn passa a efectivo
    UgenN(nó base) ← UgenN(nó base) + proc. utilizado na geração
     // do subcubo ou delta
    UintegN(noden) ← Uinteg(noden) + proc. no nó n // (nó destino onde
     // ocorre a integração)
    Ulink ← Ulink + capacidade de comunic. utilizada da conexão(nó 0 → nó n);
    P(M) ← P(M) + cn // gerar cn ou delta de cn e exportá-lo para o nó n
    Nós_Ocup ← Nós_Ocup ∪ n;
  Próximo cn
  Para Cada n ∈ {Nós} ∉ Nós_Ocup // para cada nó sem subcubo a integrar
   Para Cada cn ∈ {M, n} ∉ M' //  para cada subcubo ainda não actualizado
    csel ← max(Gain(cn,, ∈ {M, n} ∉ M') // o subcubo selec. é aquele com o ganho
     // max. para actualiz. subcubos na fase 2
    M' ← M' + (csel ); // csel muda para já actualizado
    UgenN(nó base) ← UgenN(nó base) +
     proc. gasto na geração do subcubo ou delta;
    UintegN(nó n)← UintegN(nó n) + proc. no nó n; // (onde ocorre a integração)
    Ulink ← Ulink + capacidade de comunic. utilizada da conexão(nó 0 → nó n);
    P(M) ← P(M) + cn; // gerar cn ou delta de cn e exportá-lo para o nó n
   Next cn
  Next n
  C(M) ← UgenN(nó base) + max(UintegN(nón)) + max(Ulink)
```

Algoritmo 3.8. Algoritmo *Greedy* de Duas Fases com Pesquisa de Benefício.

Algoritmo *Greedy* de Duas Fases com Pesquisa de Benefício (continuado da página anterior).

```
  3. Processamento da Fase 2:
   // Fase 2: procura em cada nó subc. já act. que possam ser utiliz. para act.
   // outros, selec. o que permita um > benefício futuro
   Window ← 0; P(M) ← P(M) + "Fase 2, Window "+Window; Nodes_Ocup ← { }// inicia
    // fase 2, window=0; limpa ocup. de nós
   Enquanto M ≠ M' // enquanto houver subcubos ainda não actualizados (ou gerados)
     Para Cada n ∈ {Nós} ∉ Nós_Ocup  // para cada nó ainda não ocupado
      Para Cada c_n ∈ {M', n} //para cada subcubo actualizado no nó n
       Para Cada c_nn ∈ {M} ∉ M' // para cada subcubo ainda não actualizado
        Se c_nn é Descendente(c_n) Então // procura descendentes do subcubo c_n
          c_sel ← max(Gain(c_nn)); // subc. selec. será aquele cujos desc. Assegurem
            // um ganho max. na act. dos seus descendentes
       Próximo c_nn
       // o subcubo mais lucrativo é alocado e processado, sendo considerados
       // os custos respectivos
       M' ← M' ∪ c_sel; // o subcubo mais lucrativo muda o estado para actualizado
       UgenN(nó_n) ← UgenN(nó_n) + proc. utilizado na geração do subcubo
        // ou delta c_sel  no nó n
       UintegN(nó_nn) ← UintegN(nó_nn) + proc. no nó nn // (onde ocorre a
        // integração de c_sel )
       Ulink ← Ulink + capacidade de comunicação utilizada da conexão,
          se n ≠ nn;
       P(M) ← P(M) + c_n;    // gera c_n ou ou delta de c_n e exporta-o para todos os nós
        // nn se houver c_n replicados
       Nós_Ocup ← Nós_Ocup ∪ n; // sinaliza que o nó n passa a ocupado
      Próximo c_n
     Próximo n
     Para Cada n, nn
      C(M) ← C(M)+max(UgenN(node n)+UintegN(node_nn)+Ulink) // update C(M)
      UgenN(n) ← 0; UintegN(nn) ← 0; Ulink(n,nn) ← 0;
     Próximo n, nn
     Window ← Window +1; P(M) ← "Fase 2, Window "+Window;
   Fim Enquanto
  4. Devolver Resultado:
   Return C(M) e P(M); // devolve o custo e plano de manutenção
End
```

Informalmente, este algoritmo pode ser descrito da seguinte maneira:

1. A primeira fase é semelhante à fase 1 do algoritmo Greedy2FPR, mas alterada para tentar tirar partido de um processamento extra no nó base (eventualmente exterior à janela de refrescamento). Então, se numa janela houver nós inactivos (sem subcubos a integrar), o algoritmo vai gerar aqueles que proporcionarão um benefício máximo na fase 2.
2. Na fase 2, para cada nó, o algoritmo procura o subcubo capaz de gerar outros que possam assegurar um benefício máximo na manutenção futura de outros subcubos

ainda não actualizados. O custo de cada janela é calculado como o custo da transacção mais demorada. O custo total de manutenção será a soma do tempo de execução de todas as janelas.

3. Semelhante ao algoritmo Greedy2FPR.
4. Semelhante ao algoritmo Greedy2FPR.

3.2.4 Avaliação Experimental Comparativa dos Algoritmos de Manutenção

Para a avaliação dos algoritmos usou-se o cubo A, descrito anteriormente na secção 2.6. É importante salientar que a dimensão produto tem hierarquias paralelas. Desta forma, os algoritmos devem suportar esta característica. Como se têm três dimensões, cada uma com quatro níveis (2+2 na dimensão produto), perfaz-se um total de 64 subcubos possíveis (Tabela 2.1). Vai supor-se também que o custo de usar as relações base é 18M tuplos (3 vezes o custo do subcubo de granularidade mais fina (cps)).

O desempenho dos algoritmos apresentados, em termos dos custos de manutenção estimados e do tempo de execução respectivo, têm de ser analisados, avaliando o impacto de vários factores, tais como: o espaço de armazenamento por nó; o número de subcubos materializados em cada nó e, indirectamente, o espaço de armazenamento utilizado em cada nó; o número de nós da arquitectura M-OLAP (número de NSOs); e a possível materialização de subcubos no nó 0 (nó base). Nesse sentido, desenhou-se um conjunto de testes que vai ser descrito juntamente com os resultados obtidos. A arquitectura M-OLAP utilizada nos testes iniciais tem 4 nós: o nó 0 contém as relações base e não tem mais espaço disponível para materializar quaisquer subcubos. Os nós 1, 2 e 3 têm espaço disponível, permitindo a sua população com subcubos. A rede de comunicação é completamente conectada e tem um DB de 16 Gbps e uma latência de [15<=La<=40] ms, diferente para cada conexão.

O primeiro teste tem como objectivo a comparação dos custos estimados por cada um dos algoritmos, variando o espaço para materialização de subcubos nos nós 1, 2 e 3 desde 1 a 80% (percentagem relativa ao tamanho de materialização total do cubo). Cinco distribuições aleatórias de subcubos foram geradas nos nós, obedecendo ao constrangimento espacial imposto. Os resultados são mostrados na Figura 3.7, sendo a média do custo de manutenção estimado para as cinco distribuições M. Inspeccionando o gráfico apresentado nessa figura, verifica-se que, para

valores baixos de espaço de materialização (até cerca de 30%), todos os algoritmos exibiram um desempenho semelhante, com uma desvantagem ligeira do *Greedy2FPR*. Todavia, para espaços de materialização acima de 30%, o algoritmo *Greedy2FSH* exibe um melhor desempenho, especialmente visível acima dos 50%.

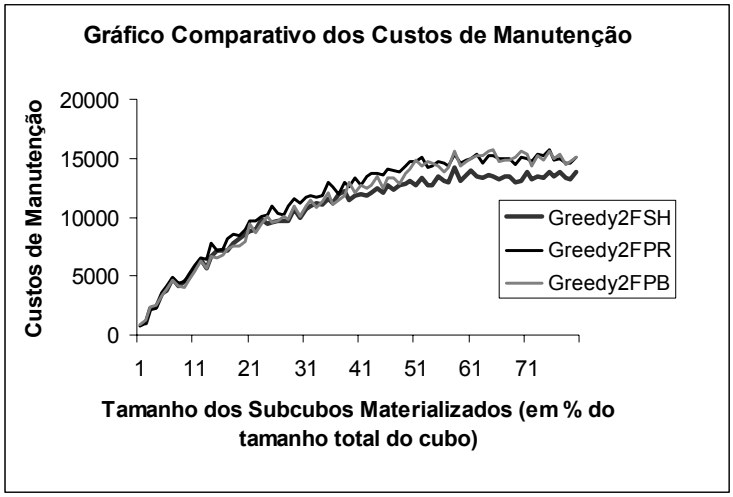


Figura 3.7. Gráfico comparativo do custo de manutenção estimado.

Num segundo teste, vai-se avaliar o impacto do número de subcubos nos custos de manutenção calculados pelos algoritmos. Neste teste, um subcubo é adicionado sucessivamente ao nó que, no momento, tenha um espaço disponível maior, tentando assim manter o espaço ocupado pelos subcubos em cada nó o mais equilibrado possível no decurso do teste. O subcubo a adicionar é seleccionado de uma forma probabilística: são gerados aleatoriamente dois subcubos e, com uma probabilidade (1-p), o maior é seleccionado. Usou-se p=0.8, favorecendo uma selecção de subcubos de tamanho mais reduzido. A cada subcubo adicionado, é calculado o custo de manutenção, utilizando cada um dos algoritmos. O máximo de espaço de materialização nos nós 1, 2 e 3, foi também variado desde 1 a 80%. Foram executados cinco testes. A Figura 3.8 mostra os valores médios dos resultados obtidos. Os resultados observados manifestam imediatamente o comportamento não-monotónico dos custos de manutenção: por vezes, a adição de um novo subcubo causa uma diminuição do custo de manutenção. Já em relação ao desempenho comparativo, observou-se um comportamento semelhante ao do primeiro teste, mas aqui mais visível.

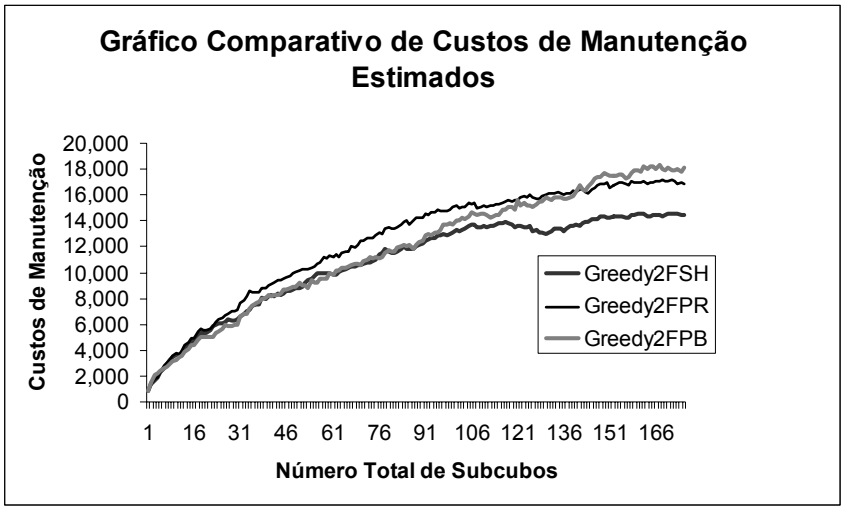


Figura 3.8. Gráfico comparativo do custo de manutenção estimado devolvido pelos algoritmos.

Um terceiro teste vai avaliar o tempo de execução de cada um dos algoritmos, sob as mesmas condições do teste anterior. Os resultados obtidos evidenciam o facto de que, para um número baixo de subcubos em cada nó, todos os algoritmos mostram um desempenho semelhante. Para mais de cinco subcubos em cada nó, os algoritmos comportam-se de uma forma muito diversa. Os algoritmos Greedy2FSH e Greedy2FPR exibem uma evolução do tempo de execução muito favorável: o seu crescimento é baixo (de 4 para 60 ms e de 2 para 20 ms, respectivamente), quando o número de subcubos cresce de 1 para 177. Mas o tempo de execução do algoritmo Greedy2FPB evidencia um crescimento dramático (de 2 para 9934 ms). O detalhe do gráfico da Figura 3.9 (à direita), onde é mostrada a relação tempo de execução x número de subcubos, evidencia a complexidade dos algoritmos: $O(n^1)$ para Greedy2FPH e Greedy2FPR e pelo menos $O(n^2)$ para Greedy2FPB. De facto, a relação tempo de execução / (número de subcubos)^2 mostra um perfil quase linear, revelando uma complexidade n^2 do algoritmo Greedy2FPB, o que constitui um obstáculo claro para sua utilização em algoritmos de selecção de cubos, especialmente em arquitecturas com muitos nós, e para cubos de elevada dimensionalidade.

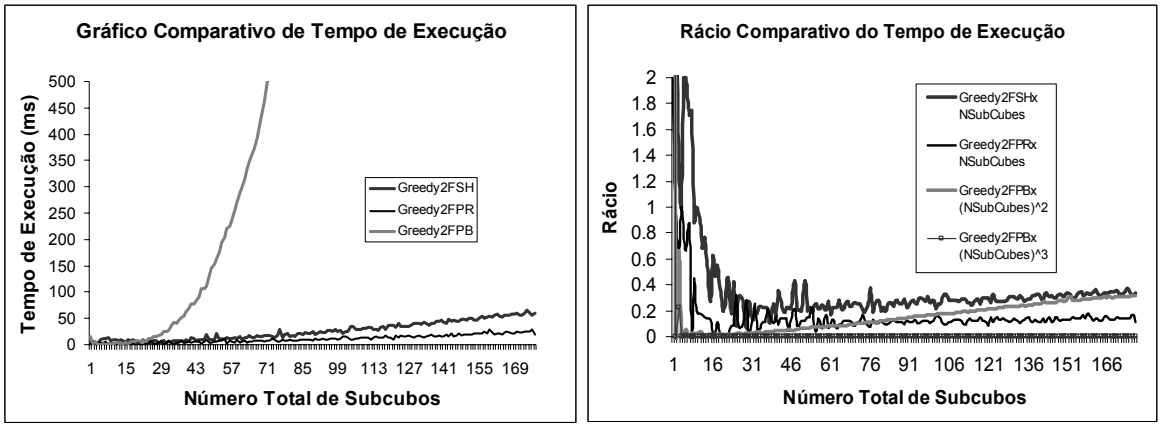


Figura 3.9. Impacto do número de subcubos em M no tempo de execução dos algoritmos.

O quarto teste aspira a avaliar o desempenho relativo dos algoritmos em arquitecturas com um número de nós diferente, ou seja, pretende avaliar-se o impacto do número de nós no desempenho dos algoritmos. Para isso, usaram-se duas arquitecturas: aquela utilizada até agora (nos testes 1 a 3) e outra com 10 nós (de facto, o nó base + 10 outros nós). Os resultados obtidos estão apresentados na Figura 3.10. Todos os demais parâmetros e condições de teste continuaram os mesmos do teste anterior.

Um comportamento semelhante, relativo ao tempo de execução, foi observado: os algoritmos *Greedy2FPH* e *Greedy2FPR* mostraram um crescimento moderado (nem sequer linear com o número de subcubos – no segundo gráfico da Figura 3.10 as linhas respectivas são pouco visíveis, pois que permaneceram praticamente sobre o eixo dos xx); já o algoritmo *Greedy2FPB* mostrou o crescimento exponencial atrás evidenciado.

Os custos de manutenção calculados pelos algoritmos mostraram um comportamento ainda mais inesperado do algoritmo *Greedy2FPB* (dada a sua heurística muito mais elaborada e complexidade). Para a arquitectura com três nós, já se viu atrás que todos os algoritmos têm um desempenho semelhante (com alguma vantagem do *Greedy2FPH*, especialmente visível para um número grande de subcubos). Contudo, para a arquitectura com 10 nós, a supremacia dos *Greedy2FPH* e *Greedy2FPR* é evidente para qualquer número de subcubos.

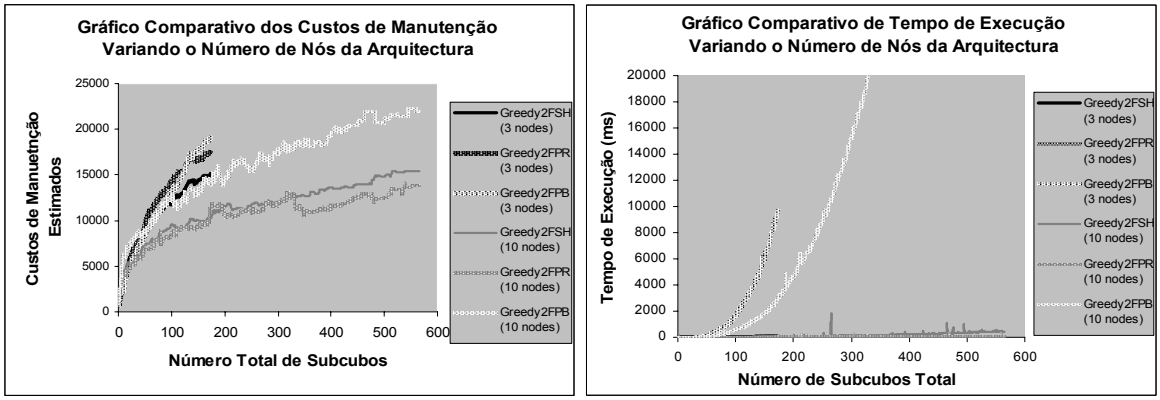


Figura 3.10**.** Desempenho comparativo em termos do custo de manutenção e do tempo de execução.

Este quarto teste pode ser também utilizado para avaliar o *scale-up* e *speed-up* da arquitectura M-OLAP. A Tabela 3.1 mostra um exemplo ilustrativo dos valores observados. Inspeccionando as primeira e segunda linhas da tabela, pode concluir-se que o *speed-up* da arquitectura M-OLAP é pobre (mesmo para o melhor algoritmo, relativamente a esta característica particular) para a manutenção de 150 subcubos, foi observado um decréscimo de 16787 para 9296 – uma razão de 1.8 (claramente menos do que 11/4=2.75 - limite teórico admissível). De facto, a fase 1 da manutenção já foi referida como sendo não paralelizável, e isso prejudica grandemente o *speed-up*. Já no que diz respeito ao *scale-up*, e observando as linha 1 e 3 da Tabela 3.1, pode concluir-se que a arquitectura M-OLAP exibe um comportamento decididamente muito melhor. Esta arquitectura evidenciou um custo de manutenção total, quase igual para os algoritmos *Greedy2FPH* e *Greedy2FPB*, e mesmo um decréscimo, quando *Greedy2FPR* é considerado (provavelmente devido à elevada probabilidade de replicação de subcubos, que é aproveitada pela heurística do algoritmo).

Tabela 3.1**.** Custos de manutenção calculados pelos vários algoritmos.

Número de Nós	**Número de Subcubos**	**Custo Manutenção (Greedy2FSH)**	**Custo Manutenção (Greedy2FPR)**	**Custo Manutenção (Greedy2FPB)**
3	150	14,318	16,667	17,765
10	150	10,025	9,296	12,893
10	495	14,788	12,464	20,744

Um último teste foi ainda efectuado, algo diverso de todos os testes anteriores. Agora, pretende-se avaliar o desempenho dos algoritmos sob condições idênticas às do teste 3, mas permitindo que o

nó base (0) materialize subcubos (uma situação normal quando a arquitectura do DWS tem um DW onde algumas vistas podem ser materializadas), previamente actualizados, e que possam ser igualmente a fonte para refrescamento de subcubos nos outros nós.

É importante referir que os custos de busca e agregação vão ser atribuídos ao nó 0, independentemente de serem o resultado de custos respeitantes às relações base ou às vistas aí materializadas. Para espaço de materialização do nó 0, vão ser usados valores de 0, 10, 20 e 40 % do tamanho total do cubo. As Figura 3.11, Figura 3.12 e Figura 3.13 mostram os resultados obtidos.

É patente um melhoramento no tempo de manutenção, mais evidente para valores baixos de espaço de materialização (10%) no nó 0, se o número de subcubos a refrescar também for baixo. Para um número elevado de subcubos, o desempenho máximo ocorre para espaços de materialização de 20% no nó 0. Um valor mais elevado para o espaço de materialização admissível no nó 0 já não é benéfico. Este comportamento era expectável: se o número de subcubos nos nós 1, 2 e 3 for elevado, espera-se que muitos dos subcubos do nó 0 possam ser utilizados como sementes, havendo nesses casos um grande benefício – de outra forma, as relações base teriam de ser utilizadas. A partir de um certo valor, as vistas materializadas no nó 0 poderão existir também nos outros nós. Como, no teste empreendido, a capacidade de processamento do nó 0 era dupla da dos restantes nós, os subcubos no nó 0 serão, em regra, preferidos, sobrecarregando esse nó e limitando seriamente a eficiência do processamento paralelo. Refira-se, mais uma vez, que o algoritmo *Greedy2FPH* mostrou o melhor desempenho global, entre todos os algoritmos testados.

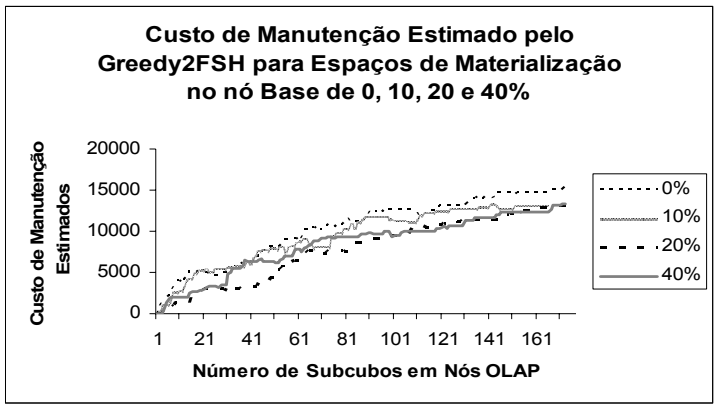


Figura 3.11. Custos de manutenção estimados pelo algoritmo Greedy2FSPH com materialização no nó base.

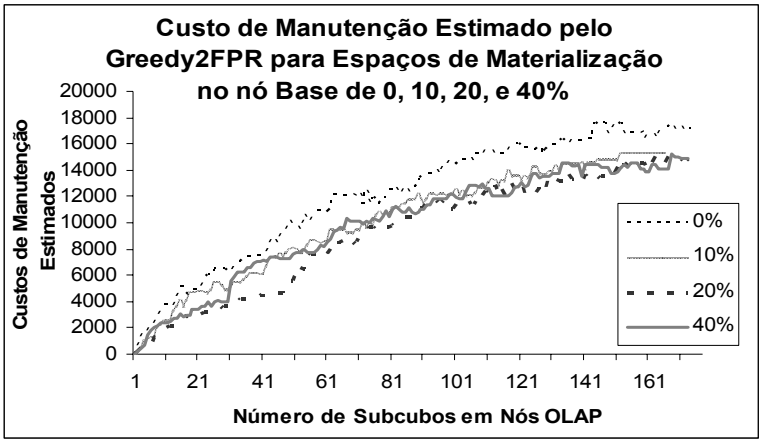


Figura 3.12. – Custos de manutenção estimados pelo algoritmo Greedy2FPR com materialização no nó base.

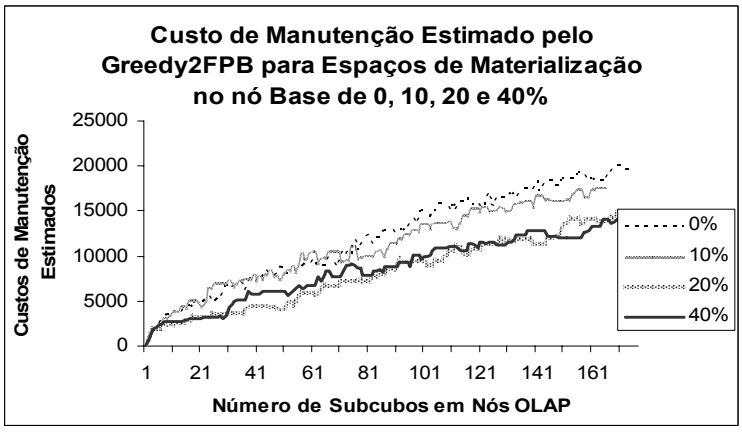


Figura 3.13. Custos de manutenção estimados pelo algoritmo Greedy2FPB com materialização no nó base.

3.2.5 Análise Crítica da Arquitectura, Modelo e Algoritmos

A secção 3.2 introduz a arquitectura M-OLAP distribuída caracterizada por um número variável (potencialmente alto) de nós-servidores OLAP (NSOs) interligados por uma rede de comunicação heterogénea. A ideia base consiste em utilizar muitos servidores pequenos com uma capacidade agregada elevada, que serão, à partida, de custo inferior a um grande servidor de capacidade total idêntica. Isto assegura, à partida, uma escalabilidade quase ilimitada da arquitectura. Outras vantagens vão ser também conseguidas, como a elevada disponibilidade e limitação (eliminação)

de pontos quentes. Para esta nova arquitectura, foi adaptado um modelo de custos proposto em [Bauer & Lehner, 2003], de forma a incluir suporte explícito de custos de comunicação e capacidade de processamento.

No cálculo dos custos de interrogação, está-se a lidar com dois tipos de custos: os custos de busca e agregação (transcritos do modelo linear centralizado) e os custos de comunicação. Os custos integrantes da primeira classe, segundo o modelo linear, têm como unidade de medida o tuplo ou célula, o que na prática mede o esforço da sua leitura e agregação. Já os contribuintes para os custos da segunda classe utilizam o tempo como unidade, pois que mais não são do que a consequência de diversos atrasos impostos pelo atravessar de canais e outras unidades de interligação. Têm-se, assim, duas parcelas de custos que utilizam unidades de medida diferentes, o que faz com que seja necessário uniformizá-las, sob pena de não ser conseguido um cálculo eficaz. Por outro lado, se pensarmos numa arquitectura multi-nó, o "poder" do hardware, que constitui o nó, vai determinar a sua maior ou menor rapidez na resposta a uma mesma interrogação. Ou seja, o tempo de resposta a uma interrogação vai depender não só do subcubo utilizado, mas também do seu hospedeiro. O "poder" do servidor, mais especificamente, a sua capacidade de processamento, permite responder às duas questões em aberto: 1) por um lado, permite transformar o custo (em tuplos ou células) no tempo de execução, uniformizando os dois termos da equação de custos descrita; 2) por outro, permite normalizar o custo numa rede heterogénea de servidores. Com base nestes considerandos, foi desenvolvido e apresentado um novo modelo de custos e respectivas equações de cálculo de custo de interrogação. Quanto aos custos de manutenção, foi aqui introduzido um novo custo: o custo de integração. Este ocorre quando o subcubo ou delta produzido vai ser integrado no nó destino, variando linearmente com o número de tuplos do subcubo ou delta a integrar e considerando a respectiva capacidade de processamento.

A abrangência espacial da arquitectura M-OLAP foi, nos exemplos apresentados, limitada a algumas centenas de metros (uma sala, edifício, ou campus). Todavia, o modelo desenvolvido, especialmente no que concerne ao cálculo dos custos de comunicação, permitiria estender o âmbito espacial, considerando uma WAN que iria assegurar as facilidades de interligação entre os vários NSOs, introduzindo apenas um conjunto de novos parâmetros contribuintes para os custos de comunicação na nova realidade alargada. Esta expansão da arquitectura permitiria acomodar a realidade emergente dos sistemas *peer-to-peer* [Kalnis et al., 2002a], possibilitando a criação de uma arquitectura generalizada, uma expansão arquitectural do Data Warehouse Híbrido Global Distribuído (mostrado na Figura 2.17 da secção 2.9). O modelo de custos estendido poderia

suportar não só a selecção estática das estruturas de optimização (residentes em DW e DM múltiplos), mas também a política de admissão e substituição em servidores OLAP dinâmicos e *peer-to-peer* em clientes.

Em resumo, no modelo aqui introduzido, os aspectos a salientar são: em primeiro lugar, o suporte da distribuição das estruturas OLAP por vários nós, considerando custos de comunicação de forma explícita, facilmente expandidos de forma a acomodar uma abrangência espacial mais alargada; depois, e especialmente, a inclusão da capacidade de processamento de cada NSO como um meio de normalização da possível heterogeneidade dos nós e conversão de unidades de custos, permitindo, no fim, utilizar como referencial de custos o tempo (de nível abstractivo elevado) e, inclusivamente, com reflexão imediata na função objectivo de um qualquer sistema com interacção com utilizadores: a sua satisfação.

Tendo como referencial o modelo mostrado e descrito, foi concebido um conjunto de algoritmos de estimativa de custos de interrogação e manutenção que estendem os trabalhos existentes em quatro vertentes distintas:

1. são vocacionados à arquitectura M-OLAP (distribuída);
2. introduzem a paralelização das tarefas do processo de manutenção;
3. geram um plano de manutenção capaz de ajudar no processo de ETL (*extract, transform and load*) ou no processo de restruturação;
4. incluem suporte explícito para os custos de comunicação e capacidade de processamento no modelo de custos.

Os testes experimentais levados a cabo mostraram um comportamento algo inesperado. Ao conceber os algoritmos, as heurísticas mais elaboradas empregues no algoritmo *Greedy2FPB* elevou as expectativas acerca do plano de manutenção que ele iria gerar e dos correspondentes custos de manutenção estimados, algo que seria pago pelo muito maior tempo de execução que implicaria. Se a última expectativa se verificou, já a primeira foi parcialmente negada, uma vez que o algoritmo *Greedy2FPH* revelou um desempenho igual ou melhor.

Em geral, pode concluir-se que os algoritmos *Greedy2FPHS* e *Greedy2FPR* têm uma grande aplicabilidade em algoritmos de selecção de cubos, porquanto evidenciaram uma complexidade baixa (o tempo de execução não depende do número de subcubos materializados e do número de nós da arquitectura) e planos de manutenção algo semelhantes. A complexidade $O(n^2)$ do *Greedy2FPB* (onde *n* representa o número de subcubos em *M*) torna-o inaplicável em algoritmos altamente iterativos, mesmo quando cubos OLAP de baixa dimensionalidade estejam a ser

considerados. Mesmo para o caso simples do *lattice* com 64 subcubos, quando aplicado à arquitectura de 10 nós, o tempo de execução foi quase inaceitável.

Geralmente, poderá dizer-se que *Greedy2FPH* é o algoritmo que mostra o melhor comportamento global, seguido, a curta distância, pelo *Greedy2FPR*. Ambos proporcionam um bom plano de manutenção, evidenciando um tempo de execução curto, podendo então ser utilizados em algoritmos de selecção de cubos em arquitecturas multi-nó que previsivelmente farão um uso intensivo da estimativa de custos de manutenção.

A concepção e implementação das alterações necessárias para acomodar os requisitos implicados pelo último teste (a pré-existência de vistas materializadas no nó base) e os resultados obtidos nesse teste mostraram as limitações do modelo e heurísticas utilizadas. Uma nova visão do processo de manutenção foi induzida: a possibilidade da coexistência de nós ou subcubos passíveis de manutenção integral (onde os subcubos são gerados integralmente – "*from scratch*"), coexistindo com outros a ser refrescados de forma incremental. Além disso, o suporte ao cálculo dos custos de manutenção na sua vertente incremental, aquela de interesse crescente, dado ser a única admissível em larga escala (devido aos constrangimentos temporais da janela de refrescamento) é bastante limitado: o parâmetro extensão de actualização, especificado por subcubo, se é conveniente para o cálculo dos custos de integração e comunicação (na prática corresponde ao tamanho do delta), não o será para o cálculo dos custos de busca e agregação, já que a suposta existência de índices ou ordenações adequados é variável, conforme o antecessor usado, e, assim, estes custos serão bastante diversos. Na prática, será necessário incluir não linearidades com impacto nos custos de manutenção, implicando a modificação do modelo de custos. Por outro lado, a operação de busca e agregação é comum ao processo de manutenção e de resposta a interrogações. É forçoso concluir que estas não linearidades poderão ser igualmente incluídas no cálculo dos custos de interrogação, permitindo assim ao modelo ser capaz de suportar uma muito maior abrangência e consequente alargamento da sua aplicabilidade. A próxima secção apresenta este novo modelo, explicando a sua génese e concepção, além das equações de custo e respectivos algoritmos de estimativa de custos.

3.3 Modelo não-Linear Distribuído

Como o nome indica, este novo modelo introduz não linearidades, permitindo assim acomodar as especificidades de cada subcubo e NSO. Neste caso é abandonada a dependência linear dos custos

em relação ao tamanho do subcubo utilizado, introduzindo pesos adicionais que permitem uma diferenciação. Como se disse no final da secção anterior, o modelo linear distribuído permite o cálculo de custos de interrogação e manutenção, mas, para estes últimos, apresenta limitações. Se a manutenção integral é suportada (já que, na prática, mais não é do que a leitura e a agregação do subcubo mais favorável já materializado num qualquer nó ou mesmo na relação base), gerando o subcubo pretendido e sua posterior integração no nó destino, a manutenção incremental é conseguida pela inclusão de uma extensão de actualização que, mesmo sendo especificada por nó e subcubo, se torna algo limitativa. Para responder a esta limitação, bastará adicionar ao *lattice* de dependências um novo peso w_m, que chamaremos peso de custo de actualização, a ser colocado em cada seta entre s_i e s_j, correspondendo ao custo de actualização de s_i a partir de s_j.

Outra situação comum é o cálculo dos custos relativos a um subcubo poder considerar outros custos, que não só aqueles relacionados com a busca e agregação de um subcubo antecessor. Por exemplo, a existência de índices ou ordenações pode beneficiar o cálculo de um subcubo relativamente a outro, ainda que o mesmo antecessor seja utilizado, introduzindo não linearidades nos custos de busca e agregação. Esta diferenciação pode ser mapeada no modelo com a introdução de um novo peso de custo, colocado em cada seta correspondente às dependências entre subcubos no *lattice*, funcionando como um valor aditivo aos custos de leitura base.

3.3.1 Modelo não-Linear Distribuído e Equações de Cálculo

Adicionando os novos pesos ao modelo anterior, teremos o que pode ser denominado como *lattice* de dependências estendido, também sob a forma de um grafo directo acíclico (Figura 3.14), no detalhe relativo ao NSO1.

Em cada vértice do grafo estará localizado um subcubo com os seguintes parâmetros e pesos associados:

- *T*: tamanho do subcubo, um valor para os custos de busca;

- *fi* e *fm*: frequências de interrogação e actualização;

- *ei* e *em*: extensão de utilização em interrogações e extensão de actualização.

Cada uma das setas tem associados dois pesos, nomeadamente:

- wi_{ij}: custos de processamento do subcubo s_i gerado a partir de s_j;

- wm_{ij}: custo de processamento de actualização do subcubo s_i a partir de s_j.

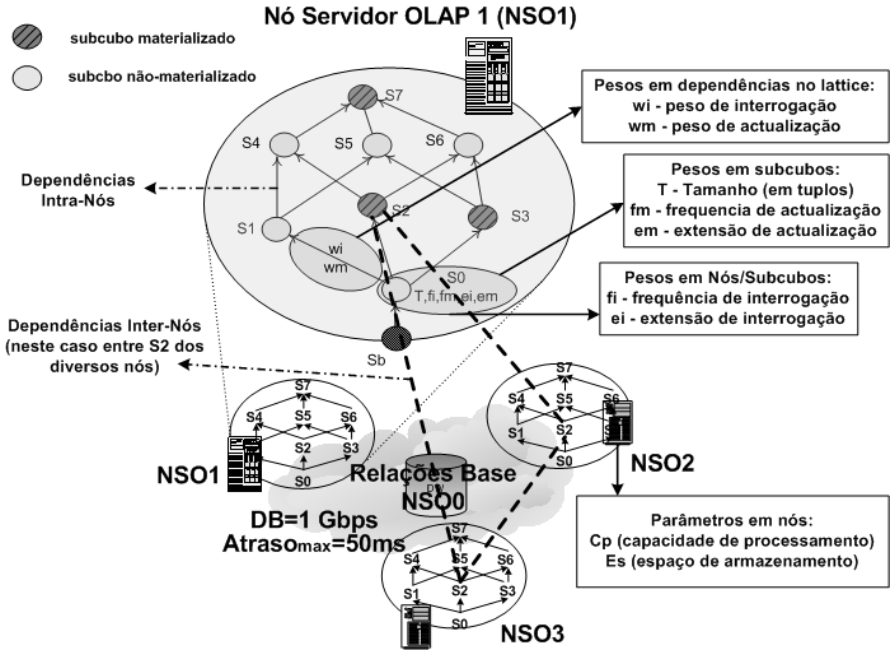


Figura 3.14. Arquitectura OLAP multi-nó e modelo de custos generalizado.

É importante salientar que wm_{ij} representa o custo de preparação de um delta [Mumick et al., 1997] utilizando *j*. Os custos de integração do delta ou subcubo vão ser considerados utilizando *em*, que corresponde ao impacto das alterações nas relações base em cada subcubo (a razão entre o número de células actualizadas no subcubo refrescado e o tamanho respectivo). Esta divisão em duas parcelas para o cálculo do custo de manutenção é necessária. No processo de paralelização, a geração dos deltas e a sua integração pode ocorrer em nós distintos e, dessa forma, ser paralelizada, o que seria impossível se os custos correspondentes não pudessem ser diferenciados.

Apresentado o modelo, importa agora deduzir as respectivas equações de cálculo, que permitam estimar os custos de interrogação e manutenção. Este novo modelo tem implicações profundas na forma de calcular os custos respectivos. As não linearidades surgem como pesos e novos termos contribuintes para os custos. Estas vão permitir distinguir as contribuições para os diversos tipos de custos a um nível de maior detalhe. Assim, e genericamente, para os custos de interrogação contribuem: os custos de busca (proporcionais ao tamanho do antecessor utilizado), de agregação

(a considerar utilizando wi_{ij}, segundo o princípio do caminho mais curto) e de comunicação. O segundo custo apenas será aplicável se o antecessor utilizado for diferente do subcubo correspondente à interrogação, e o terceiro, sempre que o NSO onde reside o antecessor utilizado seja diferente do NSO onde foi colocada a interrogação. Em resumo, $Ci(i,M) = Cb + Ca + Cc$. Assim, a equação de cálculo de custos terá de incluir agora uma parcela adicional, correspondente ao cálculo dos custos de agregação *Ca*, calculados por

$$Ca = \min \sum_{v \in Cnhs(Anc(i_i) \to i_i)} wi_v \qquad \text{(eq. 3.12)},$$

em que *v* será um qualquer caminho (*Cnh*) entre i_i e o antecessor considerado, sendo depois seleccionado aquele que implicar o custo mínimo (utilizando um algoritmo de cálculo do caminho mais curto, por exemplo Dijkstra [Cormen et al., 2001], A* [Hart et al., 1968], Bellman-Ford [Bellman, R., 1958],[Cormen et al., 2001] ou Floyd-Warshall [Floyd, 1962],[Cormen et al., 2001]). Incluindo a eq. 3.11 na eq. 3.8 ter-se-á, finalmente, para os custos de interrogação,

$$Ci(I,M) = \sum_{i_i \in I} fi_{i_i} . (\min((|\, Anc(i_i, M)\,| + \min \sum_{v \in cnhs(Anc(i_i) \to i_i)} wi_v) / Cp(Node_{Anc_{i_i}}) + CC_{No(Anc_{i_i} \to i_i)}) \qquad \text{(eq. 3.13)},$$

na qual os custos de comunicação podem ser calculados através das equações 3.14 e 3.15, alteradas a partir das equações 3.9 ou 3.10, pois que se está agora a refinar o cálculo, considerando-se que a extensão de interrogação (ei_{i_i}) vai ter impacto apenas no tamanho do fragmento correspondente à resposta à interrogação e, consequentemente, no volume de dados a transferir, visto que os custos de latência serão tipicamente idênticos e independentes do volume de dados transferidos. Assim, se para a rede considerada, o tamanho do pacote for variável, teremos:

$$Ccom(c_i)_{NSO_x \to NSO_y} = \frac{|\, c_i \,| . ei_{i_i} . 10.8}{DB} + La \qquad \text{(eq. 3.14)}.$$

De contrário, para o cálculo dos custos de comunicação aplica-se a eq. 3.5, sendo

$$Np = trunc(\frac{|\, c_i \,| . ei_{i_i} . 10.8}{Sp}) + 1 \qquad \text{(eq. 3.15)}.$$

Discutindo, agora, os custos de manutenção, sua diferenciação por tipo de manutenção considerada, os seus contribuintes e variabilidade de comportamento, há a referir o seguinte:

1. os custos de manutenção integral são, em regra, bastante maiores do que os custos de manutenção incremental;
2. o modelo em desenvolvimento, já se disse, deve dar suporte a ambos os tipos de manutenção e de uma forma mais precisa, já que a principal limitação do modelo anterior era o suporte para custos de manutenção incremental;
3. atendendo a 1. e 2., compreende-se que o comportamento de um e outro tipo de manutenções, sendo diverso, implicará a diferenciação das respectivas equações de cálculo;
4. para custos de manutenção integral, e uma vez que os factores contribuintes para os custos de geração de um subcubo são similares aos da produção do subcubo (e interrogação correspondente), a equação de cálculo deverá ser similar à eq. 3.13, com as devidas alterações, para considerar os pesos correspondentes à manutenção e os custos de integração;
5. para os custos de manutenção incremental, como os custos de busca poderão ser muito mais pequenos do que em 4. (um delta pode ser utilizado ou podem ser exteriores), vai considerar-se que o peso wm_{ij} inclui ambos, havendo que calcular o caminho de dependências que assegure um custo mínimo, analogamente à eq. 3.12.

Atendendo aos considerandos e princípios enunciados acima, ter-se-á, para os custos de manutenção integral:

$$Cm(M) = fm. \sum_{S_i \in M} (\min((|\, Anc(S_i, M)\,| + \min \sum_{v \in cnhs(Anc(s_i) \to s_i)} wi_v) / Cp(No_{Anc(s_i)}) + CC_{No(Anc_{Si} \to S_i)}) + S_i.em_{s_i} / Cp(No_{s_i})) \qquad \text{(eq. 3.16).}$$

Já relativamente aos custos de manutenção incremental,

$$Cm(M) = fm. \sum_{S_i \in M} \min(\sum_{v \in cnhs(Anc(s_i) \to s_i)} wm_v / Cp(No_{Anc(s_i)}) + CC_{No(Anc_{s_i} \to s_i)}) + S_i.em_{s_i} / Cp(No_{s_i}) \qquad \text{(eq. 3.17).}$$

Para o cálculo dos custos de comunicação aplicam-se, identicamente, as eq. 3.14 e 3.15.

Uma mostra do cálculo do custo de interrogação e manutenção será, porventura, elucidativa.

Suponha-se, na Figura 3.15, que se pretende calcular os custos de resposta à interrogação i6 (s6) colocada no NSO1. Olhando a figura, e atendendo às dependências intra e inter-nó, pode ver-se que no mesmo NSO poderão ser usados S2 e S3, ou usar S6 no nó 2, e, naturalmente, também Sb (subcubo virtual correspondente às relações base).

Então, aplicando a eq. 3.13, e atendendo aos valores das tabelas da Figura 3.15, ter-se-á, respectivamente:

Custo de resposta se S2 de NSO1 for utilizado,

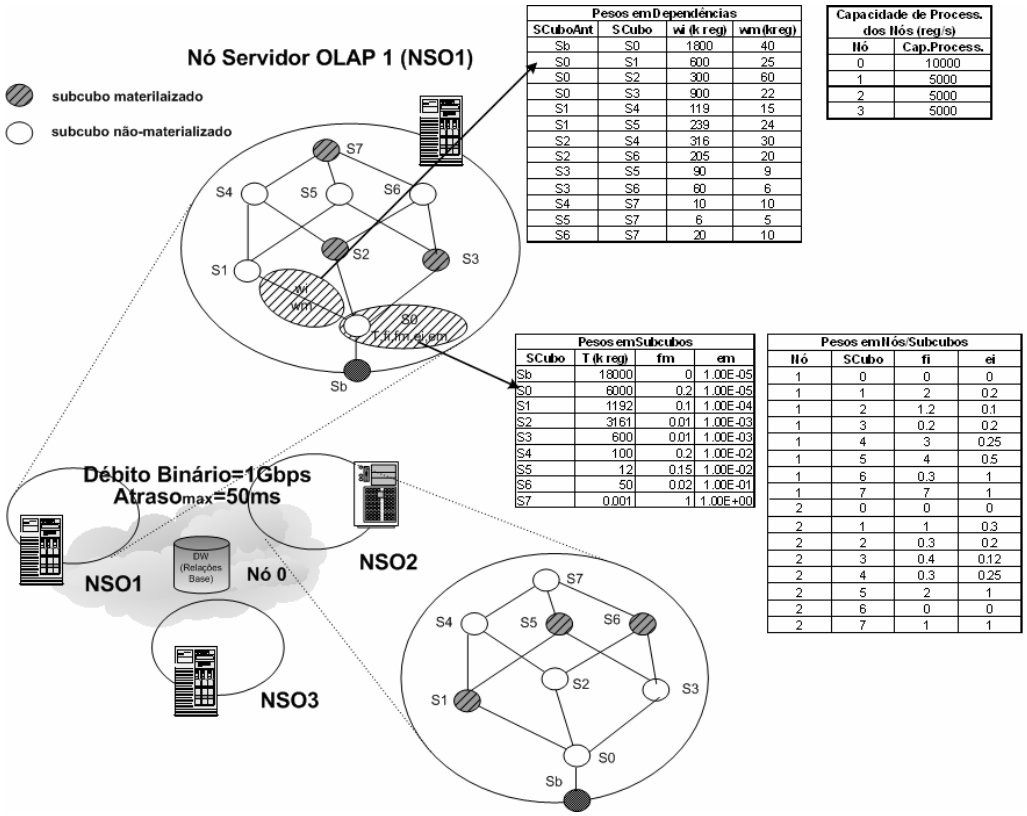
Ci(s6)=0.3*((3161E3+205E3)/5000+0)=2019.96 s;

usando S3 do NSO1: Ci(s6)=0.3*((600E3+60E3)/5000+0)=39.6 s, e

S6 NSO2: Ci(s6)=0.3*((50E3+0)/5000+(50E3*10*8)*1/1E9+0.05)=3.05 s.

Em conclusão, S6 de NSO2 deverá ser seleccionado para responder à interrogação, já que é aquele que assegura um custo mínimo.

Para o cálculo do custo de manutenção vai considerar-se também a distribuição de subcubos materializados *M* mostrada na Figura 3.15, supondo que os subcubos em NSO1 deverão ser objecto de manutenção integral, que o NSO2 deve ser refrescado incrementalmente e, ainda, que o NSO3 não tem qualquer espaço disponível para materialização.



Pesos em Dependências			
SCuboAnt	SCubo	wi (k reg)	wm(kreg)
Sb	S0	1800	40
S0	S1	600	25
S0	S2	300	60
S0	S3	900	22
S1	S4	119	15
S1	S5	239	24
S2	S4	316	30
S2	S6	205	20
S3	S5	90	9
S3	S6	60	6
S4	S7	10	10
S5	S7	6	5
S6	S7	20	10

Capacidade de Process. dos Nós (reg/s)	
Nó	Cap.Process.
0	10000
1	5000
2	5000
3	5000

Pesos em Subcubos			
SCubo	T (k reg)	fm	em
Sb	18000	0	1.00E-05
S0	6000	0.2	1.00E-05
S1	1192	0.1	1.00E-04
S2	3161	0.01	1.00E-03
S3	600	0.01	1.00E-03
S4	100	0.2	1.00E-02
S5	12	0.15	1.00E-02
S6	50	0.02	1.00E-01
S7	0.001	1	1.00E+00

Pesos em Nós/Subcubos			
Nó	SCubo	fi	ei
1	0	0	0
1	1	2	0.2
1	2	1.2	0.1
1	3	0.2	0.2
1	4	3	0.25
1	5	4	0.5
1	6	0.3	1
1	7	7	1
2	0	0	0
2	1	1	0.3
2	2	0.3	0.2
2	3	0.4	0.12
2	4	0.3	0.25
2	5	2	1
2	6	0	0
2	7	1	1

Figura 3.15. Modelo de custos generalizado para o cálculo de custos.

Uma vez que os subcubos actualizados poderão ser utilizados para gerar outros subcubos, em primeiro lugar, deve ser empreendida a manutenção incremental, já que é a mais "barata", e os custos menores, assim incorridos, podem, depois, ser benéficos na geração de outros subcubos. Então, iniciando no nó 2, e adoptando uma estratégia de actualização hierárquica que provou em 3.2.5 obter bons resultados e desempenho, ter-se-á:

Cm(M, nó 2)= Cm(S1)+Cm(S5)+Cm(S6).

Aplicando a eq. 3.17 para calcular os custos de actualização de S1:

Cm(S1)=0.1*(40E3+25E3)/10000+1192E3*0.0001*10*8/1E9+1192E3*0.0001/5000)=0.657 s.

A mesma equação pode ser aplicada para S5 e S6. A Tabela 3.2 mostra os resultados do valor do custo de manutenção relativos ao NSO2.

Tabela 3.2. Custos de manutenção incremental relativos aos subcubos materializados em NSO2 (Figura 3.15).

Δ Gerado	Nó Antec	SCubo Ant	|Scubo Ant|	Nó Integ	min∑wm	Cp(nóAnt)	Cp(NóInteg)	|Si|	em	Atr	fm	Cm(Si,M)
S1	Base	Rel. Base	1.800E+07	2	6.5E+04	10000	5000	1.192E+06	0.0001	0.05	0.1	6.574E-01
S5	2	S1	1.192E+06	2	2.4E+04	5000	5000	1.200E+04	0.01	0.05	0.15	7.236E-01
S6	Base	Rel. Base	1.800E+07	2	6.8E+05	10000	5000	5.000E+04	0.1	0.05	0.02	1.381E+00
								Custo de Manutenção Total				2.761993

Similarmente, para a manutenção integral dos subcubos em NSO1, aplicando a eq. 3.16, obter-se-ão os valores da Tabela 3.3, onde os custos de manutenção mínimos de cada um dos subcubos a materializar aparecem a sombreado.

Tabela 3.3. Custos de manutenção integral relativos ao NSO1 da Figura 3.15.

SCubo Ger.	Nó Antec	SCubo Ant	|Scubo Ant|	Nó Integ	min∑wi	Cp(nóAnt)	Cp(NóInteg)	|Si|	em	Atr	fm	Cm(Si,M)
S2	Base	Rel. Base	1.80E+07	1	2.10E+06	10000	5000	3.16E+06	1	0.05	0.01	26.425029
S3	Base	Rel. Base	1.80E+07	1	2.70E+06	10000	5000	6.00E+05	1	0.05	0.01	21.90098
S7	1	S2	3.16E+06	1	2.25E+05	5000	5000	1	1	0.05	1	677.2002
S7	1	S3	6.00E+05	1	8.00E+04	5000	5000	1	1	0.05	1	136.0002
S7	2	S5	1.20E+04	1	6.00E+03	5000	5000	1	1	0.05	1	3.6502001
S7	2	S6	5.00E+04	1	2.00E+04	5000	5000	1	1	0.05	1	14.0502
								Custo de Manutenção Total				51.976209

Para terminar, duas pequenas observações:

1. Quando as relações base (nó base) são requisitadas, assume-se que o processamento é realizado localmente, sendo contudo utilizados os *wis* e *wms* dos outros nós, já que é

legítimo supor que as eventuais ordenações ou índices possam ser recreadas também no nó base.

2. A observação anterior levantou uma questão mais global: no modelo são especificados parâmetros e pesos iguais para todos os nós, limitando assim a diversidade e eventuais diferenças de estruturas auxiliares existentes em cada nó. Tratou-se apenas de uma simplificação, mas poder-se-iam especificar *wis* e *wms* diferentes em cada nó, permanecendo, contudo, válidas as equações de cálculo deduzidas. Neste caso, haveria apenas que considerar, para estimar os custos relativos a cada interrogação ou manutenção de subcubo, os pesos e parâmetros relativos a cada nó em consideração.

3.3.2 Algoritmos de Estimativa de Custos de Interrogação

Para o cálculo dos custos de interrogação, considerando um modelo de custos não linear distribuído vão apresentar-se também dois algoritmos, à semelhança do que foi feito na secção 3.2.2. No primeiro algoritmo supõe-se a execução simulada sequencial de tarefas (ACCIESST). O segundo algoritmo, vai já incluir a execução paralela simulada das tarefas. As considerações então tecidas à volta da conveniência desta dupla sugestão, são aqui, de igual forma, válidas. Embora separados, os algoritmos podem ser implementados como serviços da mesma classe, tendo a vantagem da reutilização de estruturas de dados e outros serviços complementares.

O primeiro algoritmo (Algoritmo 3.9) é em tudo idêntico ao correspondente baseado no modelo linear, apresentando-se aqui também por uma questão de sistematização e já que a implementação considera a forma diferente de cálculo dos custos. O acrónimo ACCIESST é retirado de Algoritmo de Cálculo de Custos de Interrogação com Execução Sequencial Simulada de Tarefas [Loureiro & Belo, 2006h]. É um algoritmo simples: uma transcrição directa da eq. 3.13. É perfeitamente idêntico ao algoritmo ASCCI, diferindo apenas nas equações de cálculo utilizadas, agora as eq. 3.13 (com a variação de considerar apenas cada uma das interrogações), 3.14 e 3.15. Utiliza, tal como todos os outros, o serviço de geração e disponibilização das relações de dependência, cujas vantagens foram já discutidas anteriormente.

```
Input: L // Lattice com todos os subcubos, dependências (MDep) e tamanhos
       Ei=(Ei1,1...n,...,Ein,...n) // Extensão de utilização dos subcubos em cada
          // nó/subcubo, correspondente às cláusulas de restrição das interrogações
       Fi=( Fi1...n,1,...., Fi1...n,n) // Distribuição da frequência de interrogações
          // nos vários nós
      M=(Cu1...n,1,...., Cu1...n,n) // subcubos materializ. nos vários nós da arq. M-OLAP
      P=(P1... Pn.) // Capacidade de Processamento de cada nó
      X=(X1... Xn.) // Conexões de comunicação e parâmetros respectivos
Output: C(I, (M,Fi,Ei,P,X)) // custos de resposta às interrogações I,
          // dado Fi, Ei, M, P e X

1. Inicialização: C(I) ← 0;
2. Cálculo e totalização do custos de resposta a cada uma das interrogações:
   Para Cada nó, Fazer:
    Para Cada interrogação i, Fazer:
     Ci ← ∞; // inicializar o custo a um valor muito elevado (um máximo admissível)
     Para Cada antecessor Ca=Anc(i, M), Fazer: // o antecessor pode residir no nó
      // onde foi colocada a interrogação ou outro qualquer nó, dado o lattice
      // estendido (criado pelas dependências inter-nó)
      Ci(Ca, M) ← (busca + aggr. + custos de comun.);  // custo de resposta à
       // interrogação I, se o antecessor Ca for utilizado, calculado pelas eq. 3.13
       // (para cada interrogação), 3.14 e 3.15.
       Se Ci>Ci(Ca, M) Então Ci ← Ci(Ca, M); // cálculo do antecessor mínimo de i
       Próximo Ca
       // Totalizar o custos das interrogações sucessivas
     C(I) ← C(I)+Ci;
    Próximo i
   Próximo nó
3. Devolver o resultado:
   Return C(I)
```

.

Algoritmo 3.9. Algoritmo de cálculo de custos de interrogação com execução sequencial simulada de tarefas (ACCIESST).

Já o segundo algoritmo é bastante diferente. Se o correspondente baseado no modelo linear (Algoritmo 3.5) utilizava o conceito de "janela de execução" como um esquema de alocação de tarefas a nós, sendo uma simplificação da forma de paralelização com *pipelines*, como então foi dito, aqui, o esquema de controlo de processamento paralelo vai mesmo utilizar conceitos de alocação de tarefas e controlo de utilização de recursos típicos do esquema *pipelining*. Esquemas e conceitos como *scoreboarding,* "bolhas", *multi-pipelining*, *slots* e conflitos de dados ou recursos, vão ser de utilização comum. Dadas as características mais relevantes serem o paralelismo, a inclusão de pipelines e a reordenação de tarefas, o algoritmo é apropriadamente denominado de *Algoritmo de Execução Paralela de Interrogações em Pipeline com Alocação de Nós por Reordenação Constrangida* (APIRC) [Loureiro & Belo, 2006h]. É formalmente apresentado em Algoritmo 3.10 e, dada a novidade (neste contexto) de alguns conceitos, importa incluir aqui uma

descrição aturada da sua orgânica. Antes, porém, discutir-se-ão alguns assuntos relacionados com o processamento paralelo, e só então será utilizado um exemplo para melhor explicar a operação do algoritmo.

Num sistema OLAP real, o processamento ocorre em tempo real e cada interrogação acontece num instante temporal, considerado como $t = 0$, de forma a calcular o respectivo tempo de resposta. Se o modelo tivesse de lidar com a execução em tempo real das interrogações, seria muito mais complexo. Todos os valores dos somatórios e médias no modelo de custos proposto, relacionados com caracterização das interrogações, seriam completamente inúteis. Mesmo o algoritmo (as suas estruturas de controlo, a gestão do lançamento das interrogações nos pipelines e o estado dos objectos) seria mais complexo, tornando também o tempo de execução muito maior. Por outro lado, o objectivo aqui não é o controlo em tempo real do sistema OLAP, mas apenas a estimativa dos custos de interrogação. Não se pretende realmente saber quando uma interrogação ocorreu e como, por quem e quando foi respondida, mas tão somente que aconteceu e estimar o custo incorrido na sua resposta. Então, considerar-se-á uma sequência de interrogações geradas aleatoriamente, *Isq*, em que cada interrogação é caracterizada pelo código do subcubo (ou número), uma frequência e uma extensão (um conjunto de parâmetros consistente com aqueles incluídos no modelo generalizado de custos descrito atrás). Supor-se-á também que não há tempos mortos, o que significa que, se é possível lançar uma interrogação, esta estará disponível.

Anteriormente, em 3.2.2 e 3.2.3, para lidar com a paralelização, introduziu-se o conceito de "janela de execução", uma forma de sincronizar a resolução de conflitos de dados ou recursos. Agora, ele vai ser substituído pela técnica *pipelining*, usada amiúde para descrever e compreender a operação dos processadores [Hennessy & Patterson, 2002]. Com o *pipelining*, é possível executar várias instruções sobrepostas. É um esquema tipo linha de montagem numa fábrica: cada estágio do *pipeline* executa uma fracção da instrução, tal como numa linha de montagem cada operação contribui para a criação da peça no seu estado final. O paralelismo é evidente: no mundo dos processadores, tem-se um conjunto de recursos (replicado total ou parcialmente) e instruções que operam os dados. No processamento de interrogações numa arquitectura M-OLAP tem-se um conjunto de recursos (capacidade de processamento em nós e um débito de comunicação em cada conexão), um conjunto de interrogações a dar resposta e subcubos OLAP a ser utilizados. No mundo dos processadores, há conflitos de recursos e dados. O primeiro acontece quando um dado a utilizar num *pipeline* não está ainda disponível (pois que está a ser calculado num outro pipeline). O segundo acontece simplesmente quando um recurso necessário à execução do

próximo estágio do *pipeline* não está disponível. A forma de resolução destes conflitos pode ser mais ou menos complicada:

1. reordenamento de instruções;
2. *forwarding* ou *bypassing*, tentando tornar um recurso disponível, antes do *términus* normal da execução onde está a ser gerado;
3. ou simplesmente através da introdução de uma ou mais "bolhas" (*stalls*) no pipeline, durante as quais este permanece inactivo (à espera que os recursos sejam disponibilizados).

Transponham-se estes mecanismos para o algoritmo em concepção.

Na resposta a interrogações não há conflitos de dados, uma vez que: a) os subcubos já foram refrescados e estão disponíveis para interrogações, e é suposto que a busca e eventual agregação são executadas no mesmo nó; b) o suposto *pipeline* terá um estágio de processamento que lidará com ambas as tarefas. Assim, restam os recursos como possibilidade para a origem de conflitos: a utilização simultânea do mesmo nó ou conexão de comunicação. Na arquitectura M-OLAP não há o relógio do microprocessador para impor a sincronização temporal dos estágios do pipeline. Mas os atrasos impostos pelos conflitos de recursos devem ser respeitados. Vai aqui usar-se a técnica de *scoreboarding*, para administrar a gestão dinâmica dos pipelines. O *scoreboard* controla a execução das instruções, criando uma imagem das dependências e necessidades, e monitorizando os recursos, para decidir quando uma instrução pode ser lançada [Hennessy & Patterson, 2002]. A detecção de conflitos e a sua resolução é da sua responsabilidade. Assim, cada recurso é rotulado pelo seu tempo de libertação, tl_r, sendo contado desde o instante *ts* (início da sincronização dos *pipelines*).

O algoritmo mantém actualizado o tempo global de execução actual (*tg*) dos *pipelines*, que é utilizado para controlar o tempo de execução final em cada estágio do *pipeline* onde decorre o processamento de cada interrogação. Em cada estágio do *pipeline*, quando um recurso é necessário, o *scoreboard* controla se $tl_r <= tg$, condição cuja satisfação permite a utilização do recurso. Uma resposta negativa causa uma espera (*stall*), até que o recurso seja libertado. Para sincronizar periodicamente os *pipelines*, é estipulado um tamanho máximo para o lote (*tml*), que corresponde ao tamanho máximo do lote de interrogações a ser respondido como um todo, antes que um novo lote seja admitido. Na prática, isto previne a existência de grandes diferenças entre tempos de lançamento de interrogações nos vários pipelines e permite a limpeza periódica das estruturas de controlo do *scoreboard*, bem como a inclusão de alguns custos de comunicação,

como se verá. O *tg* que permite a libertação de todos os recursos (significando que todas as interrogações foram processadas) é adicionado aos custos de interrogação totais, antes que a sincronização dos *pipelines* e o *reset* de *tg* sejam efectuados. Este processo de sincronização não é estritamente obrigatório à operação correcta do algoritmo, podendo *tml* ser arbitrariamente grande.

```
Input: L // Lattice com todos os subcubos, pesos wi's e dependências (MDep) e tamanhos
    Ei=(Ei1,1...n,...,Ein,...n // extensão de utilização de interr. De cada nó/subcubo
    Fi,=(Fi1...n,1,...., Fi1...n,n,,Si) // Distrib. das interrogações pelos nós
       // e sequência respectiva Si
    M=(Cu1...n,1,...., Cu1...n,n) // Distribuição de subcubos pelos nós
    P=(P1... Pn.) // Capacidade de processamento em cada nó
    X=(X1... Xn.) // Conexões e parâmetros respectivos
    dmr, tml // distância máxima de reordenação e tamanho máximo do lote
Output: C(I,(M,Fi,Ei,P,X))// custo de resposta às interrogações Fi dado M, Fi,
         // Ei, P e X
1. Inicialização: C(I)←0; // custo de interrogação=0
2. Processa cada lote de interrogações:
    tg←0; // reset da hora global dos pipelines
    tap(1... nnós)←0; tlr(1...nrec) ←0; // reset do tempo actual do pipeline (tap)
              // e tempo de libertação de cada recursos (tlr)
    pip←Ø; // no início, pip (pool de interrogações possíveis) vazia
  2.1.Para cada lote Ln na sequência Si, fazer:
   2.1.1. Carrega pip e pré-processa interrogações admitidas:
       // pip é a pool de interrogações possíveis
    2.1.1.1. Enquanto Ln≠ Ø E Enquanto pip≠ cheio
       pip←pip ∪ próxima interrogação i ∈ Ln;
       pré-processar i; // calcular o custo de processamento e comunicação
            // da interrogação admitida na pip e nó respondente
   2.1.2. Carregar pipelines:
    2.1.2.1. Enquanto pip≠Ø E pipeline disponível e possível
      Lança interrogação Ia; //a+ antiga interr. da pip que obedeça ao constrangimento
      pip←pip \ Ia; // a interrogação lançada no pipeline é retirada da pool
      Actualiza tap, tlr; // actualiza tempo actual do pipeline onde foi lançada Ia
       // (considerando os seus tempos de execução) e actualiza tlr relativo aos
       // recursos alocados para execução de Ia
   2.1.3. Verifica términus de execuçãode um qualquer estágio num qq. pipeline
       ou insere uma bolha (stall):
    2.1.3.1. Quando Próximo estágio em qualquer pipeline que termine, Fazer:
      Tornar recurso e pipeline disponível;
      Actualizar tg;
    2.1.3.2. Quando ∃ Conflito, Fazer: // conflito de recurso que impede
          // o lançamento de uma qualquer Ia existente na pip
      tg ← tg + tempo necessário à resolução do conflito; // bolha no pipeline
3. Actualizar custos:
   3.1. C(I)←C(I)+tg; // totaliza custo total
4. Repete 2 e 3 até que todos os lotes estejam processados
5. Devolver Resultado: Return C(I);
```

Algoritmo 3.10. Algoritmo de Execução Paralela de Interrogações em Pipeline com Alocação de Nós por Reordenação em Janela Constrangida (APIRC).

Veja-se como decorre o processo de atribuição das interrogações aos nós de processamento. A maximização da utilização dos recursos será assegurada se a próxima interrogação seleccionada para lançar não implicar quaisquer tempos mortos, iniciando a sua execução num nó, logo que a interrogação prévia (em execução nesse nó) tenha terminado, implicando, porventura, a reordenação da ordem de execução das interrogações. Mas esta reordenação distorce a sequência de interrogações aleatória (indo buscar a interrogação mais favorável). Por outro lado, numa situação real, uma reordenação com um deslocamento muito generoso implicaria um tempo de resposta muito longo (simplesmente o lançamento - e consequente execução da interrogação - era adiado de um generosamente excessivo Δt). Desta forma, vai especificar-se um parâmetro *dmr* (distância máxima de reordenamento) que corresponde à distância máxima permitida ao *scoreboard* para procurar a próxima interrogação a lançar. Para simplificar o algoritmo, assume-se que o número de *pipelines* é igual ao número de nós da arquitectura M-OLAP (supondo que os custos de processamento são dominantes na operação do *pipeline*) e que cada nó é alocado a um, simplificando dessa maneira a resolução de conflitos. Esta simplificação não é absolutamente irrealista: se *tml* for baixo, a alocação de tarefas aos nós será a mesma, ainda que o mapeamento nó-*pipeline* não seja utilizado.

Depois da apresentação formal do algoritmo, e dada a sua complexidade, vai exemplificar-se o seu funcionamento supondo o lote de interrogações apresentado na Tabela 3.4, a processar num sistema M-OLAP com 3 nós, como foi mostrado na Figura 3.14. Desta forma, e atendendo ao atrás referido, ter-se-á um processamento paralelo com 4 *pipelines* (P0, P1, P2 e P3). Seleccionaram-se *dmr*=4 e *tml*=16. Uma vez que a atribuição de uma interrogação a um *pipeline* (nó) vai depender do nó respondente e só de entre as *dmr* interrogações seguintes, importa criar uma *pool* com as *dmr* interrogações possíveis, seguidamente referida com o acrónimo *pip*, e já pré-processadas (calculados os custos de busca, de agregação e de comunicação, e nó respondente). Uma vez uma interrogação atribuída, é feita a busca e pré-processamento da interrogação seguinte para a *pip*.

Quanto ao processo de atribuição das interrogações aos vários pipelines, há que tomar em consideração a disponibilidade do recurso necessário (aqui reduzida à disponibilização da conexão de comunicações e de uma interrogação relativa ao nó na *pip*).

Considere-se a sequência de interrogações apresentada na Tabela 3.4, que corresponde a um lote qualquer. Na Figura 3.16 mostra-se o respectivo diagrama da sequência temporal de execução, por execução paralela utilizando quatro *pipelines*, que permite acompanhar a descrição que se segue.

Tabela 3.4. Sequência de interrogações usada pelo algoritmo APIRC numa arquitectura OLAP com 3 nós (mais nó 0 - base).

Interrogação	Nó origem	Nó respondente	Custo de Busca e Agregação	Custo de Comunicação
I1	2	3	10	1
I2	1	2	5	2
I3	1	1	15	0
I4	2	1	10	1
I5	2	2	5	0
I6	1	0	30	10
I7	3	2	10	2
I8	3	1	5	1
I9	1	2	10	7
I10	2	0	25	1
I11	1	3	10	2
I12	1	2	5	1
I13	2	3	10	2
I14	1	1	15	3
I15	3	1	10	2
I16	1	3	10	1

No início, vai carregar-se a *pip* com I1, I2, I3 e I4. I3 é atribuída a P1; segue-se o carregamento de I5 na *pip*. I2 é então atribuída a P2, utilizando a conexão n2→n1, dos 5 aos 7 ms; carrega-se, na *pip* I6. I1 é atribuída a P3, utilizando a conexão n2→n3, dos 10 aos 11 ms; na *pip* é carregada I7. I4 e I5 não podem ser atribuídas (lançadas) já que o *pipeline* necessário (P2) está agora ocupado. O único disponível é P0, sendo-lhe atribuída I6 e carregada I8 na *pip*. 5ms depois, a *slot* de processamento de P2 fica livre, o que permite lançar nova interrogação, se disponível.

Em t=5ms, a *pip* contém I4, I5, I7 e I8. I5 pode ser lançada em P2, não carecendo de qualquer conexão; na *pip* é carregada I9. Em t=10ms, a *slot* de processamento de P2 e P3 ficam novamente livres, pelo que podem ser lançadas novas interrogações. Neste momento, a *pip* contém I4, I7, I8 e I9. Destas, só I7 é processada num dos *pipelines* disponíveis, P2, sendo-lhe atribuída, utilizando a conexão n2 → n3 (entretanto já livre) dos 20 aos 22 ms; I10 é carregada na *pip*, mas P3 continua a não poder ser carregado, uma vez que I10 não pode ser aí processada.

Em t=15ms, P1 fica livre também. I4 é-lhe atribuída, sendo utilizada igualmente a conexão n1 → n2 dos 25 aos 26 ms; I11 é carregada na *pip* (em t=15 ms), ficando lá I8, I9, I10 e I11. Como P3 espera por uma interrogação, I11 é-lhe imediatamente atribuída (tendo ficado em espera por 5 ms – ocorreu uma bolha); usa também a conexão n3→ n1 dos 25 aos 27 ms; I12 é carregada na *pip*, que passa a conter I8, I9, I10 e I12. Em t=20ms, o *slot* de processamento de P2 fica livre, podendo ser-lhe atribuída I9, que vai usar a conexão n2→ n1, de 30 a 37 ms; na *pip* é carregada I13. Em t=25 ms, a *slot* de processamento de P1 e P3 ficam livres. I8 é atribuída a P1, que usa

também a conexão n1 → n3 em 1 ms; I14 é carregada na *pip*. I13 é atribuída a P3 e usa n3 → n2 por 2 ms; I15 é carregada na *pip*. Em t=30ms, a *slot* de processamento de P0, P1 e P2 fica livre; a *pip* contém agora I10, I12, I14 e I15. I10 é atribuída a P0, I14 a P1, utilizando as conexões n 0→ n2 e n1 → n3, respectivamente. I12 pode ser lançada na *slot* de processamento de P2, mas n2 → n3 só está disponível 2 ms depois de ser necessária: ocorre um conflito. Mas, se o lançamento de I12 for atrasado de 2ms, o conflito é resolvido, podendo assim I12 iniciar-se em t=32 ms; na *pip* é carregada I16 (a última interrogação). Em t=30ms, só I15 e I16 permanecem na *pip*. A primeira precisa de P1 para ser processada, enquanto a segunda precisa de P3. Em t=35ms, a P3 fica livre e I16 é lançada, usando n3 → n1 por 1 ms. Em t=45ms, I14 é, por último, lançada em P1, e usa n2 → n1 por 3 ms. A última *slot* de processamento a ficar livre é P1 em t=55ms, a que teremos de adicionar 2ms de tempo de comunicação. O tempo total de processamento do lote de interrogações foi efectuado em 57 ms.

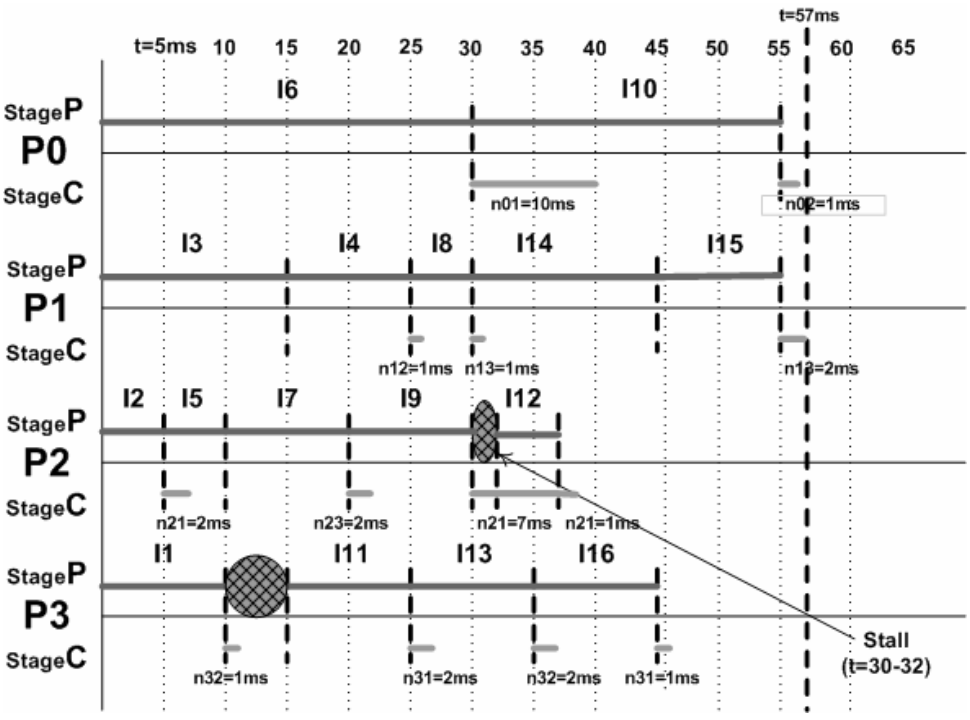


Figura 3.16. Sequência temporal da execução das interrogações do lote mostrado na Tabela 3.4, utilizando quatro *pipelines*.

3.3.3 Algoritmo de Cálculo de Custos de Manutenção: AGM2FHSMP

Na concepção do algoritmo de manutenção, à semelhança de 3.2.3,, o paralelismo é sempre utilizado, pois que assim é conseguida uma maior aproximação dos valores estimados à realidade, embora implicando um aumento de complexidade (e consequente tempo de execução), sabida a possível utilização dos valores de custos de manutenção como constrangimento temporal.

Também agora se concebeu um algoritmo *greedy* em duas fases que, à semelhança do APIRC, é concebido com base num esquema de paralelização tipo *pipelining*. O código AGM2FHSMP é a simplificação do acrónimo de *Algoritmo Greedy de Estimativa de Custos de Manutenção em Duas Fases em Sequência Hierárquica com Simulação Paralela de Processamento de Tarefas em MultiPipelining*. A apresentação formal do algoritmo de estimativa de custos de manutenção é mostrado em Algoritmo 3.11.

Como se pode observar facilmente, este algoritmo é mais complexo do que o anterior, cuja complexidade se relaciona apenas com a gestão do paralelismo, já que só ocorrem conflitos de recursos, pois a resposta a interrogações vai utilizar subcubos já materializados. Neste caso, já podem ocorrer conflitos de dados. Dada a possível utilização de uns subcubos para refrescar outros, não se pode utilizar um subcubo antes dele ter sido efectivamente actualizado. Por outro lado, em cada fase, a pesquisa completa e exaustiva do subcubo que asseguraria, no conjunto, o custo mínimo imporia um stresse elevado na procura da sequência de refrescamento e, tal como no problema da selecção de cubos, seria demasiado onerosa, para ser útil em situações reais. Importa relembrar que este algoritmo deve ser de execução rápida, dada a possível utilização em algoritmos de selecção de cubos. No balancear entre eficácia e eficiência, o prato da balança, neste caso, pende para o lado da segunda. Esta decisão é reforçada pelas conclusões retiradas da avaliação experimental dos algoritmos descritos na secção anterior, Greedy2FSH, Greedy2FPR e Greedy2FPB. Este último, de complexidade $O(n^2)$, onde n é o número de subcubos por nó, não apresentou o desempenho esperado, face à heurística empregue, bastante mais complexa. Revelou mesmo um desempenho muito semelhante aos Greedy2FSH e Greedy2FPR, de complexidade muito menor, estes com tempos de execução quase independentes do número de nós da arquitectura e número de subcubos materializados. Estas motivações vêm fortalecer a opção pela simplicidade do algoritmo que vai ser proposto seguidamente.

```
Input:  L // Lattice com todos os subcubos, tipo de manutenção a aplicar (Mt)
        // e pesos wi's e wu's
     Ea=(A1,...,An) // Extensão de actualização do cubo base
     M=(Cu1...n,1,...., Cu1...n,n) // Distribuição de subcubos pelos nós
     P=(P1... Pn.) // Capacidade de processamento em cada nó
     X=(X1... Xn.) // Conexões e parâmetros respectivos
     tml,tmp,sh // tamanho max. do lote, tamanho max. pspm e seq. process. hierarq.
     nha; // número de níveis de hierarquias cujos subcubos a actualizar
          // devem ser incluídos em cada lote
Output: C (M, L,Ea,P,X) // custo de manutenção de M, dado L, Ea, P e X
        P(M, L,Ea,P,X)  // plano de manutenção de M, dado L, Ea, P e X

1. Inicialização: C(M)←0; // custo de manutenção=0
 M'← Ø  // M' é o conjunto de subcubos já actualizados, no início está vazio
2. Carrega o lote:
 2.1. Bn←Ø; // limpa o lote
 2.2. Bn←Bn ∪ subcubos sc Em M, na sequência sh não Em M' :
   nhb<=nh(sc)<nha E Mt(sc)=tipo especificado; // nh(sc) devolve o nível
     // hierárquico de sc;
     // Mt(sc) devolve o respectivo tipo de manutenção
     // cada lote é carregado com os subcubos ainda não actualizados na sequência
     // hierárquica cujos níveis estejam entre nhb (nível hierárquico baixo) e
     // nha (nível hierárquico alto) e cujo tipo de manutenção seja do tipo
     // I(incremental) na fase 1 ou S (from scratch) na fase 2
3. Processa cada lote de subcubos a refrescar:
 tg←0; // reset da hora global dos pipelines
 tap(1... nnós)←0; tlr(1...nrec) ← 0; // reset do tempo actual dos pipelines e
   // tempo de libertação dos recursos
 pspm←Ø; // no início, pspm (pool de subcubos possíveis a manter) vazia
 3.1.Para cada lote Ln, carregado de M, Na seq. sh,  Fazer:
  3.1.1. Carrega pspm com interrogações já pré-processadas:
   Enquanto (Ln ≠ Ø E tamanho(pspm) <= tmp) // tmp é o tamanho máximo da pspm
   Busca e pré-processa o próximo subcubo s Em Ln; //  custo(s): min. custo
   // (processamento e comunicação), nó/subcubo origem e custos de integração
   pspm ← pspm ∪ s; // carrega s pré-processado na pspm; replicações não
     // são incluídas: são tratadas em 3.1.2
  3.1.2. Carrega pipelines:
   Enquanto (pspm ≠ Ø E pipeline disponível E possível E ¬∃(conflito de
   comunicação ou de disponibilidade de dados))
   Lança processamento do subcubo s no pipeline:
   pspm ← pspm \s; //s=subcubo cujo processamento é lançado num qualquer pipeline
   actualiza tap, tlr; // actualiza o tempo de libertação dos recursos utilizados
   pspm←pspm ∪ integração de s ou Δs;  // lança o processamento da integração
     // de s ou Δs no nó destino em pspm
   Se (s replicado) Então
    pspm ← pspm ∪ comunicação e integração dos s ou Δs replicados;
     // lança o processamento da transferência
     // e integração dos s ou Δs replicados nos nós destino em pspm
  3.1.3. Verifica términus de execução num qq. pipeline ou insere uma bolha
   (stall):
   3.1.3.1. Quando Próximo estágio em Qualquer pipeline que finaliza Fazer
    Recursos e pipeline torna-se disponível;
    Actualiza tg;
```

Algoritmo 3.11. O algoritmo AGM2FHSMP.

O algoritmo AGM2FHSMP (continuado da página anterior).

```
      Se estágio=integração Então
        M'←M' ∪ s;
    3.1.3.2. Quando ∃ Conflito de Comun. Ou  Conflito de Dados: // evita o
      // lançamento de qq. tarefa em pspm
      tg ← tg + tempo necessário à resolução do conflito; // insere uma bolha
       // (devido a conflito de comunicação ou dados)
4. Actualiza custos:
    C(M) ← C(M) + t;  // totaliza custos de manutenção
5. Repete 2 e 3 até que todos os lotes relativos a subcubos a actualizar por
    manutenção incremental estejam  processados
6. Repete 2 e 3 até que todos os lotes relativos a subcubos a actualizar por
    manutenção integral  estejam  processados
7. Devolver Resultado: Return C(M)
```

A heurística baseia-se na premissa base de que a manutenção incremental é substancialmente menos onerosa do que a manutenção integral. Desta forma, o algoritmo é dividido em duas fases distintas: na primeira, vai executar-se a manutenção incremental, deixando para a segunda a manutenção integral que pode beneficiar dos subcubos entretanto refrescados. Cada uma das fases vai ser dividida em duas sub-fases: a primeira lida com os subcubos que só podem ser mantidos utilizando as relações base. Os deltas (ou subcubos) são gerados no nó base, sendo integrados nos nós destino, depois de transportados. A segunda sub-fase prossegue com a manutenção, através da agregação dos deltas primários (ou subcubos gerados), mantidos numa estrutura M' (M actualizada), que são subsequentemente transportados e integrados. Esta sub-fase lida com o refrescamento dos subcubos, segundo uma sequência hierárquica (SH) ditada pelas relações de dependência do *lattice*. É uma abordagem tipo "*breath first, deep last*". O algoritmo selecciona, para refrescamento, os subcubos de nível crescente de granularidade, nível a nível, o que significa que os subcubos mais perto das relações base são actualizados no início, terminando o processo com a actualização do subcubo de agregação total. Em cada um dos níveis, a sequência de actualização é dada pelo número do subcubo. Para cada subcubo calculado, o algoritmo vai procurar replicações, tentando evitar reprocessar deltas ou subcubos. Estas heurísticas permitem justificar parte do nome do algoritmo: *Algoritmo Greedy de Estimativa de Custos de Manutenção em Duas Fases em Sequência Hierárquica*. A simulação da execução paralela com *pipelines* justifica o restante nome: *com Simulação Paralela de Processamento de Tarefas em MultiPipelining*. Tal como os algoritmos propostos em 3.2.3, este vai também produzir, além do cálculo do custo de manutenção, um plano de manutenção.

O processo de paralelização a empreender é algo semelhante ao descrito para o algoritmo APIRC (apresentado na secção anterior), só que agora cada pipeline tem 3 estágios:

1. processamento para a geração do delta ou subcubo;
2. a sua transmissão inter-nós (implicando custos de comunicação);
3. integração do delta ou subcubo gerado no nó destino.

Ainda que não estritamente necessária, a *pip* anterior é também utilizada, embora renomeada para *pspm* (*pool* de subcubos possíveis a manter). A sua utilização permitirá a avaliação do impacto do seu tamanho no desempenho do algoritmo, já que da variação da sua dimensão vai depender a probabilidade das bolhas nos *pipelines*. Também a divisão em lotes permanece pelas mesmas razões atrás invocadas e também porque permite a divisão de cada sub-fase em carregamento de lotes e processamento de lote, quando as tarefas de produção de subcubos (ou deltas) são lançadas nos *pipelines*. Agora, o tamanho dos lotes não é restringido a um número máximo de interrogações, mas por um número de hierarquias a actualizar *nha*. Como, de resto, este algoritmo é semelhante ao APIRC, não vão tecer-se mais considerações acerca do seu funcionamento.

Uma nota apenas quanto à selecção do subcubo de custo mínimo a usar para a manutenção de um outro. O primeiro é procurado em M' (o conjunto dos subcubos entretanto actualizados+relação base), já que inicialmente só a relação base reflecte as alterações havidas. Esta situação obriga ao refrescamento da *pspm*, sempre que um novo subcubo é actualizado (algo como a necessidade do refrescamento da cache de um microprocessador).

3.3.4 Implementação dos Algoritmos de Estimativa de Custos e Ambiente de Teste

Os algoritmos concebidos foram implementados como um método com vários parâmetros, para seleccionar uma de várias formas de estimativa de custos de interrogação e/ou manutenção. Este método está incluído numa classe nuclear chamada M, cujo estado funciona como uma área de entrada/saída. Como entrada, contém os subcubos materializados nos vários nós e a informação acerca do tipo de manutenção de cada subcubo (incremental ou integral), que têm de ser carregados antes de invocar o método de estimativa de custos. Como saída, a classe mantém e disponibiliza os valores dos custos de interrogação e manutenção. Estão também disponíveis outros métodos que permitem estabelecer ou inspeccionar o estado da classe, além de outros de uso interno.

O sistema, como um todo, foi implementado em Java, e é composto por um conjunto de classes que podem ser agrupadas como se mostra a seguir:

- as classes correspondentes ao cubo: com informação relativa aos subcubos (identificação, nível de hierarquia, tamanho), hierarquias dimensionais (incluindo suporte a hierarquias paralelas), cálculo de dependências entre subcubos do *lattice* e sua disponibilização, cálculo generalizado de wi e wm utilizando o algoritmo Dijkstra [Dijkstra, 1959], e frequência e extensão de actualizações;
- a classe ligada às interrogações, identificando os nós onde foram colocadas, a sua frequência e extensão de utilização;
- a classe relativa à arquitectura M-OLAP, com nós servidor OLAP, a sua capacidade de processamento e alguma informação adicional: hardware instalado, localização, etc.;
- classes correspondentes aos parâmetros globais e específicos que permitem ao sistema conhecer como gerar as estruturas de dados dos algoritmos e gerir a sua execução;
- classes relativas às estruturas de dados que suportam o estado gerido pelos algoritmos: *M*, como descrito acima, lote e pool de subcubos possíveis a manter (pspm).

Para executar o estudo experimental, utilizou-se o cubo A, já utilizado em 3.2.4, sendo aqui igualmente válidas as suposições então indicadas.

Dividiu-se a avaliação em duas partes: a primeira (secção 3.3.5) é relacionada com a avaliação dos algoritmos de estimativa de custos de interrogação; a segunda (secção 3.3.6) avalia o algoritmo de estimativa de custos de manutenção. Além disso, como foram concebidos dois algoritmos de cálculo de custo de interrogações, empreendeu-se a sua análise comparativa: avaliou-se o seu desempenho em termos dos custos calculados e tempo de execução, variando o número de interrogações e nós da arquitectura M-OLAP. O teste do algoritmo de manutenção mostra o seu desempenho em termos dos custos de manutenção e tempo de execução, variando o número e o tamanho total dos subcubos materializados, o tipo de manutenção e o número de nós da arquitectura M-OLAP.

Como teste preliminar de verificação, utilizou-se o lote da Tabela 3.4 e os resultados obtidos corresponderam à sequência temporal mostrada na Figura 3.16 e à discussão realizada no final da secção 3.3.2.

3.3.5 Avaliação Comparativa dos Algoritmos de Estimativa de Interrogações

Para este conjunto de testes usou-se uma arquitectura M-OLAP com três nós (à imagem da mostrada na Figura 3.14), cujo espaço de materialização é suposto ser suficiente para armazenar todos os subcubos possíveis. A distribuição dos subcubos materializados (*M*) é gerada de forma aleatória, com um tamanho máximo de 1, 2, 3, 4, 5, 6, 8, 10, 15, 20, 30, 40, 50, 60% do espaço de materialização total do cubo em cada nó. Os valores dos custos de interrogação e tempos de execução foram obtidos utilizando cinco conjuntos de interrogações aleatórias (com 40, 80, 120, 160 e 200 interrogações, respectivamente). O teste foi repetido 5 vezes, sendo então gerado um número correspondente de diferentes Ms. Os resultados mostrados nos gráficos são a média dos valores obtidos em cada execução dos algoritmos.

Os valores do custo de interrogação são mostrados na Figura 3.17. O custo calculado pelo algoritmo APIRC (com execução paralela, à esquerda na figura) é menor do que o valor retornado pelo ACCIESST (série, à direita na figura). Para valores baixos de espaço de materialização, ambos os algoritmos calculam valores quase iguais (por exemplo para 0%-4%, a razão é, em média, 1.00-1.05, respectivamente). Para 5% do espaço de materialização, a razão salta para 1.83, subindo para 2.06 para 30% de espaço de materialização. A explicação para este comportamento reside no facto de, para um espaço de materialização baixo, muitas das interrogações terem de ser respondidas pelas relações base (no nó 0). Desta forma, os NSOs ficam inactivos e nenhuma paralelização é possível. Quando o número de subcubos cresce (duplicou entre 4% a 5% de espaço de materialização), o paralelismo acorda e os custos de interrogação caem. Mas a razão entre os custos de interrogação é sempre menor do que o valor teórico máximo: três. Isto pode ser explicado pela acção conjunta de vários factores:

- como foi utilizado um valor baixo para *dmr* (8), a probabilidade de conflitos de recursos de processamento é elevada;
- a probabilidade de conflitos de comunicação;
- só a segunda sub-fase do APIRC é paralelizável, e só usando os nós 1, 2 e 3, o que limita o poder de paralelização das tarefas.

Em ambos os gráficos, pode ver-se uma relação directa entre o custo de processamento de interrogações e o número de interrogações, e uma relação inversa entre o custo de interrogações e o espaço para materialização de subcubos, mais evidente para valores baixos. Isto é

particularmente interessante, já que com valores baixos de espaço de materialização podem obter-se ganhos importantes em termos de custos de resposta às interrogações.

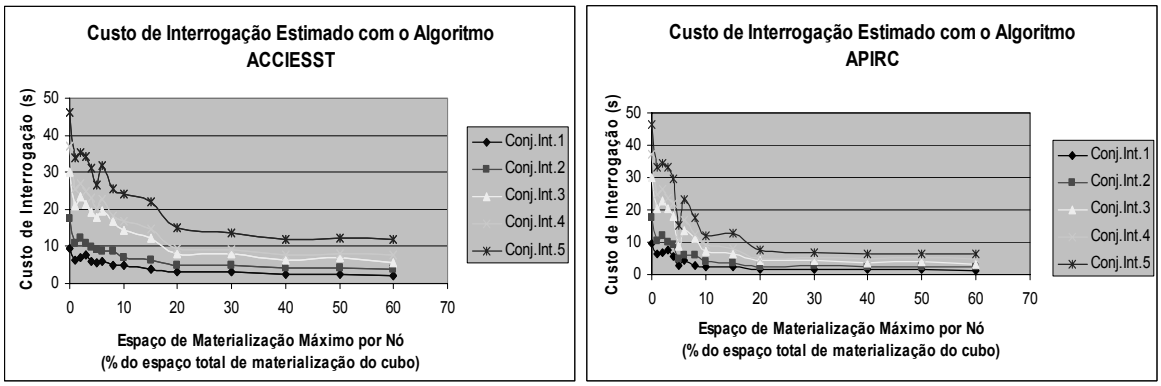


Figura 3.17. Comparação dos custos de interrogação estimados pelos algoritmos ACCIESST (à esquerda) e APIRC (à direita) numa arquitectura M-OLAP cm 3 nós.

Os gráficos da Figura 3.18 mostram o tempo de execução dos algoritmos ACCIESST (à esquerda) e APIRC (à direita). O primeiro exibe um tempo de execução substancialmente menor, em relação directa com a sua complexidade. Além disso, o tempo de execução do ACCIESS é aproximadamente constante para todos os valores do espaço de materialização e praticamente independente do número de interrogações.

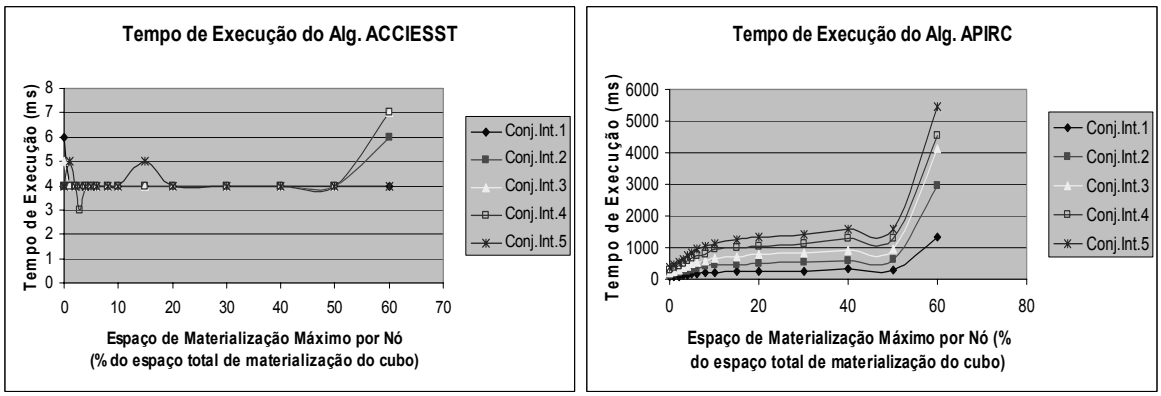


Figura 3.18. Análise do tempo de execução dos algoritmos ACCIESST (à esquerda) e APIRC (à direita) numa arquitectura M-OLAP com 3 NSOs.

Já o APIRC revela um comportamento diverso: exibe uma relação directa entre o seu tempo de execução e o número de interrogações ou espaço de materialização. A explicação para esses comportamentos é devido a opções de concepção. O ACCIESST inspecciona todos os nós e subcubos e então procura por interrogações relacionadas. No APIRC, como há lotes, uma interrogação é tratada individualmente, e só se existir. Os saltos inesperados observados nos valores podem ser motivados pela criação de estruturas de dados a cargo do ambiente de execução e também por causa da execução concorrente de outras tarefas do sistema operativo.

O último teste deste conjunto tenta avaliar o impacto do número de nós da arquitectura M-OLAP no tempo de execução dos algoritmos e também nos custos estimados de resposta às interrogações, o que permite obter algumas indicações acerca do "*scale-up*" e "*speed-up*" da arquitectura M-OLAP. Foram utilizadas duas arquitecturas M-OLAP: a usada nos testes explicitados anteriormente (com três nós) e outra com seis nós. Executados os algoritmos, foi registado o tempo de execução e medida a razão de custos de interrogações. Para avaliar o "*speed-up*", aplicaram-se cinco conjuntos de interrogações (com o mesmo número de interrogações) à arquitectura M-OLAP com 3 e 6 nós. Quando o "*speed-up*" foi testado, aplicou-se à arquitectura com 6 nós um conjunto de interrogações com 2 vezes (6/3) o número de interrogações correspondentes aplicadas à arquitectura com 3 nós. Os resultados relativos à razão entre tempos de execução são mostrados na Figura 3.19, para o algoritmo APIRC, pois que ambas as grandezas em avaliação têm que ver com a eficiência e eficácia do paralelismo.

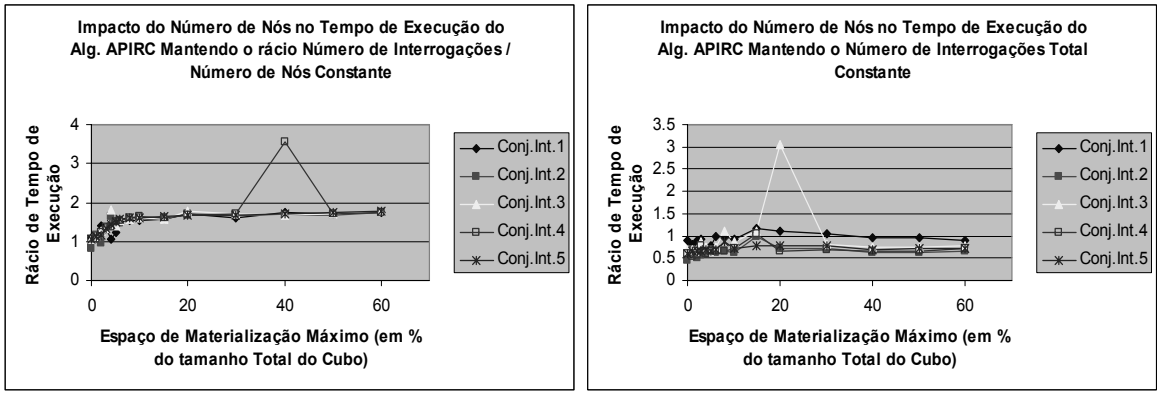


Figura 3.19. Impacto do número de nós da arquitectura M-OLAP no tempo de execução do algoritmo APIRC.

A observação destes gráficos mostra que o APIRC suporta facilmente arquitecturas de muitos nós. Mantendo o número de interrogações (Figura 3.19, gráfico da esquerda), a razão entre tempos de execução para M-OLAP com 3 nós x 6 nós é 0.85 (para espaços de materialização > 5%, quando o paralelismo da arquitectura realmente funciona). Se o número de interrogações foi aumentado de um factor igual ao aumento do número de nós (Figura 3.19, gráfico da direita), a razão observada entre tempos de execução foi de 1.7 (próxima da razão entre o número de interrogações). Estes resultados mostram que o número de interrogações, e não o número de nós, é relevante para o tempo e execução do algoritmo APIRC.

Os gráficos da Figura 3.20 estão relacionados com o impacto observado do número de nós no tempo de resposta estimado às interrogações colocadas. O "*speed-up*" (gráfico da esquerda) mostra valores entre 0.4 e 1 para o rácio (razão dos custos de interrogação da arquitectura M-OLAP com 3 e 6 nós), com um valor médio de 0.63 (para espaços de materialização > 5%). Isto significa que um aumento de 6/3 no poder da arquitectura consegue uma diminuição de (1/0.63=1.69) no tempo de resposta (custos de interrogação), com os valores mais salientes para espaços de materialização de 6 e 8% (onde foram observados valores de 0.53 e 0.56), mostrando um bom "*speed-up*" da arquitectura. Relativamente ao "*scale-up*" (gráfico da direita na Figura 3.20), para espaços de materialização acima de 5%, podem observar-se valores entre 1 e 1.5, com um valor médio de 1.15, próximo do valor ideal, mostrando um bom "*scale-up*" da arquitectura M-OLAP. Deve referir-se também que, para espaços de materialização no intervalo [6, 15]%, este valor para o "*scale-up*" é igualmente válido.

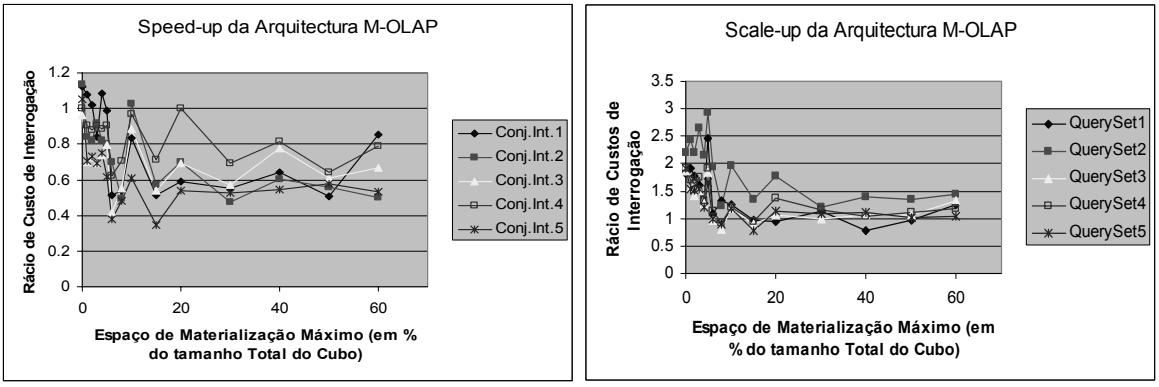


Figura 3.20. Impacto do número de nós da arquitectura M-OLAP nos custos de resposta a interrogações.

3.3.6 Avaliação do Algoritmo de Estimativa de Custo de Manutenção

Neste conjunto de testes, pretendeu avaliar-se o algoritmo AGM2FHSMP. O teste inicial deste conjunto foi concebido para mostrar o impacto do número de subcubos em *M* no tempo de execução do algoritmo e também avaliar o "*scale-up*" da arquitectura M-OLAP relativamente ao aspecto importante do tempo de refrescamento das suas estruturas de optimização, utilizando este modelo de custos da arquitectura e respectivo algoritmo. *M* foi carregado com distribuições de subcubos geradas aleatoriamente, a ser materializadas em qualquer dos nós da arquitectura, para espaços de armazenamento seleccionados; estimaram-se os custos de manutenção para arquitecturas M-OLAP com 3 e 6 nós (mais o nó base), identicamente ao ambiente utilizado no último teste da secção anterior. Cada valor foi calculado como a média do custo de manutenção e tempo de execução de 30 *M*s diferentes, para cada espaço de materialização seleccionado. O gráfico da Figura 3.21 (à esquerda) mostra o resultado deste teste. A razão entre o custos de manutenção observado para as arquitecturas de 3 e 6 nós mostra valores próximos de 1 (0.8 e 0.86 para o intervalo [6-15]%). Isto mostra a escalabilidade da arquitectura. A adição de nós suplementares tem um impacto reduzido nos custos de manutenção.

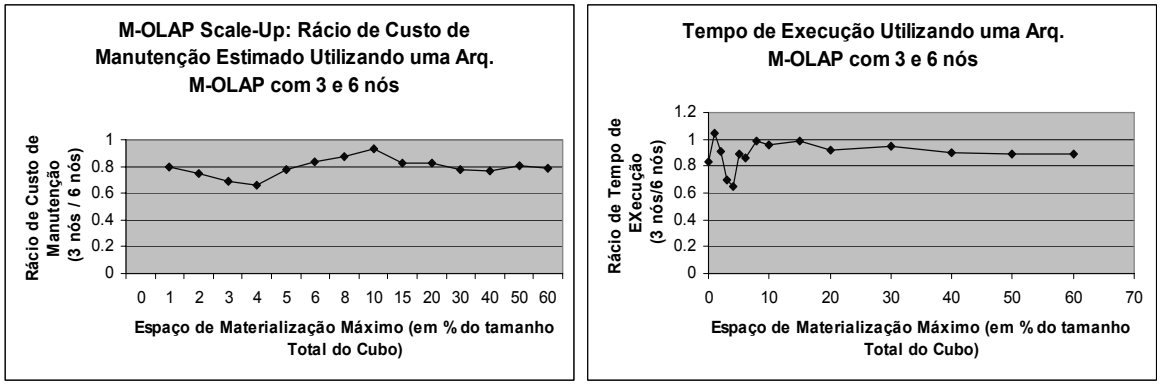


Figura 3.21. Custos de manutenção e tempo de execução relativas a distribuições aleatórias de subcubos.

Também o tempo de execução do algoritmo AGM2FHSMP para as duas arquitecturas (gráfico da Figura 3.21, à direita) mostra uma razão de 0.9, significando que um aumento de duas vezes no número de nós teve um impacto reduzido no tempo de execução do algoritmo.

Os dois aspectos concomitantes, salientados a propósito dos dois gráficos da Figura 3.21, são características muito desejáveis da arquitectura M-OLAP e algoritmo AGM2FHSMP: a adição de

NSOs extra tem um impacto reduzido nos custos de manutenção, mas dado o "scale-up" da arquitectura no que toca aos custos de interrogação, são esperados grandes benefícios em termos da satisfação dos utilizadores OLAP. Por outro lado, o desempenho evidenciado pelo APIRC permite suportar um número ainda mais elevado de NSOs na arquitectura M-OLAP.

O segundo teste pretendeu avaliar o "*speed-up*" da arquitectura e desempenho relativo do algoritmo. Mantendo o armazenamento global igual, vão materializar-se subcubos de forma aleatória numa arquitectura M-OLAP de 3 e 6 NSOs. O resto das condições do teste são idênticas às do anterior. Os resultados deste teste estão apresentados na Figura 3.22. Nela pode observar-se (no gráfico da esquerda) que os custos de manutenção da arquitectura com 6 nós são menores, mas algo mais baixos do que o limite teórico. De facto, a razão 6/3 cai para uma média de 1.72 no intervalo [6-15]%. Mais uma vez, a causa pode ser encontrada no facto de a sub-fase 1, em ambas as fases, ser quase sequencial, o que prejudica a paralelização global das tarefas. Quanto ao tempo de execução (ver Figura 3.22, gráfico da direita), os valores obtidos mostram um aumento com o número de nós, ainda que menor do que 2 (observou-se uma média geral de 1.27 e 1.49 no intervalo [6-15]%).

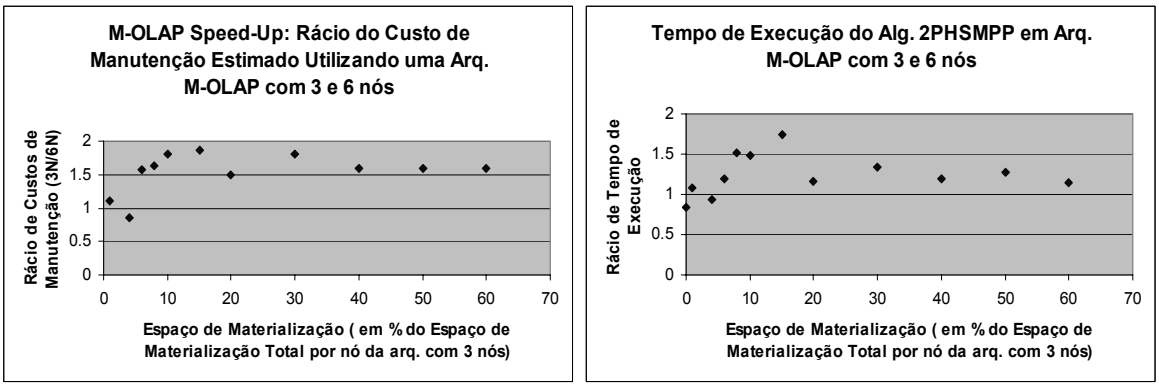


Figura 3.22. Rácios de custo de manutenção e tempo de execução relativas a distribuições aleatórias de subcubos.

3.3.7 Discussão do Modelo não-Linear e dos Algoritmos Desenvolvidos

O modelo linear distribuído apresentado em 3.2.1 foi refinado e generalizado, pela inclusão de não linearidades e, em resposta a algumas limitações detectadas, inclui agora uma maior diferenciação dos custos e seus elementos contribuintes. Estas alterações geraram um aumento de complexidade que foi, no entanto, limitada, pelo não refinamento de alguns dos novos pesos e

parâmetros ao nível do nó, na sua individualidade, assumindo-se com isso algum erro de estimativa. No entanto, como já foi dito, tornar a granularidade de alguns dos elementos constituintes do modelo mais fina é bastante fácil, já que o seu impacto é limitado e o referencial desenvolvido é suficientemente genérico para permitir a sua acomodação. Essencialmente, o suporte simultâneo e mais preciso do cálculo dos custos de manutenção, nas suas vertentes integral e incremental, é o aspecto mais saliente deste novo modelo.

Acompanhando o refinamento e generalização do modelo, os novos algoritmos concebidos assumiram uma evolução similar:

a) o seu refinamento foi conseguido:

- pela utilização do esquema de simulação de processamento paralelo utilizando *multi-pipelining* e *scoreboarding* [Hennessy & Patterson, 2002] que, no modelo anterior, tinha sido aproximado pela discretização temporal conseguida com o conceito de janela de execução e transacção;
- pela inclusão de um conjunto de parâmetros e pesos como elementos de entrada que permitiram uma diferenciação dos custos a um nível mais detalhado e, assim, uma maior precisão, especialmente notória para os custos de manutenção;

b) a sua generalização decorre, em alguma medida, das especializações introduzidas, o que pode parecer um contra-senso: as não linearidades e os novos pesos e parâmetros permitiram o suporte simultâneo de custos de manutenção integral e incremental, sendo assim possível a sua utilização em arquitecturas mais genéricas, onde os vários tipos de manutenção coexistirão a diferentes níveis.

Os algoritmos aqui propostos constituem uma ferramenta nuclear. A generalidade do modelo subjacente, o seu suporte a um conjunto alargado de requisitos e a sua implementação em componentes torna-os adequados a ser utilizados em algoritmos de selecção e alocação de cubos de estruturas multidimensionais centralizadas ou distribuídas e na presença de constrangimentos (espaciais e temporais de manutenção), sendo mesmo capazes de providenciar a optimização destas estruturas numa arquitectura dum DW híbrido global distribuído.

Além disso, os algoritmos são de fácil utilização, face à existência de um número de parâmetros e classes alargado, mas cujo significado é claro e específico, onde sobressai a forma simples de definição da estrutura do *lattice*. Esta é uma característica introduzida logo na implementação dos algoritmos correspondentes ao modelo linear centralizado 3.1.1, e que, dada a sua especial

efectividade, foi mantida nas sucessivas gerações de algoritmos. O algoritmo então descrito (Algoritmo 3.2 da secção 3.1.2) permite, por especificação das dimensões, hierarquias (incluindo hierarquias paralelas) e composição de cada subcubo, calcular e registar todos os relacionamentos entre subcubos. Esta geração não se limita às dependências directas, mas pode generalizar-se a qualquer par de subcubos relacionados a qualquer nível da hierarquia do *lattice*; utilizando esta informação nos algoritmos de manutenção hierárquicos, foi também desenvolvido um algoritmo que gera os níveis de cada subcubo no *lattice*. É de salientar que estas relações são calculadas uma vez, sendo registadas de forma a serem disponibilizadas em todas as subsequentes pesquisas de antecessores de subcubos, em algoritmos de estimativa de custos de interrogação ou manutenção, evitando o seu recálculo continuado (sempre que uma nova interrogação ou subcubo tivesse de ser processado). Este mesmo conceito foi aqui utilizado também no cálculo dos pesos *wi* e *wm*: o algoritmo *Dijkstra* [Dijkstra, 1959], [Cormen et al., 2001] é utilizado para calcular caminho de custo mínimo entre qualquer par de nós, evitando o seu recálculo, sempre que fosse depois necessário calcular um peso entre dois nós específicos.

Também, dada a natureza do modelo, algumas especificidades relacionadas com a existência de índices ou ordenações pode ser suportada. Para alargar a aplicabilidade dos algoritmos, o custo de manutenção de uma distribuição M de subcubos pode ser estimada, considerando que cada subcubo pode ser mantido integral ou incrementalmente, capacitando-o para utilização em optimização estática ou dinâmica. A natureza paralela da arquitectura M-OLAP forçou a inclusão de conceitos de *pipelines* e gestão de execução de processos na concepção dos algoritmos APIRC e AGM2FHSMP, tentando com isso mimetizar a execução real da execução da resposta às interrogações e do processo de manutenção.

Os resultados da avaliação experimental mostraram que, globalmente, os algoritmos propostos são eficientes e escaláveis. Relativamente à estimativa dos custos de interrogação, os testes efectuados mostraram que o tempo de execução do algoritmo ACCIESST é quase independente do número de interrogações e espaço de materialização disponível e com um valor bastante baixo; o tempo de execução do APIRC é mais dilatado (duas ordens de grandeza maior do que para o ACCIESST), mas só dependente linearmente do número de interrogações. Ambos os algoritmos são escaláveis (já que o número de nós tem um impacto reduzido no seu tempo de execução). Como regra geral, pode dizer-se que, dada a muito menor complexidade do ACCIESST, ele deverá ser eleito sempre que só for pretendida uma estimativa comparativa de custos de interrogação, como por exemplo quando se pretende uma informação comparativa de conjuntos de interrogações, ou distribuições ou quando se pretende avaliar a elegância de duas distribuições de

subcubos *M* diferentes. Já o APIRC será útil quando for pretendido um valor mais aproximado da realidade dos custos de resposta a interrogações, uma vez que leva em conta o inerente paralelismo da arquitectura M-OLAP. Esta informação permite conhecer a magnitude aproximada do tempo de resposta a interrogações e avaliar a satisfação dos utilizadores.

Olhando para os resultados obtidos com o algoritmo AGM2FHSMP, pode dizer-se que este é escalável. O seu tempo de execução cresce a uma taxa menor do que o crescimento do número de nós da arquitectura, e é também clara a fraca dependência relativamente ao espaço de materialização (e, indirectamente, ao número de subcubos materializados).

Em termos gerais, o "*speed-up*" e "*scale-up*" evidenciados pelos gráficos relativos aos custos de interrogação e manutenção, usando o modelo proposto, permitem concluir que a arquitectura M-OLAP é benéfica, já que permite uma redução de custos de interrogação com um impacto limitado nos custos de manutenção. Trata-se de um comportamento muito interessante, visto que o custo financeiro de vários servidores OLAP simples é substancialmente menor do que o custo de um único servidor centralizado com a mesma capacidade agregada. De igual forma, uma vez que o hardware se foi tornando sucessivamente mais barato, novos servidores podem ser adicionados a uma arquitectura M-OLAP a um custo baixo, desde que os custos de gestão sejam controlados, o que é possível se um esquema de optimização quase automático estiver disponível, capaz de executar a geração da distribuição OLAP apropriada, através da aplicação de algoritmos de selecção e alocação de cubos. Discutiu-se também que este sistema pode ser alargado na sua abrangência espacial. Através da inclusão de alguns novos parâmetros no modelo de cálculo dos custos de comunicação, é possível distribuir os NSOs por localizações geograficamente dispersas, interligadas por uma WAN, e possivelmente alargar os benefícios da distribuição, uma vez que a alocação sugerida para os dados seguirá a sua utilização, baixando previsivelmente os custos de comunicação.

Capítulo 4

Selecção de Cubos em OLAP Centralizado

4.1 Optimização em Problemas Combinatórios NP-hard

Problemas de optimização surgem em quase todos os domínios, desde a pesquisa e desenvolvimento científico à aplicação industrial. Eles aparecem quando a tarefa é encontrar a melhor de entre muitas soluções possíveis para um dado problema, desde que exista uma noção clara de como avaliar a qualidade da solução. Contrariamente a outros problemas de optimização, os problemas de carácter combinatório têm um número finito de soluções candidatas. Assim, uma forma óbvia de resolução seria a enumeração de todas as soluções candidatas, comparando-as. Infelizmente, para a maioria destes problemas, esta abordagem não é praticável, já que o número de soluções possíveis é simplesmente muito grande. É o caso da selecção de vistas ou cubos a materializar para minimização de custos em interrogações, especialmente de natureza agregada [Harinarayan et al., 1996], como, aliás, já foi dito e parcialmente mostrado na secção 2.7, quando foi calculado o número de possíveis soluções de *M*, mesmo para um esquema dimensional muito simples. No entanto, a dificuldade não reside apenas no número de soluções para *M* de uma dada instância, mas o seu comportamento relativamente aos factores que a determinam. A noção de dificuldade do problema é capturada pela teoria da complexidade computacional [Garey & Johnson, 1979], [Papadimitriou, 1994], que caracteriza o problema como NP-hard quando o tempo necessário para resolver uma instância cresce, no pior caso, exponencialmente com o tamanho da instância.

Para alguns problemas de carácter combinatório, foram encontrados algoritmos que são muito mais rápidos do que a pesquisa exaustiva, continuando a obter uma solução óptima: diz-se que correm num tempo polinomial, dependente do tamanho do problema. Dois exemplos a referir são: as

abordagens *Branch & Bound* e *Branch & Cut*, ambas baseadas em algoritmos de pesquisa em árvore, sendo o segundo um híbrido do primeiro com o método planos de corte. O algoritmo *Branch & Bound* [Land & Doig, 1960] utiliza um método para encontrar um limite superior e inferior para a solução óptima (*bound*) e um esquema de enumeração que funciona por divisão sucessiva do problema geração, construindo uma árvore com os seus ramos, daí o nome *branch*. Este esquema foi proposto para solução do problema clássico do viajante (*Travelling Salesman Problem*-TSP) em [Moller & Pekny, 1991], [Fischetti & Toth, 1992]. No entanto, para muitos dos problemas, esta abordagem não é utilizável, pois que parece não haver relaxamentos discretos que sejam suficientemente fortes para resolver grandes problemas.

A segunda abordagem, *Branch & Cut*, é ainda mais elaborada, sendo baseada em programação linear [Dantzig, 1963], [Hillier & Lieberman, 2005]. Aqui, a ideia base é encontrar um relaxamento na forma de um programa linear com a mesma solução óptima do problema original. Depois, existe um algoritmo de procura típico denominado "simplex" [Hillier & Lieberman, 2005] que assegura sempre uma solução óptima. O seu nome (*cut*) deriva da utilização de um plano de corte que pode ser utilizado recursivamente para cortar um "politope" que é gerado pelo alargamento do espaço de pesquisa com os vectores de solução discreta para vectores de variáveis contínuas, de forma a conter sempre o politope da solução do problema. A cada sucessivo "politope" é aplicado um algoritmo de resolução de programação linear, processo repetido até que a solução óptima seja conseguida ou o algoritmo falhe em encontrar um novo corte factível. Como este último caso é mais provável, há que dispor de uma regra de divisão e geração de ramos, necessária para dividir o problema em subproblemas, aplicando depois o procedimento corte recursivamente a cada um dos subproblemas. Esta necessidade da regra de divisão e geração de ramos dá a segunda parte do nome da abordagem. Este método foi aplicado a uma variedade de problemas. Para o clássico TSP é descrito em [Padberg & Rinaldi, 1987], [Padberg & Rinaldi, 1991]. Contudo encontrar cortes apropriados é uma tarefa muito complicada e, acima de tudo, dependente do problema. No caso do TSP, houve um grande esforço de investigação com vista a desenvolver teorias apropriadas; no entanto, para muitos dos outros problemas de optimização combinatória essas teorias não existem. Assim, a aplicação desta abordagem é, simplesmente, não exequível.

Além da complexidade dos métodos de resolução óptima dos problemas de carácter combinatório e da sua dependência do problema (nas teorias necessárias à sua adaptação), há as limitações referidas para cada abordagem que a inviabilizam para muitos outros casos. Assim, acredita-se que, para muitos problemas de carácter combinatório, um tal algoritmo, que proporcione uma solução óptima num tempo polinomial dependente do problema, não existe. Na grande maioria dos casos, a única

forma de abordar os problemas é aplicar heurísticas de pesquisa que não garantem encontrar a solução óptima, procurando apenas uma solução aproximada. Uma heurística pode assim ser definida como um método de pesquisa que encontra soluções (quase) óptimas, num tempo curto. Em comparação com as abordagens exactas, não garante encontrar soluções óptimas, nem sequer proporciona geralmente uma garantia de encontrar soluções dentro de um certo intervalo da solução óptima.

Apesar disso, as heurísticas são de enorme importância, já que são, na maioria das vezes, a única forma de chegar a soluções de alta qualidade para grandes problemas de carácter combinatório de interesse prático. Além disso, muitas heurísticas têm a vantagem de ser aplicáveis a um leque alargado de problemas e, assim, o tempo para desenvolver um algoritmo de optimização adaptado a um novo problema é normalmente curto.

Também há a referir que estas heurísticas podem facilmente ser modificadas para levar em conta alterações na função objectivo. Se um constrangimento for adicionado à descrição do problema, as heurísticas podem ser facilmente modificadas para lidar com a nova situação.

Do atrás dito pode concluir-se que, devido à complexidade dos espaços combinatórios, os métodos exactos só em casos raros se constituem como alternativa às heurísticas. Além disso, mesmo em casos onde é possível utilizar os métodos exactos, é muitas vezes necessário providenciar heurísticas poderosas que resolvam parte do problema (lembre-se a heurística necessária para disponibilizar bons limites superiores na abordagem *branch & bound*). Assim, os algoritmos aproximativos são a forma factível de obtenção de soluções aproximadas óptimas a um custo computacional admissível.

Nas últimas décadas, foram feitos enormes progressos no desenvolvimento de heurísticas cujo objectivo é encontrar soluções de alta qualidade num tempo o mais curto possível. As heurísticas para a solução de problemas de carácter combinatório podem ser divididas em algoritmos específicos de problemas e métodos independentes do problema. Alguns exemplos de técnicas modernas independentes do problema são algoritmos de pesquisa na vizinhança, como pesquisa local, pesquisa tabu, ou *simulated annealing*, e métodos inspirados biologicamente, tal como algoritmos evolucionários, colónia de formigas ou redes neuronais. Existem também métodos híbridos que combinam duas ou mais estratégias de procura, por exemplo, algoritmos miméticos que são híbridos de métodos de procura na vizinhança e algoritmos evolucionários. Estes métodos podem ser englobados no que pode ser chamado de "heurísticas retiradas da natureza", uma área que utiliza algumas analogias com sistemas naturais e sociais [Schwefel & Männer, 1990], [Männer & Manderick,

1992]. A metáfora natural é considerada aqui um esquema conceptual que é derivado da física, biologia ou ciências sociais. Quanto às heurísticas, elas são obtidas:

- utilizando um certo número de tentativas repetidas;
- empregando um ou mais agentes (*hill climbers*, partículas, genomas, formigas, etc.);
- operando com mecanismos de competição ou cooperação (no caso de utilização de agentes múltiplos);
- intercalando procedimentos de auto-modificação dos parâmetros heurísticos ou da representação do problema.

A biologia, na natureza, tem dois grandes motores: a selecção, que premeia os indivíduos mais fortes e penaliza os mais fracos e a mutação, que introduz elementos de aleatoriedade e permite o aparecimento de novos indivíduos diferentes. Nas heurísticas inspiradas na biologia tem-se algo semelhante: a selecção é a ideia básica da optimização e a mutação é a ideia base da procura não determinista.

As heurísticas retiradas da natureza, na generalidade, reúnem algumas características, nomeadamente:

- modelam (de perto) um fenómeno existente na natureza;
- são não deterministas;
- apresentam muitas vezes uma estrutura paralela implícita (utilizam múltiplos agentes);
- são adaptativas.

Às heurísticas genéricas atrás enunciadas, poderão acrescentar-se heurísticas específicas, como: técnicas *greedy*, pesquisa local, pesquisa local aleatória só aceitando melhores soluções, aplicação de uma regra não determinista de aceitação de movimentos não melhorativos e utilização de informação relativa ao estado dos agentes, implementando uma memória no sistema.

Combinando as meta-heurísticas e as heurísticas específicas, poderá obter-se um conjunto de algoritmos diversificado, quer nas suas versões "puras", quer em toda uma panóplia de hibridações imagináveis. No entanto, os algoritmos apresentam um conjunto de características que podem ser utilizadas para os classificar. Em [Corloni et al. 1996] são propostas as características apresentadas na Tabela 4.1 para uma classificação dos algoritmos.

Tabela 4.1. Características dos algoritmos de optimização aproximativos, tendentes à sua classificação.

Código	Característica	Código	Característica
A1	Construtivo	A2	Não construtivo
B1	Espaço estruturado	B2	Espaço não estruturado
C1	Solução única	C2	População de soluções
D1	Sem memória	D2	Com memória

Ainda segundo [Corloni et al. 1996], considerando as três primeiras características e incluindo mais algumas heurísticas, ter-se-á a classificação dos algoritmos de optimização mostrada na Tabela 4.2.

Dois aspectos principais têm de ser pesados na concepção dos algoritmos:

- O grau de focalização (*exploitation*), ou seja, o grau do esforço dirigido à pesquisa local na presente região do espaço de pesquisa (se uma região é promissora, há que procurar mais cuidadosamente).
- O grau de exploração (*exploration*), traduzido no grau de esforço gasto na procura em regiões distantes do espaço (por vezes há vantagens na escolha de uma solução numa região distante do espaço e/ou aceitação de uma solução pior, já que se ganha a possibilidade de descobrir soluções novas, potencialmente melhores).

Estas duas possibilidades são conflituantes: um bom balancear entre elas é muito importante e deve ser cuidadosamente ajustado em cada algoritmo.

Outro balanço que deve ser considerado é entre esforço (por exemplo número de iterações) e eficácia (por exemplo valor da solução final). De alguma maneira, a concepção de um bom método heurístico é um problema multi-atributo com dois objectivos conflituantes: esforço e eficácia.

Há, também, uma outra questão, alvo de muita investigação. Em alguns algoritmos há um parâmetro, referido como controlo, aprendizagem ou parâmetro de equilíbrio, que varia lentamente, com o objectivo de evitar óptimos locais e permitir assim a exploração do espaço. Quão mais lenta for a sua variação, maior será a probabilidade de ser obtido um óptimo global como solução final.

Tabela 4.2. Classificação dos algoritmos de optimização.

	B1-C1	B1-C2	B2-C1	B2-C2
A1	Redes Neuronais	Início Múltiplo	*Greedy* Aleatório	Sistemas de Colónia de Formigas
A2	Máquina de Boltzmann	Algoritmos Genéticos Estratégias Evolucionárias Enxame de Partículas	*Simulated annealing* Pesquisa Tabu	*Sampling and Clustering*

4.2 Heurísticas de Construção e de Melhoramento

Neste trabalho, vai-se limitar o âmbito dos algoritmos aproximativos a uma classificação bastante mais restrita do que a apresentada na Tabela 4.1. Assim, ir-se-á apenas utilizar a característica A, dividindo as heurísticas de solução aproximada como heurísticas de construção ou de melhoramento. Estes dois tipos de métodos são significativamente diferentes, já que os primeiros trabalham em soluções parciais, começando com uma solução vazia, tentando depois estendê-la da melhor forma possível, para gerar soluções completas para o problema, enquanto os métodos de melhoramento se movem no espaço de procura de soluções completas, tomando como entrada uma solução possível, e tentam encontrar melhores soluções através de transformações inteligentes passo-a-passo. Ambos os tipos de heurísticas podem ser implementadas eficientemente e são, muitas vezes, capazes de produzir soluções quase óptimas em problemas de optimização de carácter combinatório.

Há um grande número de abordagens para os dois tipos de heurísticas ora mencionadas. Os algoritmos de construção incluem vários tipos de heurísticas altamente dependentes dos problemas. Por exemplo, para o TSP há a heurística do vizinho mais próximo, a heurística da inserção e adição, a heurística *greedy*, a heurística *saving* [Clark & Wright, 1964] e heurísticas baseadas em *spanning trees* (algoritmos Boruvka, Prim e Kruskal). Nesta classe, e no domínio dos algoritmos baseados na vida, há a salientar a optimização por colónia de formigas. Exemplos de heurísticas de melhoramento são algoritmos baseados em pesquisa na vizinhança, tais como: pesquisa local, *simulated annealing*, aceitação de limiar (*threshold accepting*) e pesquisa tabu (*tabu search*), que podem ser aplicados a vários problemas. Também nesta classe, e no domínio dos algoritmos baseados em sistemas biológicos e sociais, têm-se os algoritmos evolucionários (salientando, dentre estes, os algoritmos genéticos) e os algoritmos de optimização por enxame de partículas.

De todos os algoritmos mencionados vão descrever-se, já nas próximas secções, três deles, pois que utilizados nos trabalhos descritos neste capítulo: heurísticas de construção *greedy*, heurísticas de melhoramento genéticas e por enxame de partículas. Outras heurísticas de melhoramento, baseadas em pesquisa de vizinhança, serão descritas em capítulos posteriores, sempre que tal for oportuno.

4.3 Algoritmos de Construção Greedy

Considerado um problema, os algoritmos de construção criam soluções de uma forma incremental. Começam com uma solução inicial vazia e, iterativamente, vão adicionando componentes definidos oportunamente para a solução sem voltar ou olhar para trás, até que uma solução completa seja obtida. No caso mais simples, os componentes da solução podem ser adicionados de forma aleatória. Mas se for tida em conta na selecção dos sucessivos componentes da solução uma heurística capaz de estimar o benefício "míope" (a razão deste epíteto será percebida mais à frente), provavelmente melhores resultados serão obtidos e ter-se-á uma *heurística construtiva greedy*. Uma heurística deste tipo vai adicionar, em cada passo, o componente-solução que consiga o benefício máximo, medido através de alguma informação heurística. Esta informação é usada para criar uma medida que é altamente dependente do problema. Como pode ver-se na descrição formal da heurística de construção *greedy* mostrada em Algoritmo 4.1, começa-se com uma solução vazia para o problema (*sp*=vazio). Segue-se o procedimento iterativo executado enquanto a solução não esteja completa, onde, em cada iteração, é adicionado a *sp* o novo componente *c* da solução, seleccionado pelo melhor valor estimado para a informação heurística utilizada, no caso, a função *ComponenteGreedy*(s_p), que devolve o componente-solução de maior benefício. Assim, a decisão em cada passo depende das decisões tomadas até aí – os efeitos de decisões futuras são desconhecidos: as escolhas *greedy* podem ser vistas como regras de decisão locais. A escolha *greedy* é efectuada de forma a que o valor da função objectivo esperada seja maximizada em cada passo.

As soluções obtidas pela utilização de algoritmos *greedy* são, em regra, melhores do que as obtidas por selecção aleatória de componentes-solução. Contudo, os algoritmos de construção *greedy* apresentam uma clara desvantagem: a produção de um número muito limitado de soluções.

Porém, a principal limitação dos algoritmos de construção *greedy* consiste no facto de as decisões *greedy* tomadas em passos iniciais do processo construtivo constrangerem fortemente as possibilidades em fases posteriores, muitas vezes determinando movimentos muito pobres nas fases finais do processo de construção da solução, o que, usualmente, leva a soluções sub-óptimas.

```
Heurística Construtiva Greedy
Begin
    s_p ← { }
    while s_p ≠ solução completa do:
            c ← ComponenteGreedy( s_p )
            s_p ← s_p ∪ e
    end
    Return s_p
End
```

Algoritmo 4.1. Algoritmo simplificado da heurística construtiva *greedy*.

Como se vê, até pelo tamanho da descrição formal do algoritmo, esta heurística é bastante simples, havendo que lidar com duas questões: a primeira (e fulcral) é a definição da função "*ComponenteGreedy*" que devolve o próximo componente a ser seleccionado para integrar a solução; a outra, é a forma de definir o limite de execução do ciclo.

Um exemplo do funcionamento desta heurística será, porventura, mais claro. Nesta discussão prévia, tem-se tratado do exemplo clássico do TSP simétrico. Veja-se como aplicar o algoritmo a esse problema. No TSP tem-se um grafo pesado completo $G = (N, A)$, onde N é o conjunto de nós, representando as cidades e A o conjunto de arcos que conectam completamente os nós N. A cada arco é atribuído um valor d_{ij}, correspondente à distância do arco $(i, j) \in A$. O problema a resolver consiste em encontrar o circuito de comprimento Hamiltoniano mínimo no grafo, onde o tal circuito é definido como um circuito fechado que visite exactamente uma vez cada um dos $n = |N|$ nós de G. No TSP simétrico, considera-se que a distância de um arco é independente do sentido em que ele é atravessado.

Uma forma directa de resolver o problema é iniciar a viagem numa qualquer cidade e escolher sempre ir para a cidade mais próxima ainda não visitada, repetindo a escolha até chegar à cidade de partida. Este algoritmo é conhecido como a heurística construtiva *greedy* do vizinho mais próximo. A aplicação do algoritmo a casos reais mostra que as soluções fornecidas são, muitas vezes, fortemente sub-óptimas, mas rápidas. De facto, os algoritmos de construção são, tipicamente, os métodos

aproximativos mais rápidos, mas as soluções que eles obtêm são poucas vezes de muito elevada qualidade e não garantem ser óptimos relativamente a pequenas alterações. Os resultados produzidos pelas heurísticas construtivas podem, muitas vezes, melhorar, através da utilização de algoritmos de pesquisa local.

4.4 Algoritmos Genéticos

Os algoritmos genéticos fazem parte do domínio dos denominados algoritmos evolucionários, inspirados pelo poder da evolução natural, de que fazem parte também, por exemplo, as estratégias de evolução e a programação evolucionária. Em qualquer destes algoritmos, tem-se uma população de soluções candidatas que evolui, sujeita a replicação, variação e selecção, os três conceitos fundamentais que, segundo a teoria da evolução natural, proposta por Charles Darwin no seu livro "A Origem das Espécies" [Darwin, 1859], vão determinar a evolução. Além dos três pilares já referidos, Darwin acrescentou mais alguns conceitos e princípios, considerados revolucionários para a sua época. Da sua observação de vários ambientes longamente isolados (ilhas Galápagos), compreendeu a forma como os pilares fundamentais da evolução poderiam operar. Referiu assim que:

1. Novos organismos não podem evoluir somente por replicação, já que os descendentes produzidos seriam cópias idênticas. Devido aos erros no processo de replicação, são introduzidas variações que dão origem ao desenvolvimento gradual de novos organismos.
2. A recombinação sexual é outra forma de variação, sendo, ela mesma, um produto da evolução.
3. A replicação não pode operar infinitamente, já que isso exigiria recursos ilimitados, o que não é o caso, pois que a realidade indesmentível, agora cada vez mais patente, é que os recursos são limitados: até mesmo as estrelas morrem! Assim, os indivíduos da mesma ou de outras espécies têm de competir por esses recursos e só os mais adaptados sobrevivem.
4. Em resumo, a evolução natural implica a adaptação das formas de vida ao seu ambiente, pois que só o mais adaptado tem uma possibilidade de sobreviver o tempo suficiente para se reproduzir: é o "princípio da sobrevivência do mais apto".

Colocadas as questões desta forma, a evolução natural pode ser pensada como um imenso processo de optimização no qual a adaptação é maximizada. Cada indivíduo tem em si informação genética, dita *genótipo*. Quanto aos traços de cada indivíduo (que constituem o seu *fenótipo*), são a manifestação física do genótipo, em relação com os diferentes ambientes em que o indivíduo cresceu,

reflectindo factores genéticos e ambientais. Sempre que um indivíduo se reproduz, o seu genótipo (ou parte dele) é legado à sua descendência. O *genótipo* é assim potencialmente imortal, enquanto cada indivíduo pode ser visto como uma máquina mortal com o objectivo de assegurar a imortalidade da informação genética. Em resumo, utilizando o genótipo como plano de construção dum organismo, é criado o respectivo fenótipo, através do qual a aptidão do genótipo é indirectamente avaliada, enquanto a replicação, combinada com a variação, permitem a respectiva melhoria, gerando genótipos progressivamente mais aptos a criar organismos mais capazes.

4.4.1 Genética Básica

Em termos biológicos, cada organismo contém a informação genética (o seu genoma) em *cromossomas* - não mais do que cadeias de ADN (*Ácido Desoxirribo Nucleico*) - que podem ser conceptualmente divididos em *genes*, blocos funcionais capazes de codificar uma *proteína*, posicionados num local particular do cromossoma, denominado *locus*. Em termos de um modelo muito simplificado, pode pensar-se um gene como o codificante de um determinado traço, como, por exemplo, a cor dos olhos. Os valores possíveis para um gene são denominados alelos. Assim, por exemplo, para as pessoas, há um número finito de possíveis cores de olhos.

O processo de reprodução pode decorrer de duas formas distintas: por simples divisão e replicação dos cromossomas (reprodução assexuada), ou por recombinação sexual. Este último processo permite uma muito maior variabilidade, já que pressupõe a formação de gâmetas[7], o que implica um fenómeno de meiose[8], na qual se dão dois eventos que concorrem para uma grande diversidade: o *crossover* (troca de genes entre dois cromossomas homólogos, um de cada pai) e a disjunção aleatória dos cromossomas (maternos e paternos) para cada gâmeta, que vai permitir a ocorrência, numa uma única célula, de uma combinação única e completamente nova de genes. Em ambas as formas de reprodução podem ocorrer erros: um ou mais alelos de genes são mudados, genes são apagados, ou são inseridos em outros *locus* dos cromossomas. Neste caso, diz-se que ocorreu uma mutação.

Ao pensar-se na quase infinita diversidade de formas de vida, custa a perceber como mecanismos tão simples a possam ter originado. De facto, há ainda a referir que um gene pode afectar

[7] Célula sexual que possui metade dos cromossomas das outras células do indivíduo, o que permite que se mantenha uma guarnição cromossómica constante na espécie.
[8] Processo de divisão celular no qual ocorre redução dos cromossomas para metade, originando os gâmetas.

simultaneamente mais do que um traço do fenótipo, algo conhecido como *pleiotropia*; por outro lado, uma só característica do fenótipo pode ser determinada por interacção simultânea de muito genes, efeito conhecido como *poligenia*. O resultado da interacção entre genes que provoca a variação genética que ocorre durante a mutação e/ou recombinação é, assim, muito complexa e dificilmente previsível, fenómeno conhecido por *epistasis*.

4.4.2 Genética Computacional

Depois desta breve alusão aos mecanismos genéticos que criaram a imensa diversidade de seres vivos, e percebida a vida como o produto do imenso método de optimização, nada mais natural do que pensar em utilizar estes mecanismos numa simulação computacional e conceber uma heurística de optimização. Realmente, Holland, nos anos 60, procurava estudar os fenómenos de adaptação, tal como ocorriam na natureza, e desenvolver formas de importar esses mecanismos para os sistemas computacionais. Assim concebeu os algoritmos genéticos (AG) [Holland, 1992], [Mitchel, 1996], em 1975, como uma abstracção da evolução biológica. No modelo proposto, os cromossomas ou genomas eram cadeias de bits – alelos um e zero - sobre os quais eram aplicados operadores de inspiração genética, como *crossover*, mutação e inversão, seguidos por algum tipo de selecção natural. Na Figura 4.1 é mostrado um esquema elucidativo acerca do funcionamento de um AG.

Num AG, em primeiro lugar, uma população inicial é criada de forma aleatória, normalmente não considerando questões de adaptação, tendo, no entanto, de respeitar constrangimentos, se o processo de optimização os incluir. Como se mostra na figura, cada indivíduo representa uma solução completa para o problema a solucionar, daí que se esteja em presença de uma heurística de melhoramento. Então, no ciclo principal, uma população temporária é seleccionada a partir da população corrente, utilizando uma estratégia de selecção. Depois, os operadores genéticos mutação e/ou recombinação e inversão são aplicados a alguns ou todos os membros (indivíduos) da população temporária.

No final de cada ciclo (correspondente a uma geração), a nova população substitui a antiga e é avaliada através da função de aptidão. O ciclo vai ser repetido sucessivamente até que um critério de finalização seja satisfeito (um limite temporal ou um número de gerações previamente estabelecido).

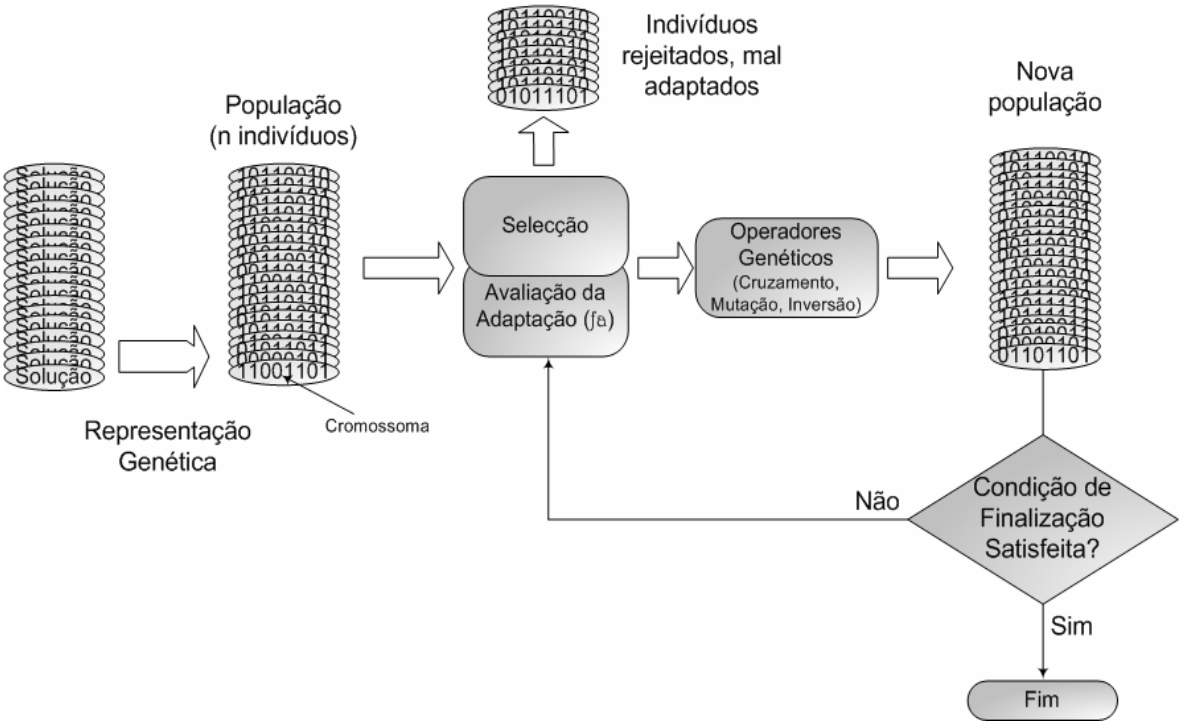


Figura 4.1. Esquema funcional de um algoritmo genético.

Em Algoritmo 4.2, mostra-se a descrição formal de um AG genérico que pode assumir variações múltiplas, já que há um conjunto de parâmetros que, assumindo valores e formas diversas, vão gerar essa diversidade. Além disso, é importante referir, desde já, que o próprio algoritmo conheceu muitas alterações ao esquema base, uma das quais será descrita e utilizada no próximo capítulo.

Como se vê na Figura 4.1, a avaliação de cada indivíduo é a parte central de um algoritmo genético. A função de aptidão é normalmente a função objectivo do problema a ser solucionado pelo algoritmo genético. Esta é uma das duas questões a serem resolvidas para aplicar um algoritmo genético a um problema, e tem de ser definida individualmente. A outra prende-se com a forma de codificar o problema no genoma, para permitir depois o processo de evolução.

```
Procedimento AGP; // algoritmo genético padrão
1. Inicialização: inicializar uma população P de indivíduos de forma aleatória
2. Avaliar a População Inicial:
   Para Cada Indivíduo I Em P, Fazer:
    Aptidão(I)←f(I); // calcular a aptidão de cada indivíduo da população incial
   FimPara
3. Ciclo correspondente a uma geração:
   P'←{ }; // limpa nova população
   Enquanto |P'|<=|P|, Fazer: // enquanto não for gerada uma população com
    // número de indivíduos P
    // Aplicar Operador Cruzamento:
    {Pais} ← selecção (P); // seleccionar 2 pais em P de acordo com um esquema
       // de selecção para reprodução
    {Filhos}←cruzamento ({Pais}); // gerar 2 filhos por cruzamento dos pais
    P' ← P' ∪ {Filhos};
    // Aplicar operador mutação
    Mutar(selecção (P')); // aplicar operador mutação num indivíduo selec. em P'
   FimEnquanto
   P ← P'; // substituir a população pela nova população
   Para Cada Indivíduo I Em P, Fazer:
    Aptidão(I)←f(I); // calcular a aptidão de cada indivíduo da nova população
   FimPara
4. Ciclo das sucessivas iterações :
   Repetir 3. Até que um dado critério de convergência seja verificado
    // normalmente um determinado número de iterações pré-estabelecido
5. Retornar Resultado:
   Return melhor(P); // retornar o melhor genoma da população
```

Algoritmo 4.2. Descrição formal genérica de um AG padrão.

O AG, tal como mostrado em Algoritmo 4.2, é adaptado a optimizações de carácter combinatório onde não existam constrangimentos. Em casos onde estes se apliquem, há que introduzir um passo adicional. As soluções produzidas podem simplesmente não satisfazer o(s) constrangimento(s), pelo que são inválidas. Vários mecanismos podem ser aplicados:

- inserir um algoritmo de reparação que transforme soluções não factíveis em possíveis, normalmente aplicado depois do operador genético (geração dos novos indivíduos) ou quando toda a nova população foi gerada;
- não reparar a solução faltosa, mas aplicar uma penalização (um termo adicional que diminua fortemente a sua adaptação), tornando o indivíduo respectivo bem menos capaz de gerar descendência (já que com menor adaptação), de forma a que o AG fique focado na região do espaço de soluções factíveis;
- simplesmente eliminar as soluções inválidas;
- actuar sobre os próprios operadores genéticos de forma a que só soluções válidas sejam produzidas.

Em resumo, a heurística genética é de simples implementação, sendo aplicável a problemas de optimização nas mais diversas áreas, bastando ter uma forma de avaliar a qualidade de cada solução e de encontrar uma forma de codificar o problema no genoma do indivíduo. A sua simplicidade implica baixa complexidade e, assim, rapidez de execução. Para muitos problemas, o cálculo da aptidão domina o tempo de execução do algoritmo, pelo que, sempre que disponíveis, devem ser exploradas características específicas do problema, para diminuição da respectiva complexidade. Outra alternativa será a utilização de algoritmos paralelos, permitindo a utilização de uma rede de *workstations*.

4.4.3 Operadores Genéticos

A nova população é gerada por actuação dos operadores genéticos. O operador *crossover* simula a reprodução sexuada, onde um novo genoma é produzido por combinação de pedaços de cromossomas de dois pais; já os operadores mutação e inversão simulam os "erros" na cópia dos cromossomas, sendo de alcance mais limitado, normalmente constituindo uma fonte de diversidade adicional de forma a prevenir fenómenos de convergência prematura.

O operador *crossover* vai dividir o genoma de cada um dos pais $A, B \in \{0,1\}^n$ (uma cadeia de *n* bits) em pedaços, utilizando-os depois em combinações para gerar a descendência. Assim, os pedaços podem ter vários níveis de granularidade, de acordo com o número de pontos p de divisão utilizados, com valores típicos de $p = \{1,2\}$. O(s) ponto(s) de divisão podem ter uma localização fixa l ou, mais comummente, gerar a sua posição na cadeia de bits de uma forma aleatória $l = rnd\{1,2,...,n\}$.

Outro esquema de cruzamento é a utilização de uma máscara, que indica, para cada bit do genoma, a qual dos pais vai transmitir o valor nesse *locus*.

Crossover **de um ponto**

Neste operador [Holland, 1992], é utilizado um ponto l para efectuar a divisão do genoma (cadeia de *n* bits) dos pais em dois pedaços, gerando-se A_1 e A_2 a partir de A e B_1 e B_2 a partir de B. Depois, o pedaço inicial de um pai é combinado com o segundo pedaço do outro pai ($A' \leftarrow A_1 \cup B_2$) gerando-se um dos descendentes (A') e reciprocamente ($B' \leftarrow A_2 \cup B_1$), gerando B'. A Figura 4.2 mostra a sua actuação na geração de duas soluções A' e B'.

$$A_i' \leftarrow \begin{Bmatrix} A_i \ se\ i \le l \\ B_i \ se\ i > l \end{Bmatrix} \text{ e } B_i' \leftarrow \begin{Bmatrix} B_i \ se\ i \le l \\ A_i \ se\ i > l \end{Bmatrix}. \text{ Exemplo: } \begin{matrix} A = 0110 \mid 1001 \\ B = 1010 \mid 1100 \end{matrix} \rightarrow \begin{matrix} A' = 0110 \mid 1100 \\ B' = 1001 \mid 1010 \end{matrix}$$

Figura 4.2. Exemplo de *crossover* de um ponto.

Crossover **de dois pontos**

Neste caso [Holland, 1992], não se tem um ponto de divisão, mas dois $l_1 < l_2$, criando-se três pedaços de genoma de cada um dos pais, posteriormente recombinados. A Figura 4.3 mostra o esquema e um exemplo.

$$A_i' \leftarrow \begin{Bmatrix} B_i \ se\ l_1 \le i \le l_2 \\ A_i \notin l_1 \le i \le l_2 \end{Bmatrix} \text{ e } B_i' \leftarrow \begin{Bmatrix} A_i \ se\ l_1 \le i \le l_2 \\ B_i \notin l_1 \le i \le l_2 \end{Bmatrix}.$$

$$\text{Exemplo: } \begin{matrix} A = 0 \mid 1101 \mid 001 \\ B = 1 \mid 0101 \mid 100 \end{matrix} \rightarrow \begin{matrix} A' = 0 \mid 0101 \mid 001 \\ B' = 1 \mid 1101 \mid 100 \end{matrix}$$

Figura 4.3. Exemplo de *crossover* de dois pontos.

Crossover uniforme

No *k-crossover*, tem-se uma generalização do *crossover* para *k* pontos de divisão. O *crossover* uniforme [Syswerda, 1989] é um *k-crossover* onde *k=n*, ou seja a divisão é efectuada bit a bit da cadeia constituinte do genoma, mas é especificada uma máscara M, uma cadeia de bits de tamanho n, cujo valor, para cada bit M_i, indica qual dos pais a utilizar para copiar o bit respectivo. A Figura 4.4 mostra o respectivo esquema e exemplo.

$$A_i' \leftarrow \begin{Bmatrix} B_i \ se \ M_i = 1 \\ A_i \ se \ M_i = 0 \end{Bmatrix} \text{ e } B_i' \leftarrow \begin{Bmatrix} A_i \ se \ M_i = 1 \\ B_i \ se \ M_i = 0 \end{Bmatrix}.$$

Exemplo: $\begin{matrix} A = 01101001 \\ B = 10101100 \end{matrix} \quad dado \quad M = 10110010 \quad \rightarrow \quad \begin{matrix} A' = 11110101 \\ B' = 00101100 \end{matrix}$

Figura 4.4. Exemplo de *crossover* uniforme.

Sendo *k-crossover* uma generalização do *crossover*, o *crossover* uniforme também o é, sendo assim o *crossover* de um e dois pontos um caso especial do *crossover* uniforme. Para os casos considerados nos exemplos da Figura 4.2 e Figura 4.3, ter-se-ia um $M = 00001111$ e $M = 01111000$, respectivamente.

Mutação tipo comutação de bit

Trata-se do operador para executar a mutação [Goldberg, 1989] em genomas de cadeias de bits. Actua por simples comutação de alguns bits na cadeia que constitui o genoma. A operação mostra-se na Figura 4.5.

$$A = 1111\underline{\ }0101 \quad \rightarrow \quad A' = 1110\underline{0}0101$$

Figura 4.5. Mutação por comutação de bit.

Este tipo de mutação pode executar-se de duas maneiras principais: 1) especificar-se uma probabilidade de mutação de um qualquer bit do genoma; 2) pré-definir o número de bits a aplicar a mutação (taxa de mutação) e seleccionar o *locus* no genoma de forma aleatória.

Inversão

O operador genético inversão [Goldberg, 1989] é um alternativo à mutação. São escolhidos dois pontos no cromossoma. A sub-cadeia entre esses dois pontos é simplesmente invertida. O exemplo da Figura 4.6 é ilustrativo da operação.

$$A = 11|110|101 \quad \rightarrow \quad A' = 11|011|101$$

Figura 4.6. Operador inversão.

4.4.4 Estratégias de Selecção

Antes de apresentar os esquemas mais comuns de selecção, de entre os muitos descritos na literatura, importa discutir, ainda que brevemente, o seu efeito.

Foi atrás referido que os recursos, não sendo ilimitados, colocam um imperativo de competição que leva à evolução. Assim, é expectável que os bons espécimes tenham uma maior probabilidade de passar o seu legado genético à nova geração e, assim, a aptidão global da espécie tenda a melhorar. Quaisquer boas características (adquiridas por um cruzamento ou mutação afortunado), ou simplesmente porque o ambiente sofreu alterações, devem, desejavelmente, ser preservadas na descendência. Mas a diversidade genética é também importante. Se uma população for constituída por clones, a evolução quase pára. O operador *crossover* torna-se inoperante e os operadores mutação e inversão são de muito menor acção. A selecção deve focar o processo de procura de soluções em zonas promissoras do espaço de soluções, mas não restringir o seu domínio. Desta forma, o processo evolutivo depende fortemente da relação de forças entre a intensidade selectiva e a diversidade genética. Uma elevada intensidade selectiva vai lesar, quase certamente, a diversidade genética (já que material genético sub-óptimo, mas que pode ajudar a encontrar o mínimo global, é

removido muito rapidamente) e ocorre um fenómeno de convergência prematura. Todavia, a pressão selectiva não pode ser arbitrariamente baixa: a exploração seria favorecida, mas isso levaria a que a procura em áreas promissoras fosse abandonada, e o AG, simplesmente, não convergiria. Quando este mecanismo evolucionário foi transposto para o computador, a relação de forças referida atrás foi passada para o mecanismo de selecção, ou qualquer outra forma de preservação da diversidade genética, sem favorecer a focalização em detrimento da exploração e vice-versa (conforme terminologia e problemática discutida no final da secção 4.1). De facto, pode ter-se uma população e um mecanismo de selecção para a reprodução e confiar que a interacção entre os operadores genéticos preservem a diversidade, mas pode actuar-se também directamente sobre a própria população e usar um mecanismo de selecção para substituição. Este último mecanismo pode incorporar princípios de preservação, por exemplo, evitar a morte de indivíduos raros.

De entre os mecanismos de selecção do primeiro tipo, podem destacar-se, a selecção proporcional, a selecção baseada no ranking e a selecção competição, cujo estudo analítico é descrito em [Blickle & Thiele, 1996]. Quanto aos mecanismos do segundo grupo, podem destacar-se a substituição geracional e a selecção de estado estacionário.

Selecção Proporcional

Na selecção proporcional, a probabilidade de seleccionar um indivíduo s_i é dada por: $p(s_i) = \frac{f(s_i)}{\sum_{s_j \in P} f(s_j)}$, ou seja, a probabilidade de selecção de um indivíduo é directamente proporcional à sua aptidão e inversamente proporcional à soma das aptidões dos restantes indivíduos.

Uma forma de implementar este esquema de selecção é simular uma roleta [Goldberg, 1989], onde o tamanho de cada ranhura – a "*slot*", coroa circular onde pode parar a bola - da roleta é proporcional à probabilidade de cada indivíduo ser seleccionado, calculada pela razão entre a sua aptidão e o somatório da aptidão dos indivíduos da população corrente (uma aproximação à equação de cima, para que a soma de todas as probabilidades seja um). Somando a probabilidade do indivíduo corrente à probabilidade acumulada dos anteriores, cria-se a roda da roleta com as correspondentes ranhuras. Por exemplo, se a probabilidade dos indivíduos 1 e 2 for 0.24 e 0.12, respectivamente, então a ranhura 1 ficará no intervalo [0-0.24] e a ranhura 2 no intervalo [0.25-0.36]. A última ranhura deverá ter o valor final de 1. Em resumo, cada individuo na população ocupará uma ranhura de tamanho

proporcional à sua aptidão. Quando a bola gira na roleta, como a um indivíduo mais apto corresponderá uma ranhura mais larga, a probabilidade da bola parar na sua ranhura será maior.

Uma desvantagem apontada a este tipo de selecção é que, com a diminuição da variância dos valores de aptidão dos indivíduos de uma população, a selecção vai-se tornando aleatória.

Selecção Truncamento

Na selecção truncamento [Mühlenbein & Schlierkamp-Voosen, 1994], só uma fracção T dos melhores indivíduos pode ser seleccionada com igual probabilidade. Variando T, a intensidade da selecção pode ser alterada.

Selecção *Ranking*

A selecção *ranking* [Baker, 1985] procura responder à desvantagem referida atrás para a selecção proporcional, mantendo a pressão de selecção constante, independentemente da variância dos valores da aptidão dos indivíduos da população. A selecção *ranking* tem duas variações: *ranking* linear (*RL*) e *ranking* exponencial (*RE*). Na primeira, os indivíduos são ordenados de acordo com o seu valor de aptidão. Então, o nível N é atribuído ao melhor indivíduo e o nível 1 ao pior. A probabilidade de selecção é então atribuída de forma linear aos indivíduos, de acordo com a sua posição na lista ordenada por nível. Como o seu nome mostra, o *ranking* exponencial difere do *ranking* linear de forma a que a probabilidade dos indivíduos na lista tenha um peso exponencial, sendo r o parâmetro do método (a base do expoente). À medida que r se aproxima de 1, menor será a "exponencialidade" do método; se $r = 1$, a pressão de selecção é nula (todos os indivíduos terão a mesma probabilidade de serem seleccionados).

Selecção Torneio

Na selecção torneio [Blickle & Thiele, 1995], *t* indivíduos (o tamanho do torneio) da população são seleccionados aleatoriamente (independentemente da respectiva aptidão). O melhor, de entre os *t* seleccionados, é o escolhido. Variando *t*, a intensidade da selecção pode ser alterada. Se *t=2*, ter-se-á um torneio binário.

Selecção Competição

A selecção competição é parecida com a selecção torneio binário, mas em vez de ser seleccionado o melhor indivíduo de entre os dois retirados aleatoriamente da população, é introduzida uma probabilidade no processo. Assim, com uma probabilidade *P*, será escolhido o melhor indivíduo e, com uma probabilidade *1-P*, o de menor aptidão será escolhido. Deste modo, o valor de P permite modificar a intensidade da selecção competição.

Selecção por Aptidão Uniforme

Todos os métodos de selecção descritos atrás podem ser já considerados como estandardizados. A selecção de aptidão uniforme [Hutter, 2002] é bastante diversa, pois que parece encerrar, em si mesma, um contra-senso. Como se faz uma selecção com aptidão uniforme? Se a aptidão for uniforme, qualquer indivíduo terá a mesma probabilidade de ser seleccionado e a pressão selectiva será nula. Na realidade, irá perceber-se que isto não é assim.

O método pode ser descrito da seguinte forma: se o valor menor/maior da aptidão dos indivíduos da população corrente P for $f_{\min} / f_{\max}$, vai seleccionar-se um valor de aptidão f uniformemente no intervalo $\left[f_{\min} / f_{\max}\right]$. Depois, o indivíduo $I \in P$ com o valor de aptidão mais próximo de f é seleccionado. Este método preserva a diversidade, evitando completamente a convergência, abandonando-a em absoluto. Mas, como se referiu no início, à primeira vista, isto parece significar que não há qualquer pressão selectiva, e o método não deverá simplesmente funcionar. Nada mais ilusório: como o método selecciona de forma uniforme dentro dos níveis de aptidão, os indivíduos em níveis de aptidão com baixas populações são efectivamente favorecidos. A probabilidade de seleccionar um indivíduo específico com aptidão f é inversamente proporcional à distribuição da sua aptidão. Como numa distribuição normal os indivíduos com maior/menor aptidão têm mais baixas distribuições, terão então uma maior probabilidade de serem seleccionados, havendo então uma pressão selectiva efectiva.

Selecção Geracional

Trata-se de um dos métodos de substituição da população (trata-se aqui de "matar" indivíduos). É o esquema mais utilizado em AGs ditos geracionais (como aquele mostrado no Algoritmo 4.2). Uma vez gerada a nova população, esta substitui completamente a antiga.

Selecção Estado Estacionário

Na selecção tipo estado estacionário [Syswerda, 1991], o número de descendentes produzidos é menor do que o número de pais. Desta forma, é necessária uma estratégia para decidir que pais vão ser substituídos. Múltiplas variantes existem, tais como substituição do pior ou substituição do mais velho [Hancock, 1994].

Este esquema de substituição é necessário, por exemplo, em heurísticas genéticas ditas de estado estacionário (daí o nome deste tipo de selecção). Veja-se o porquê do nome. Suponha-se que, numa população P de indivíduos, dois deles são seleccionados, não importa por que método de selecção, para reprodução, procedendo-se ao respectivo cruzamento. Os dois descendentes gerados, no caso de AG de estado estacionário, devem ser imediatamente incluídos na população P, havendo assim que aplicar a selecção de estado estacionário, eliminando dois indivíduos para inserir os recém-gerados. Esta inclusão imediata dos novos indivíduos permite que sejam considerados imediatamente no processo evolutivo, ou seja, o processo decorre sem sobressaltos, ao contrário do que ocorre no esquema geracional, onde, num momento, toda a população é substituída de uma única vez. Tem-se, assim, uma população de conteúdo menos dinâmico, sem movimentos bruscos, daí o nome estacionário. Será talvez de mencionar um esquema de selecção mais recente que funciona com um AG estacionário, denominado remoção por aptidão uniforme [Legg & Hutter, 2005].

4.5 Optimização por Enxame de Partículas

A optimização por enxame de partículas (OEP) advém de outra heurística inspirada na vida, neste caso fruto do estudo de comportamentos de indivíduos enquanto constituintes de um grupo. Faz parte das heurísticas melhorativas (tal como os AG) pois que cada interveniente no processo de busca de soluções constitui, em cada iteração e logo desde a sua geração, uma solução completa.

A expressão "inteligência de enxame" foi forjada no final dos anos oitenta do século passado para referir os sistemas robóticos celulares, nos quais uma colecção de agentes simples localizados num ambiente, interactuam de acordo com regras locais. Foi definida em [White & Pagurek, 1998], como "A propriedade dos sistemas de agentes não inteligentes, com capacidades individuais limitadas, de exibirem, colectivamente, comportamentos inteligentes".

O conceito de enxame de partículas teve origem numa simulação de um sistema social simplificado. O objectivo original, tal como era visto pelos seus autores, era a simulação gráfica da coreografia

graciosa, mas imprevisível, de um bando de aves. As simulações iniciais foram sendo modificadas para incorporar a correspondência da velocidade do vizinho mais próximo, continuando com o eliminar de variáveis auxiliares e incorporação de pesquisa multidimensional e aceleração por distância. Os autores acabaram, no entanto, por perceber que o modelo conceptual era, de facto, um optimizador. Através de um processo de tentativas e erros, vários parâmetros alheios à optimização foram eliminados do algoritmo, sendo o resultado final uma implementação muito simples e original, proposta em [Eberhart & Kennedy 1995] e [Kennedy & Eberhart, 1995].

Tal como os algoritmos genéticos, trata-se de uma técnica baseada na evolução de uma população (embora aqui a evolução não deva ser entendida como evolução biológica, mas evolução espacial, um deambular no espaço de soluções), sendo o sistema iniciado com uma população de soluções aleatórias. Mas se há algumas semelhanças na forma, diferem em muitas outras questões, especialmente na profunda separação da forma como são procuradas soluções sucessivamente melhores: o AG é baseado em competição, na luta pela sobrevivência, algo embutido nos seres biológicos mais elementares; já a OEP assume uma postura de mais alto nível, surgindo a aprendizagem e partilha de conhecimento como o paradigma capaz de conduzir a melhores soluções. Difere também na própria forma como é representado o problema (e conhecimento), na forma como se processa a procura das soluções e nos próprios agentes de procura. Assim, se nos AGs a representação do problema é feita através dum genoma, na OEP tem-se uma localização no espaço das soluções; se a procura de soluções era feita nos AGs através dos operadores genéticos e selecção, na OEP o agente movimenta-se no espaço multi-dimensional; por último, os indivíduos dos AGs, carregando uma herança genética, surgem agora como partículas, localizadas numa posição do espaço multidimensional, dotadas de uma velocidade e capazes de acelerar, procurando melhores posições. Esta é a responsável pela alteração da posição da partícula, de forma a explorar o espaço de todas as soluções possíveis, e não, como nos algoritmos genéticos, que utilizam a transmissão à próxima geração, através de passagem directa ou reprodução das soluções existentes. À medida que as partículas se movem pelo espaço, elas testam diversas localizações. Cada localização tem um valor de adaptação de acordo com o seu nível, avaliado em termos da satisfação dos objectivos.

É assumido que as partículas são sociais por natureza e, portanto, capazes de interagir com outras numa determinada vizinhança. Cada partícula tem à sua disposição dois tipos de informação: a primeira é relacionada com as suas experiências passadas (conhecida como conhecimento individual) e a outra relacionada com o conhecimento acerca dos sucessos havidos na sua vizinhança (referida como transmissão cultural).

As partículas (como indivíduos sociais) tendem a ser influenciadas pelos seus próprios sucessos e também pelos sucessos de outras, com as quais interagem. Assim, cada partícula mantém o rasto das suas coordenadas no espaço do problema que estão associadas à melhor solução que foi atingida até ao momento, sendo guardado também o valor da adaptação (*pbest*). Um outro valor relativo à "melhor" solução, que é mantido pela versão global do optimizador do enxame de partículas, é o "melhor" valor geral e a sua localização (*gbest*), conseguido até então por uma qualquer partícula da população. O conceito de optimização por enxame de partículas consiste, em cada passo, na alteração da velocidade (por aceleração) de cada partícula, em direcção às localizações *pbest* e *gbest*. A aceleração é pesada por um termo aleatório, diferentemente gerado para a aceleração em direcção à localização *pbest* e *gbest*.

4.5.1 Optimização por Enxame de Partículas - Versão Contínua

Tal como foi proposto originalmente, a OEP tinha como principal característica a natureza contínua do espaço de pesquisa. O processo de implementação da versão global da OEP surge em [Eberhart & Shi, 2001] e é o seguinte:

1. Iniciar uma população (*array*) de partículas, com posições e velocidades aleatórias relativas a *d* dimensões do espaço do problema.
2. Para cada partícula, calcular a função optimização de adaptação desejada nas *d* variáveis.
3. Comparar o cálculo relativo ao ponto anterior de cada partícula, relativamente a *pbest* (melhor adaptação da própria partícula até ao momento). Se o valor agora calculado for superior do que *pbest*, actualizar este com o valor corrente e o mesmo para a localização de *pbest*, mudando-a para a localização corrente da partícula no espaço *d-dimensional*.
4. O mesmo que em 3), mas não para *pbest*, relativo a *gbest*, e sendo actualizado o valor de *gbest* com o valor actual da adaptação da partícula corrente (aquela cuja adaptação ultrapassou *gbest*), sendo também actualizada a posição do índice da partícula.
5. Alterar a velocidade e posição da partícula de acordo com as equações seguintes:
   a) $v_{id} = v_{id} + c_1*rnd()*(x_{pbest,i} - x_i) + c_2*rnd()* (x_{gbest} - x_i)$
   b) $x_i = x_i + v_i$
   c) A velocidade das partículas em cada dimensão é limitada a $v_{max.}$

em que

x_i é a posição corrente da partícula i,

$x_{pbest,i}$ é a melhor posição atingida pela partícula i,

x_{gbest} é a melhor posição global do enxame,

v_i é a velocidade da partícula i,

v_{max} é o limite máximo de velocidade admissível para as partículas numa dada dimensão;

c_1 e c_2 são constantes relativas ao denominado factor de mola que, originalmente, para a maioria das aplicações, tomava o valor de 2.0.

6. Repetir de 2) a 6) até que um critério seja atingido, normalmente uma suficientemente boa adaptação ou um dado número máximo de iterações.

Das regras e equações definidas atrás, no ponto 5., importa esclarecer, mais aprofundadamente, alguns parâmetros que levaram à introdução de alterações, em particular:

- V_{max} serve de factor de constrangimento à capacidade exploratória global do enxame de partículas. Um valor grande de v_{max} facilita a exploração global, enquanto um valor pequeno encoraja a exploração local. Mas, se for muito grande, as partículas podem voar por cima de boas soluções; caso contrário, podem não explorar de forma suficientemente profunda regiões localmente boas. De facto, podem ficar presas em óptimos locais, impossibilitadas de se afastar destes para atingir uma melhor posição no espaço do problema. Antes de ser utilizado o peso de inércia, era normalmente utilizado um valor entre 10 a 20 % do valor dinâmico de cada variável em cada dimensão. Como se verá abaixo, actualmente é utilizado um valor igual ao intervalo dinâmico de cada variável em cada dimensão.

- As constantes de aceleração c_1 e c_2 representam o peso dos termos de aceleração estocástica que empurram cada partícula em direcção às posições *pbest* e *gbest*. O ajustamento destas constantes altera o nível de "tensão" no sistema, daí o nome de factor de mola. Valores baixos permitem às partículas "vogar" em regiões alvo, antes de serem puxadas para fora, enquanto valores altos resultam em movimentos abruptos em direcção a regiões alvo, ou para fora delas.

- W, o denominado peso de inércia, um novo parâmetro introduzido na equação a), foi proposto em [Shi & Eberhart, 1998a], [Shi & Eberhart, 1998b], cujo conceito foi desenvolvido para melhor controlar a exploração e focalização, ou seja, a possibilidade, numa primeira fase, de empreender uma pesquisa genérica, e depois um aprimorar dos resultados anteriores. Inicialmente, a sua introdução pretendia possibilitar a eliminação de v_{max}. Mas acabou por perceber-se que a sua inclusão permitia trabalhar sempre com v_{max} com o valor

do intervalo dinâmico de cada variável (em cada dimensão), já que o peso de inércia permite depois o seu ajuste dinâmico. Tal como foi originalmente desenvolvido, w era muitas vezes linearmente decrementado de cerca de 0.9 a 0.4 durante a execução do processo. Uma selecção adequada do peso de inércia proporciona um balanceamento entre a exploração global e focalização local, e resulta num menor número de iterações para se chegar a uma solução suficientemente boa.

Dados estes considerandos, as equações acima resultam nos passos seguintes:

$$v_i = w*v_i + c_1r_1(x_{pbest,i} - x_i) + c_2r_2(x_{gbest} - x_i) \quad (1)$$

$$\text{se } (|v_i| > v_{max}) \text{ então } v_i = (v_{max} / |v_i|) * v_i \quad (2)$$

$$x_i = x_i + v_i \quad (3)$$

em que r_1 e r_2 são números aleatórios de valor entre 0 e 1, rnd() nas equações do ponto 5.

4.5.2 Optimização por Enxame de Partículas - Versão Discreta

O algoritmo OEP descrito na secção anterior é adequado à optimização em problemas contínuos. A velocidade e localização de cada partícula podem variar de forma contínua. Contudo, muitos dos problemas, nomeadamente no campo da engenharia, são formulados como optimizações em problemas combinatórios. Uma das soluções poderá ser discretizar o espaço, de forma a que a partícula já só possa estar em determinadas localizações, um pouco à imagem da mecânica quântica, em que o electrão só pode estar em níveis definidos de energia. Já a velocidade pode continuar a assumir valores supondo um espaço contínuo. Este modelo vai já permitir a representação da solução combinatória no espaço discreto, sendo a transição entre localizações assegurada pela velocidade de cada partícula. Será apenas necessária uma função que permita calcular uma nova posição discreta, sabida a posição inicial e a velocidade.

Veja-se este modelo no espaço binário. Cada partícula parece mover-se num hipercubo, mudando um conjunto de vários bits. A velocidade da partícula vai determinar o número de bits alterados em cada iteração, ou seja, a distância de *Hamming* entre a partícula no instante *t* e *t+1*. Uma partícula na qual não ocorram quaisquer mudanças de bit, não se move; pelo contrário, ocorrerá uma mudança máxima ao comutarem-se todas as suas coordenadas binárias. Se este modelo parece descrever bem a questão do espaço discreto binário, o conceito de velocidade nesse espaço pode ter duas interpretações:

1. Continuar a ser uma grandeza contínua e aditiva, que determinará a nova posição da partícula, podendo o processo de discretização ser efectuado antes ou depois do cálculo do novo posicionamento da partícula. Teremos um espaço discreto de *L* localizações por dimensão, onde $L=2^n$, com *n* sendo o número de bits a utilizar para a codificação do problema. Foi esta a solução proposta e utilizada em [Yoshida et. al., 2000], num modelo que combina espaço discreto e contínuo, já que a própria natureza das variáveis do problema assim o exigiam.
2. Abandonar a ideia da dependência linear espaço/velocidade, substituída pela noção de um espaço/velocidade probabilístico, um pouco à imagem das equações que regulam a dinâmica electrónica na mecânica quântica. Teremos apenas duas localizações por dimensão (uma dimensão por cada bit). Neste espaço, a posição da partícula pode ser 0 ou 1; já quanto à velocidade, não representará uma grandeza aditiva ao espaço, mas irá definir uma probabilidade da partícula estar num estado ou noutro. Em resumo, a partícula move-se num espaço de estados restringido a 1 e 0 em cada dimensão, onde cada v_{id} (velocidade da partícula *i*, na dimensão *d*), representa a probabilidade de cada x_{id} (posição de cada partícula *i*, na dimensão *d*) tomar o valor 1.

Esta última interpretação está subjacente à proposta de [Kennedy & Eberhart, 1997]. A fórmula que regula a dinâmica das partículas na sua componente velocidade permanece inalterada, só que agora a localização é um inteiro $\{0,1\}$. Como a velocidade representa a predisposição da partícula efectuar uma transição, e esta ocorre num intervalo [0,1], vai ter de utilizar-se uma função que permita esse mapeamento. A função sigmóide, muito utilizada em redes neuronais, responde aos requisitos de transformação:

$sig(\boldsymbol{v}_i^k) = \frac{1}{1+\exp(-\boldsymbol{v}_i^k)}$, em que $\boldsymbol{v}_i^k$ refere a velocidade da partícula *i*, na iteração *k*.

Assim, a fórmula que regula a dinâmica das partículas continua a ser:

$$\boldsymbol{v}_i^{k+1} = w.\boldsymbol{v}_i^k + c_1.rnd().(pbest - \boldsymbol{s}_i^k) + c_2.rnd().(gbest - \boldsymbol{s}_i^k) \qquad \text{(eq. 4.1)},$$

sendo considerada a mesma limitação para a velocidade em cada dimensão:

$$Se\ \ \boldsymbol{v}_i > \boldsymbol{v}_{\max},\ \ então\ \ \boldsymbol{v}_i = \boldsymbol{v}_{\max};\ \ se\ \ \boldsymbol{v}_i < -\boldsymbol{v}_{\max},\ \ então\ \ \boldsymbol{v}_i = -\boldsymbol{v}_{\max} \qquad \text{(regra 4.1)}$$

Já a localização da partícula, dada a introdução do espaço probabilístico, será definida pela regra:

$$Se \quad rnd() < sig(v_i^{k+1}) \quad então \quad s_i^{k+1} = 1; \quad senão \quad s_i^{k+1} = 0 \qquad \text{(regra 4.2)}$$

Na eq. 4.1, *rnd()* será uma função que gerará números quase aleatórios seleccionados de uma distribuição uniforme no intervalo [0.0, 1.0].

O algoritmo de *O*ptimização *D*iscreta por *E*nxame de *P*artículas (ODiEP) é apresentado formalmente em Algoritmo 4.3, sendo mostrado na Figura 4.7 um seu esquema funcional, algo semelhante à Figura 4.1, mas apenas no que concerne ao ciclo e não na representação real das soluções, na forma de representação do conhecimento e na maneira de efectuar a procura no espaço das soluções.

```
Procedimento ODiEP; // optimização discreta por enxame de partículas
1. Inicialização: inicializar um Enxame E de partículas de forma aleatória
   (pos. e velocidade) no espaço n-dimensional
2. Avaliação do enxame inicial e actualização de pbest e gbest:
   Para Cada partícula p Em E, Fazer:
      Pbest ← posição(p);
      Se aptidão(p) > aptidão(gbest) Então // a aptidão da partícula é superior
         // à melhor anterior
         Gbest← posição(p);  //  actualizar gbest
   FimPara
3. Ciclo correspondente à movimentação e avaliação do enxame:
   Para Cada partícula p Em E, Fazer:
      // Aplicar Dinâmica da partícula:
      // calcular nova velocidade em cada dimensão
      Calcular v_i^{k+1}(p): aplicar equação (4.1) e regra (4.1);
      // calcular nova posição em cada dimensão
      s_i^{k+1}(p): aplicar a regra de actualização dada por (4.2);
      // Avaliação e actualização própria: avaliar cada partícula e actualizar pbest
      Se aptidão(p) > aptidão(pbest) Então // aptidão da partic. > melhor anterior
         Pbest← posição(p); //  actualizar pbest
```

Algoritmo 4.3. Algoritmo de optimização por enxame de partículas na sua versão discreta.

Algoritmo de optimização por enxame de partículas na sua versão discreta (continuado da pág. anterior)

```
        // Avaliação e actualização global:
        Se aptidão(p) > aptidão(gbest) Então // aptidão da partic. > melhor global
           Gbest← posição(p);  //  actualizar gbest
    FimPara
4. Ciclo das sucessivas iterações:
    Repetir 3. Até que um dado critério de convergência seja verificado
5. Retornar Resultado:
    Return Gbest; // retornar a melhor solução conseguida
```

Também nesta versão da OEP o parâmetro v_{max} persiste. Mas aqui o seu conceito é diverso, pois que também difere a noção de velocidade. Neste caso, vai simplesmente limitar a probabilidade máxima de que um bit possa tomar um valor 0 ou 1. Dado utilizar-se a função sigmóide, a exploração de novas soluções será favorecida por valores pequenos de v_{max}, ao contrário do que sucedia na OEP contínua. Na prática, v_{max} é muitas vezes utilizado com valores [-4, +4], mantido constante ao longo de todas as iterações. De qualquer modo, a selecção dos valores adequados para os parâmetros na eq. 4.1 terá de ser realizada de uma forma experimental, já que, muito possivelmente, eles irão depender da natureza do problema a resolver, e a sua solução analítica é muito complexa.

Analogamente ao referido em 4.4.2, relativamente à utilização dos algoritmos genéticos em problemas de carácter combinatório com constrangimentos, também aqui a partícula pode estar localizada numa região do espaço que corresponda a uma solução inválida, pois que viola o constrangimento. As soluções então sugeridas são, aqui, igualmente aplicáveis, mas devidamente adaptadas.

Em primeiro lugar pode-se inserir um algoritmo de reparação que transforme soluções não factíveis em possíveis, normalmente aplicado depois de calculada a nova posição de cada partícula, ou quando todo o enxame foi movimentado. Segundo, não fazer a reparação de uma solução faltosa, mas aplicar uma penalização (um termo adicional que diminua fortemente a sua elegância), impossibilitando a posição da partícula de tornar-se *pbest* ou *gbest*, de forma a que o ODiEP fique focado na região do espaço de soluções factíveis. Terceiro, simplesmente reposicionar-se a partícula para a sua posição *pbest* ou para *gbest*. Finalmente, pode também actuar-se sobre a própria mecânica da partícula, de forma a que só movimentos válidos sejam permitidos.

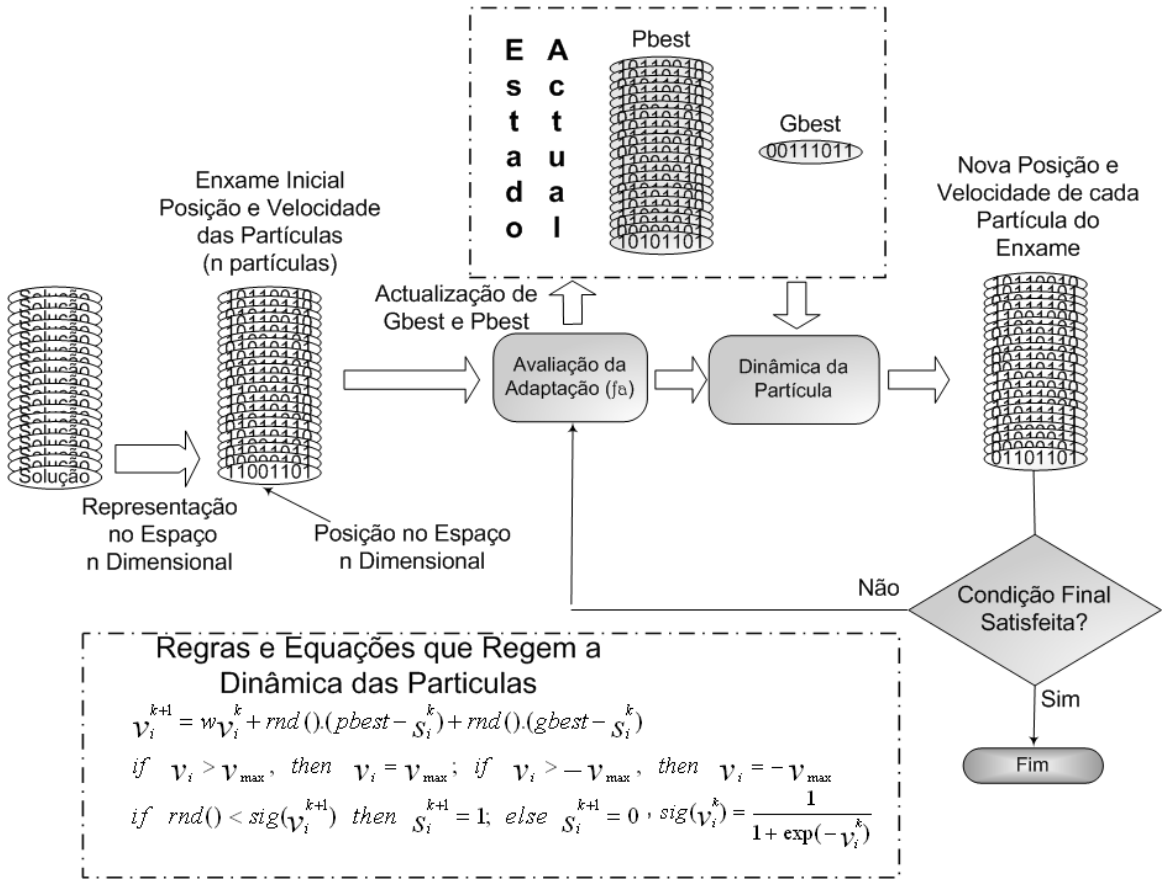


Figura 4.7. Esquema funcional de um algoritmo de optimização discreta por enxame de partículas.

4.6 O Problema da Selecção de Cubos em Ambiente OLAP Centralizado

O problema da selecção de cubos é central neste trabalho, tendo sido atrás descrito e aflorado por várias vezes, já que é o seu objecto de investigação. Como problema de optimização de carácter combinatório, vai poder ser tratado por um dos métodos descritos na secção anterior ou nela referidos. Por outro lado, já se sabe que no cerne de qualquer problema de optimização se tem uma função objectivo que convém definir claramente e mensurar. Importa, assim, colocar o problema em termos mais formais (definição 4.1), para depois especificar como executar a avaliação de cada solução.

Definição 4.1: Selecção de M em OLAP centralizado

Seja $I = \{I_1, I_2, ..., I_n\}$ o conjunto de interrogações com frequências de acesso $F = \{f_1, f_2, ..., f_n\}$ e extensões de interrogação $Ei = \{ei_1, ei_2, ..., ei_n\}$; seja a frequência de actualização e extensão de actualização Fm e $Em = \{em_1, em_2, ..., em_n\}$ respectivamente e seja S o espaço disponível para materialização de subcubos. A solução para o problema de selecção do cubo a materializar é um conjunto $M = \{s_{u1}, s_{u2}, ..., s_{un}\}$ com constrangimento $\sum_j su_j \leq S$, tal que o custo de resposta às interrogações I e de manutenção de M, $Ci(I, M) + Cm(M)$, seja mínimo.

É este o objectivo da optimização que terá de ser mensurado. Aqui vai utilizar-se o modelo de custos linear, respectivas equações de cálculo e algoritmos de estimativa de custos descritos em 3.1, já que correspondem à definição dos custos especificados na definição 4.1. Assim, irão ser utilizadas as equações 3.2 e 3.4 para cálculo dos custos de interrogação e manutenção, consideradas no cálculo dos custos em Algoritmo 3.1 e Algoritmo 3.3, usados para estimativa dos custos correspondentes à função objectivo. Esta será utilizada para cálculo de benefício no algoritmo *greedy* e para função de elegância ou aptidão em AG e ODiEP, algoritmos de optimização para problemas de carácter combinatório a utilizar nesta secção. Se os dois primeiros tinham sido já extensivamente propostos para este problema, o último foi aqui introduzido.

4.7 Algoritmo SCO-AGR: Selecção de Cubos OLAP com Algoritmo Greedy

O algoritmo aqui utilizado (Algoritmo 4.4) deriva directamente do proposto em [Harinarayan et al., 1996], considerando adicionalmente os custos de manutenção na função objectivo.

O algoritmo de manutenção, a utilizar aqui, poderia ter sido alterado, concebendo-se uma heurística específica para este caso, atendendo à natureza construtiva do algoritmo (de acordo com considerandos tecidos em 3.1.3). Optou-se, no entanto, pela uniformização, já que é pretendida uma

avaliação comparativa do desempenho dos algoritmos em causa. Assim, o algoritmo SCO-AGR é bastante simples, uma transposição do Algoritmo 4.1, sendo instanciada a função de benefício em termos da função objectivo da definição 4.1 e dos componentes a utilizar como os subcubos possíveis no *lattice*, sendo a solução o conjunto *M* dos subcubos a materializar que impliquem um custo mínimo.

```
Input: L // Lattice com todas as combinações de granularidade e dependências
       I=(I1... In), S  //  Interrogações e espaço disponível de materialização
Output:  M // Conjunto de subcubos a materializar

Begin
  1. Inicialização:
   E ← S; // E representa o espaço ainda disponível para materialização
   M ← { }; // heurística construtiva: inicialmente M está vazio
  2. Ciclo a executar para Selecção dos Sucessivos Subcubos:
   Enquanto ( E>0 E ∃s∈L∧s∉M : B(M∪{s})>B(M) ) Fazer:  // enquanto houver espaço
     // disponível para materializar novos subcubos e a adição de um subcubo s
     // ainda não materializado trouxer benefício
     sopt ←  Ø ; B ( { sopt },M) : ←  0; // procura um subcubo s ainda não
       // materializado  com o benefício máximo
     ParaCada (s∈L∧s∉M ) // s serão os subcubos ainda não materializados em M;
        // vai calcular-se o ganho para cada um dos seus descendentes
       ParaCada ( s' ∈ { descendentes (s) } )
         B ({s}, M) ← B({s}, M) + (C (s', M) – C (s', M ⋃{s}))
       FimPara
       B ({s}, M) ← B({s}, M) – Cm(s, M  ⋃{s}) // subtrair custos de manutenção
     FimPara
     Se (B{s}, M) > B({sopt}, M) // testa s como o subcubo que proporciona ganho max.
       sopt ←  s; // sopt é o subcubo óptimo
     Fim Se
     Se ( E – Tamanho( {sopt} ) > 0) Então // se o subcubo sopt  ainda puder ser
       // materializado, adiciona-o a M
       M ←  M ⋃ {sopt};  // adiciona o subcubo óptimo seleccionado a M
       E ←  E – tamanho({sopt}); // actualiza o espaço de materialização disponível
     Senão
       E ←  0
     Fim Se
   FimEnquanto
  3. Devolver o resultado:
   Return (M)
End
```

Algoritmo 4.4. Algoritmo SCO-AGR.

Na prática, o algoritmo irá iniciar com um M vazio, a que vai sendo adicionado um novo subcubo (de entre os ainda não materializados) que implique um benefício máximo. A métrica de benefício a utilizar será o ganho em termos da redução de custo implicado pela materialização adicional do novo

subcubo, algo que poderá ser designado como ganho absoluto. Alternativamente, pode também ser avaliado o ganho em termos relativos, dividindo o ganho absoluto pelo espaço gasto pelo subcubo adicional a materializar, o que poderá ser designado como densidade de ganho. O algoritmo terminará logo que não for possível materializar qualquer subcubo adicional, ou porque o espaço de materialização já está todo ocupado, ou então porque não possibilitam qualquer ganho adicional.

4.8 Algoritmo SCO-AGP: Selecção de Cubos OLAP com Algoritmo Genético Padrão

A utilização de um algoritmo genético requer a codificação do problema no genoma dos indivíduos e saber como avaliar cada solução (a sua elegância). A elegância é, neste contexto, simplesmente a soma dos custos de interrogação e manutenção que importa minimizar. Quanto à codificação, há que mapear a possível selecção ou não de um subcubo no genoma. Recordando o que foi dito, em problemas de carácter combinatório, o genoma é constituído por uma cadeia de bits. Ora, nada mais simples do que mapear cada subcubo possível num bit do genoma e, assim, se o bit for 1, isso significará, em termos de *M*, que o correspondente subcubo está materializado, e inversamente. O mapeamento é representado na Figura 4.8.

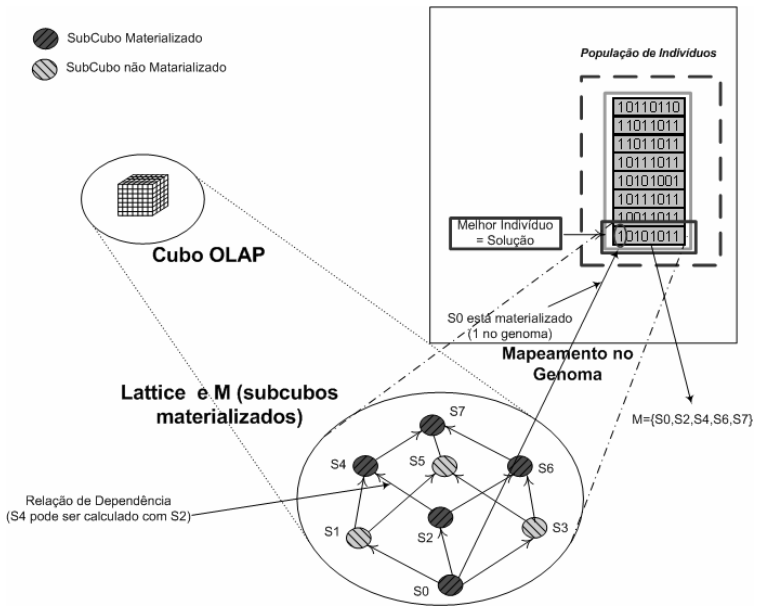


Figura 4.8. Mapeamento do problema da selecção de subcubos no genoma num AG.

Considerando o primeiro bit do genoma (o bit 0, como se pode observar), existe um mapeamento de S0 no bit 0, de S1 no bit 1, ..., de S7 no bit 7. Para *n* possíveis subcubos do *lattice* $L = \{s_o, s_1, ..., s_n\}$ o genoma $G = \{g_o, g_1, ..., g_n\}$ terá a seguinte correspondência: se $g_n = 1 \Rightarrow s_n \exists M$; senão $s_n \nexists M$. Na Figura 4.8, o *M* mostrado é mapeado no genoma do último indivíduo da população, sendo mostrada a correspondência relativa a S0.

Relativamente ao constrangimento espacial, é incluído um mecanismo de reparação com duas variantes. A primeira é baseada numa selecção aleatória de subcubos que vão sendo sucessivamente desmaterializados até que o constrangimento seja satisfeito - o pseudocódigo pode ser visto em Algoritmo 4.5. É uma heurística bastante simples e, portanto, de complexidade muito baixa, o que é vantajoso do ponto de vista do desempenho. Dada a natureza aleatória do esquema, a sua eficácia poderá ser pobre, mas, por outro lado, a própria heurística do AG pode "aprender" o constrangimento e dirigir o processo de pesquisa para zonas promissoras, não violadoras do constrangimento, e, assim, a eventual baixa eficácia do mecanismo pode ser quase não lesiva da eficácia global da qualidade das soluções produzidas. Por outro lado, o baixo custo computacional permanece.

```
Input: M  // Conjunto dos subcubos materializados da solução inválida,
          // (correspondendo ao fenótipo do genótipo de um qq. indiv. da popul. do AG)
       L  // Lattice com todas as combinações de granularidade e tamanhos respectivos
       S  //  Espaço disponível de materialização
Output: M' // Conjunto dos subcubos a materializar que já observam o constrangimento
           // espacial, resultante do genótipo do melhor individuo

Begin
  1. Inicialização:
    M' ← M; // copia M para M' de onde vão ser retirados os sucessivos subcubos
      // até que o constrangimento seja acatado
  2. Remoção de sucessivos subcubos para satisfação do constrangimento:
    Enquanto Tamanho(M')>S Fazer:  // enquanto a solução de materialização não satisfizer
      // o constrangimento espacial
      s_rem ←  rnd(0, nScub(L)-1); // selecciona aleatoriamente um qq. subcubo
        // do lattice: nScub(L) devolve o número total de subcubos no lattice;
        // supõe-se o subcubo 0 é o primeiro e portanto o último será nScub(L)-1
      M'←M'\s_rem; // o subcubo seleccionado aleatoriamente é desmaterializado; não foi
        // incluído um mecanismo para evitar que s_rem não exista em M', pois que,
        // se for esse o caso, não há qq. problema, já que o ciclo vai repetir-se,
        // procurando-se outro subcubo
    FimEnquanto
  3. Retornar resultado:
    Return (M');
End
```

Algoritmo 4.5. Heurística de reparação aleatória.

A segunda heurística é bastante mais elaborada, utilizando uma heurística *greedy* local inversa: começa com o M a reparar e, em cada ciclo, procura qual dos subcubos materializados implica um menor aumento de custo, desmaterializando-o. O ciclo pára quando o constrangimento já estiver satisfeito. O algoritmo pode ser visto em Algoritmo 4.6.

Como se pode observar, por comparação face ao Algoritmo 4.5, é acrescentado um ciclo de pesquisa com a heurística *greedy*, que introduz uma complexidade adicional, proporcional ao número de subcubos em *M*. Uma outra heurística opcional foi incluída, passível de utilização em ambos os algoritmos de reparação: a remoção de subcubos em *M* que permitam um decréscimo de custos, ou seja, aqueles cujo custo de manutenção não é compensado pelo decréscimo dos custos de interrogação.

```
Input: M  // Conjunto dos subcubos materializados da solução inválida (correspondendo ao
          // fenótipo do genótipo de um qq. indivíduo da população do GA)
       L  // Lattice com todas as combinações de granularidade e tamanhos respectivos
       S  // Espaço disponível de materialização
Output: M' // Conjunto dos subcubos a materializar que já observam o constrangimento
           // espacial, a converter para o genótipo do individuo

Begin
  1. Inicialização:
   M' ← M; // copia M para M' de que vão ser retirados os sucessivos subcubos até
     // que o constrangimento seja acatado
   Enquanto (Tamanho(M') > S) Fazer:  // enquanto a solução de materialização não
     // satisfizer o constrangimento espacial, calcular subcubo em M'
     // cuja desmaterialização seja menos lesiva do custo
   2. Selecção greedy de cada subcubo cuja remoção, a cada passo, represente um
      aumento mínimo de custo:
     C_act ← C(M'); // calcula o custo actual de M', para calculo posterior da perda de custo
      // implicada pela desmaterialização de cada um dos subcubos em M'
     PerdaMin ← ∞;// inicializa a um valor máximo, para permitir o cálculo de um mínimo
     ParaCada (s ∈ M') // s será cada um dos subcubos materializados em M'
       // vai calcular-se o mínimo aumento de custo e respectivo subcubo s_rem
       Perda ← C(M'\s) − C_act;
       Se PerdaMin < Perda Então
         PerdaMin←Perda;
         s_rem ←  s;
       FimSe
     FimPara
   3. Remoção do subcubo s_rem seleccionado:
     M' ← M'\s_rem ; // o subcubo de aumento de custo mínimo seleccionado é desmaterializado
   FimEnquanto
  4. Devolver o Resultado:
   Return (M');
End
```

Algoritmo 4.6. Heurística de reparação greedy inversa.

O algoritmo genético é um AG padrão, geracional, ou seja, a evolução processa-se com um conceito bem definido de geração: no final da actuação dos operadores genéticos com criação de nova população, a nova geração substitui integralmente a anterior. Ou seja, no decorrer do processo evolutivo, há duas populações: a actual e aquela a ser gerada.

É adoptado o princípio do elitismo, ou seja, o melhor indivíduo é preservado, passando à próxima geração. O processo de selecção para reprodução pode ser feito por um esquema de competição ou proporcional, seleccionáveis por um parâmetro. Quanto ao *crossover*, é de um ponto, seleccionado aleatoriamente em cada geração. A mutação é controlada por dois parâmetros: a taxa de mutação (a percentagem de indivíduos a sofrer mutação) e a probabilidade de mutação (a probabilidade de um bit ser comutado). O processo de reparação actua logo que são gerados novos indivíduos ou sempre que um indivíduo é objecto de mutação. O pseudocódigo do algoritmo de Selecção de Cubos OLAP com Algoritmo Genético Padrão (SCO-AGP) mostra-se em Algoritmo 4.7, sendo uma instanciação do Algoritmo 4.2, atendendo às considerações anteriores.

```
Input: L  // lattice com todas as combinações de granularidade,
          // dependências, tamanho, dimensões e hierarquias
       I={I1, ..., In}, Ei={Ei1, ..., Ein}, Em={Em1, ..., Emn}, Fm  // perfil de
          // interrogações e actualizações: frequência e extensão
       Pg // parâmetros do algoritmo genético: NumIndiv, NumGerac,
          //TpSelec, PatamarSelecTpC, TxMutac, ProbMutac
       Pb // parâmetros base: S (espaço disponível para materialização),
          // Tr (tipo de reparação)
Output: M  // conjunto de subcubos a materializar
Begin
 1. Inicialização:
  Inicializar parâmetros e serviços do sistema; // carregar parâmetros genéticos,
   // perfil de carga, lattice e esquema dimensional
  P←{ }; // população de indivíduos vazia
 2. Geração, reparação e avaliação da população inicial com mostra do estado:
  Repetir NumIndiv Vezes: // vai gerar a população P de NumIndiv indivíduos
   // e repará-la
   Gerar indivíduo g ; // cada indivíduo tem um genoma com n bits, onde n=número
    // de subcubos do lattice
   Se (Tamanho(g)>S) Então ReparaIndiv (g, Tr); // se indivíduo gerado com
    // genoma cuja solução de materialização viola constrangimento,
    // vai repará-lo, aplicando algoritmo Algoritmo 4.5 ou Algoritmo 4.6
    P←P∪g ;
    Fitness(g)=f (g, I, Ei, Em, Fm); // calcula a adaptação de cada indivíduo
     // gerado (f calculada aplicando Algoritmos 3.1 e 3.3)
  FimRepetir
  MostrarEstado (P); // mostra estado instantâneo da população e melhor solução
   // gerada: quantos indivíduos propõem a materialização de um determinado subcubo
   // (algo indicativo da diversidade genética ) e melhor genoma, correspondendo
   // ao M que mostrou um menor custo
```

Algoritmo 4.7. Selecção de Cubos OLAP com Algoritmo Genético Padrão (SCO-AGP).

Algoritmo 4.7. Selecção de Cubos OLAP com AG Padrão (continuado da página anterior).

```
3. Evolução (sucessivas gerações):
 // gerada, reparada e mostrada a população inicial, segue-se a evolução
 Gerac←0; // contador do número de gerações
 Enquanto ( Gerac<NumGerac)) Fazer: // condição de finaliz. é o número de gerações
  PontoCruz← rnd (1, nScub(L)-2); // selecciona o ponto de crossover a usar para
   // esta geração: nScub(L) devolve o número total de subcubos no lattice;
   // supõe-se o subcubo 0 é o primeiro, e portanto o último será nScub(L)-1
  P'←{ } // cria a nova população vazia
 Enquanto |P'|<=NumIndiv Fazer: // executar cruzamentos até gerar uma nova
   // população com número de indivíduos igual a P
   P'←P' ∪ melhor (P); // princípio do elitismo: o melhor indivíduo é preservado
    // na nova população e assim fica disponível para futuros cruzamentos
   // Cruzamentos
   Pais←{ }; Filhos←{ } // cria pais e filhos: seleccionar dois pais da população
    // a ser objecto de cruzamento gerando dois filhos
   Repetir 2 Vezes:
    Pais←SelecIndiv (P, TpSelec, PatamarSelecTpC); // selecciona indivíduos
     // conforme tipo de selecção (proporcional ou competição)
   FimRepetir
   Filhos←Crossover(Pais, PontoCruz); // gera os dois filhos, aplicando
    // o operador crossover de um ponto

   ParaCada filho Em Filhos, Fazer: // vai reparar cada filho gerado
    Se (Tamanho(filho)>S) Então ReparaIndiv (filho, Tr); // se indivíduo
     // gerado com  genoma cuja solução de materialização viola constrangimento,
     // vai repará-lo, aplicando Algoritmo 4.5 ou Algoritmo 4.6
     P'←P' ∪ filho ;
   FimPara
  FimEnquanto
  // Mutação
  Repetir NumIndiv*TaxMutac Vezes: // vai operar a mutação sobre um número
   // de indivíduos de TaxMutac
   IndivMutar← rnd (0, NumIndiv-1) // selecciona aleatoriamente o indivíduo a
    // sofrer mutação, supõe-se que o primeiro indivíduo é o 0 e aplica a cada bit
    // do genoma do indivíduo a mutar a probabilidade de mutação ProbMutac
    // (probabilidade em 100)
   ParaCada bit b Em genoma (P', IndivMutar) Fazer:
    Se rnd (0,100)<ProbMutac Então
      Se b=0 Então b ←1; Senão b ←0;
   FimPara
   Se (Tamanho (P', IndivMutar)>S) Então ReparaIndiv ((P',IndivMutar), Tr);
    // se indivíduo mutado tiver genoma cuja solução de materialização viole
    // constrangimento, vai repará-lo, aplicando Algoritmo 4.5 ou Algoritmo 4.6
  FimRepetir
  // nova população gerada e reparada: aplicar operador substituição
  P←{ } // limpa a população antiga
  Para Cada Indiv Em P', Fazer:
   P←P ∪ Indiv;
  FimPara
  // avaliação da aptidão da nova população
  ParaCada Indiv Em P, Fazer:
   Fitness(Indiv)= f (Indiv, I, Ei, Em, Fm); // calcula a aptidão de cada indiv.
    // ( f  calculada aplicando algoritmos 3.1. e 3.3)
  FimPara
```

Algoritmo 4.7. Selecção de Cubos OLAP com AG Padrão (continuado da página anterior).

```
    MostrarEstado (P); // mostra estado instantâneo da população e melhor
       // solução gerada
    Gerac←Gerac+1; // incrementa o número da geração
   FimEnquanto
  4. Devolver o resultado:
   Return M (melhor (P)); // retorna o M correspondente ao genoma do melhor indivíduo
End
```

4.9 Algoritmo SCO-ODiEP: Selecção de Cubos OLAP por Optimização Discreta com Enxame de Partículas

Neste algoritmo, importa também solucionar a questão relacionada com a forma de codificar o problema. Como atrás foi referido, a cada ponto no espaço corresponde uma solução. A natureza discreta do espaço impõe que, em cada dimensão, a partícula só pode estar na posição 0 ou 1. Se a cada dimensão do espaço se fizer corresponder um subcubo possível do *lattice*, o problema da codificação parece resolvido. Suponha-se uma partícula P que, num espaço 8-dimensional, está, num dado momento, na posição (01101001). Dado o mapeamento ora definido, esta posição corresponderá à proposta de materialização $M = \{s_1, s_2, s_4, s_7\}$, supondo que a primeira dimensão é mapeada para s_0 e a última para s_7. A solução de mapeamento pode ser vista na Figura 4.9.

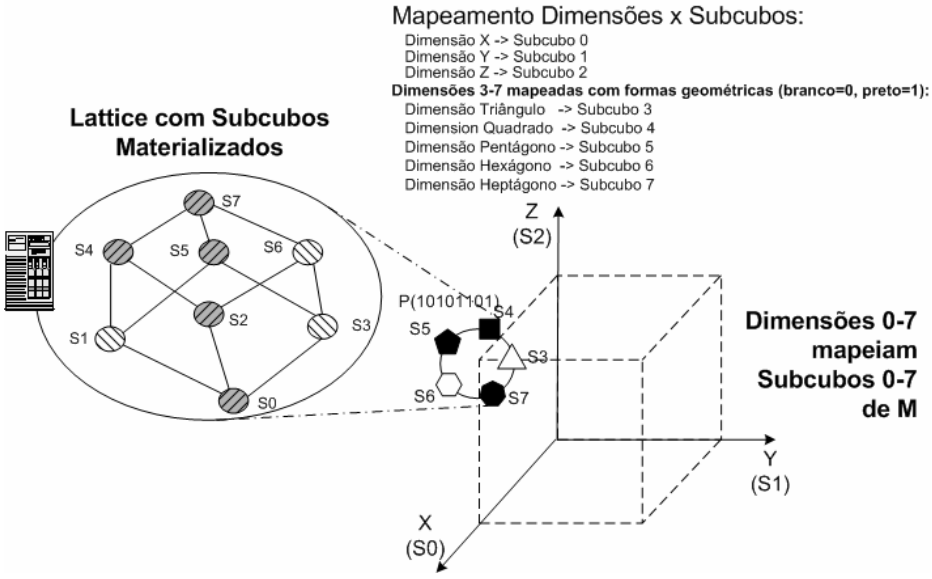


Figura 4.9. Mapeamento do problema da selecção de subcubos no espaço ODiEP.

O esquema de mapeamento da figura é, talvez, um tanto difícil de perceber. O problema é que, como há 8 possíveis subcubos, isso implica 8 dimensões no espaço ODiEP. Como a representação de mais de três dimensões não é, de todo, simples, optou-se pela especificação das três dimensões do espaço euclidiano, sendo as restantes 5 representadas por figuras geométricas, com a correspondência indicada na figura (algo intuitiva, já que o número de faces do polígono vai sugerir o número do subcubo mapeado). A posição das cinco figuras no espaço euclidiano dá a posição da partícula (sendo a correspondente posição na dimensão, 1, se o polígono estiver a negro e inversamente). No caso da figura, a partícula estará posicionada no ponto P(1,0,1,0,1,1,0,1): X(1), Y(0), Z(1), Triângulo(0), Quadrado(1), ..., Heptágono(1). Quanto à função de elegância, tem-se a soma dos custos de interrogação e manutenção, identicamente ao utilizado em 4.8.

Já o reposicionamento das partículas, sempre que a sua localização sugira uma solução inválida é tratado de forma semelhante, também empregando as heurísticas concebidas para situações análogas em 4.8, só que aqui, não importa apenas o reposicionamento da partícula, mas que a velocidade, na dimensão correspondente, possa gerar essa nova posição. Vai mostrar-se aqui (em Algoritmo 4.8) o correspondente ao Algoritmo 4.5, adaptado ao ODiEP, e considerando a questão acabada de referir.

```
Input: PosPartic // Posição da partícula no espaço n-dimensional, correspondendo
         // aos subcubos materializados
       VelPartic // velocidade da partícula
       L // Lattice com todas as combinações de granularidade e tamanhos respectivos
       S // Espaço disponível de materialização
Output: PosPartic'// Nova posição da partícula, depois de reposicionada
          // de forma que já seja uma localização válida
        VelPartic'// Nova velocidade da partícula
Begin
 1. Inicialização
  PosPartic' ← PosPartic; // copia a posição da partícula para PosPartic' que
    // vai ser reposicionada (algumas posições 1 vão passar a zeros, para que a nova
    // posição seja válida): que o M correspondente obedeça ao constrangimento espacial
 2. Remoção de sucessivos subcubos para satisfação do constrangimento:
  Enquanto Tamanho(Map(PosPartic', M)) > S  Fazer:  // enquanto o mapeamento da posição da
    // partícula em M não satisfizer o constrangimento espacial
    dim_rep ← rnd({dim : PosPartic' = 1}); // selecciona aleatoriamnete um dimensão onde
      // a partícula esteja numa posição no espaço=1
    dim_rep(PosPartic') ← 0; // a posição da partícula na dimensão seleccionada é colocada a 0
      // (operação que corresponde à desmaterialização do subcubo respectivo)
      // vai procurar uma velocidade na dimensão que calcule uma posição consistente
      // com a nova posição
```

Algoritmo 4.8. Algoritmo de reparação aleatória para reposicionamento de partícula.

Algoritmo 4.8. Algoritmo de reparação aleatória para reposicionamento de partícula (continuado da pág. anterior).

```
    Enquanto NovaPos ≠ 0 Fazer
      NovaVel ← rnd([−V_max, +V_max]) // selecciona aleatoriamente uma nova vel. para a partícula
      Se rnd() < sig(NovaVel)
      Então NovaPos ← 1 ;
      Senão NovaPos ← 0 ;
      FimSe
    FimEnquanto
    dim_rep(VelPartic') ← NovaVel ;
  FimEnquanto
 3. Devolver resultado:
  Return (PosPartic', VelPartic');
End
```

O pseudocódigo do algoritmo de Selecção de Cubos OLAP por Optimização Discreta com Enxame de Partículas (SCO-ODiEP) apresenta-se em Algoritmo 4.9, sendo uma instanciação do Algoritmo 4.3. Não se incluem quaisquer mecanismos de hibridação genética (relocalização em *gbest* ou *pbest* [Angeline, 1998] ou ainda cruzamentos [Løvbjerg et al., 2001]), nem enxames multi-fase [Al-Kazemi & Mohan, 2002], opções a incluir em versão a apresentar no próximo capítulo, quando for tratada a optimização em arquitecturas M-OLAP.

```
Input: L  // lattice com todas as combinações de granularidade, dependências, tamanho,
          // dimensões e hierarquias
       I={I1, ..., In}, Ei={Ei1, ..., Ein}, Em={Em1, ..., Emn}, Fm  // perfil de
          // interrogações e actualizações: frequência e extensão
       Pe // parâmetros do algoritmo de enxame de partículas: NumPartic,
          // NumIter, Vmax, Wini, c1, c2
       Pb // parâmetros base: S (espaço disponível para materialização),
          // Tr (tipo de reparação)
Output: M // conjunto de subcubos a materializar

Begin
 1. Inicialização:
  Inicializar parâmetros e serviços do sistema; // carregar parâmetros OEP,
   // perfil de carga, lattice e esquema dimensional
  E←{ }; d←NSCubos(L)  // enxame vazio e d=número de dimensões do espaço OEP
   // discreto (igual ao número de subcubos possíveis)
  D←{d1, ..., dd}; // espaço OEP d dimensional
  gbest ← 0;
```

Algoritmo 4.9. Selecção de Cubos OLAP por Optimização Discreta com Enxame de Partículas (SCO-ODiEP).

Algoritmo 4.9. (SCO-ODiEP), continuado da página anterior.

```
  2. Geração, reparação, avaliação e mostra do estado do enxame inicial:
   Repetir NumPartic Vezes: // vai gerar o enxame com NumPartic partíc. e repará-lo
    p←GeraPartic(Vmax, Dim(nScub(L)) // gera velocidade aleatoriamente (entre
     // -Vmax e Vmax) e aplica regra 4.2 para calcular posição para cada dimensão
     // (igual ao número de subcubos possíveis, dada o mapeamento do problema)
    Se (Tamanho(M (p))>S) Então ReparaPartic (p, Tr); // se partícula gerada em
     // posição cuja solução de materialização viola constrangimento, vai repará-lo,
     // aplicando Algoritmo 4.8 ou correspondente OEP de Algoritmo 4.6
    E←E∪p;
    Fitness(p)=f (p, I, Ei, Em, Fm); // calcula a adaptação de cada partícula
     // gerada (f calculada aplicando algoritmos 3.1 e 3.3)
    pbest(p) ← Fitness(p); // pbest inicial de cada partícula é a aptidão da posição
     // inicial da partícula
    Se Fitness(p)>gbest Então gbest ← Fitness(p); // calcula gbest relativo
     // ao enxame inicial
   FimRepetir
   MostrarEstado (E); // mostra estado instantâneo do enxame e gbest: quantas
    // partículas propõem a materialização de um determinado subcubo e gbest,
    // correspondendo à melhor posição de uma qq. partícula
  3. Movimentação do enxame
   // gerado, reparado e mostrado o enxame inicial, segue-se a movimentação
   Iter←0; // contador do número de iterações
   Enquanto ( Iter<NumIter)  Fazer: // condição de finaliz. é o número de iterações
    ParaCada p Em E, Fazer: // vai movimentar cada partícula, aplicando equações
     // e regras da dinâmica das partículas
     ParaCada d Em D, Fazer:
      // calcular nova velocidade da partícula para dada dimensão d
      v_Pd^(iter+1) = w.v_Pd^iter + c1.rnd().(pbest_Pd - s_Pd^iter) + c2.rnd().(gbest - s_Pd^iter)   // aplicar eq. 4.1
       // s_Pd^iter é a posição da partícula p na dimensão d na iteração iter
      Se v_Pd^(iter+1) > v_max Então v_Pd^(iter+1) = v_max SenãoSe v_Pd^(iter+1) < v_max Então v_Pd^(iter+1) = -v_max ; // aplicar regra 4.1
      // calcular localização da partícula
      Se se rnd() < sig(v_Pd^(iter+1)) Então s_Pd^(iter+1) = 1 Senão s_Pd^(iter+1) = 0 ; // aplicar regra 4.2
     FimPara
     Se (Tamanho (M (p))>S) Então ReparaPartic (p, Tr); // se partícula ocupa
      // posição cuja solução de materialização viola constrangimento, reposiciona-a
      // aplicando algoritmo Algoritmo 4.8 ou correspondente OEP de Algoritmo 4.6
    FimPara
   FimEnquanto
   ParaCada p Em E, Fazer: // calcula aptidão de cada partíc. e act. pbest e gbest
    Fitness(p)=f (p, I, Ei, Em, Fm); // calcula a adaptação de cada partícula
     // gerada (f calculada aplicando algoritmos 3.1 e 3.3)
    Se Fitness(p)>pbest(p) Então pbest(p) ← Fitness(p); // se nova aptidão é
     // melhor do que a melhor aptidão atingida pela partícula
    Se Fitness(p)>gbest Então gbest ← Fitness(p); // calcula gbest relativo à
     // nova posição das partículas do enxame
   FimPara
  4. Devolver resultado:
   Return M (gbest); // retorna o M correspondente à melhor posição atingida
    // por uma qualquer partícula do enxame
End
```

4.10 Avaliação Comparativa Experimental Simulada

Especialmente para avaliar o desempenho do algoritmo SCO-ODiEP, já que representa uma nova proposta para a optimização da selecção de cubos, realizaram-se testes experimentais divididos em dois grupos: o primeiro grupo de testes procurou avaliar o novo algoritmo relativamente à heurística *greedy* profusamente utilizada neste domínio, tendo sido implementado, para isso, o algoritmo SCO-AGR; o segundo utilizou o SCO-AGP para termo de comparação.

4.10.1 Implementação: Componentes

A ideia base foi conceber um sistema de simulação que pudesse servir como bancada de trabalho, permitindo a expansão fácil, por simples inclusão de novos componentes. Desta forma, seria possível o suporte não só do problema tratado neste capítulo, mas das soluções aos problemas a abordar em outras fases deste trabalho.

Concebeu-se assim um conjunto de classes dotadas de interfaces apropriadas, o que permitiria a sua reutilização em outros casos. Todos os componentes foram implementados em Java, incluindo as classes que se apresentam seguidamente, conforme parcialmente descrito em [Loureiro & Belo, 2006a].

Relativamente ao algoritmo SCO-ODiEP, destacam-se as seguintes classes:

- ParamAlgPSwarm, que carrega e disponibiliza os parâmetros de trabalho que regem a execução do algoritmo ODiEP, uma extensão aos parâmetros base; podem referir-se: o número de partículas, o número de iterações a executar, *Vmax*, *w* inicial e tipo de reparação a aplicar.
- ParticleSwarm, que permite a criação de objectos enxame, mantendo o estado (especialmente no que concerne à localização, velocidade, pbest e gbest) e disponibilizando um conjunto de serviços à classe principal.
- PSA que implementa o algoritmo SCO-ODiEP (Algoritmo 4.9).

O algoritmo SCO-AGP utiliza as classes:

- ParamAlgGenet, com função similar ao ParamAlgPSwarm, que carrega e disponibiliza os parâmetros de trabalho que controlam o funcionamento do AG, sendo uma extensão aos parâmetros base; à semelhança da OEP, podem referir-se: número de indivíduos, número de

gerações, tipo de selecção, patamar (se selecção tipo competição), percentagem de cruzamentos e de mutação, probabilidade de mutação e tipo de *crossover*.

- Genoma, que permite a criação da população de indivíduos (com os respectivos genomas), mantendo o seu estado (especialmente o seu genótipo) e disponibilizando um conjunto de serviços à classe principal.
- Selecção, que implementa o processo de selecção, de acordo com o tipo de selecção especificada num parâmetro integrado em ParamAlgGenet.
- GA implementa o próprio algoritmo genético (SCO-AGP) (Algoritmo 4.7).

Já o algoritmo SCO-AGR, bastante mais simples, inclui as classes:

- EspaçoDisp, uma classe que mantém o espaço disponível, para assim se poder controlar o constrangimento espacial.
- Greedy, a implementação do Algoritmo 4.4.

O sistema, como um todo, utiliza o serviço de outras quatro classes:

- M, uma proposta de materialização de subcubos, dotada de um conjunto de serviços, de entre os quais há a salientar: 1) o cálculo dos custos de interrogação e manutenção, uma implementação dos algoritmos Algoritmo 3.1 e Algoritmo 3.3, de acordo com o modelo de custos linear (secção 3.1) e equações de cálculo respectivas (secção 3.1.1); 2) a reparação, uma implementação do Algoritmo 4.5 e Algoritmo 4.6 com interface apropriada para suportar de igual forma o Algoritmo 4.8 e o equivalente ao Algoritmo 4.6 para ODiEP.
- QualifQueryCubeView, capaz de calcular e disponibilizar as relações de dependência entre subcubos (atendendo à estrutura do *lattice*), usando não mais do que a definição das dimensões e hierarquias e o código de cada subcubo: é, assim, uma implementação do Algoritmo 3.2.
- StateDistrib que torna disponíveis os serviços necessários à visualização do estado instantâneo da população de indivíduos ou do enxame, bem como da melhor solução atingida até ao momento.
- OutBook, utilizado para tornar persistentes estados instantâneos, sua evolução ou resultados finais ou parciais, e assim possibilitar a sua inclusão em gráficos, tabelas ou relatórios, o que é conseguido através de um conjunto de serviços de saída de dados.

Para o suporte ao algoritmo é ainda utilizado um outro conjunto de classes, formando três grandes grupos, que carregam e tornam disponíveis todos os parâmetros e outra informação relacionada com a estrutura do *lattice* e ambiente:

- Parâmetros arquitecturais de base;
- O *lattice* (subcubos, tamanhos, dimensões, hierarquias, hierarquias paralelas);
- As interrogações, suas frequências e extensão e a frequência e extensão de manutenção.

4.10.2 Implementação: Ambiente e Parâmetros

Para dados de teste, usou-se, na maioria das vezes, o cubo A (Tabela 2.1, secção 2.6), supondo um custo de utilização das relações base de três vezes o tamanho do subcubo de granularidade mais fina (cps). Foi igualmente utilizado o cubo B (Tabela 2.3, secção 2.6) quando foi pretendida uma avaliação da capacidade dos algoritmos para lidar com problemas mais complexos, já que, neste caso, é adicionada uma nova dimensão (com hierarquia dimensional), traduzindo-se num número de subcubos quatro vezes maior.

Dada a profusão de parâmetros existentes nos algoritmos SCO-AGP e SCO-ODiEP, procurou-se informação relativa aos valores típicos a utilizar em problemas de optimização de igual carácter. Usando os valores encontrados, procedeu-se depois a alguns ajustes finos, de uma forma empírica, simplesmente por afinação sucessiva, de forma a maximizar o desempenho.

Começando com o SCO-AGP, a visualização do estado instantâneo e a qualidade das soluções obtidas permitiram, de imediato, perceber a facilidade de convergência do algoritmo. Algumas alterações em parâmetros como população, número de gerações e tipo de selecção levou a um ajuste do balanço entre a exploração e focalização e da intensidade selectiva X diversidade genética. Usaram-se assim os parâmetros genéticos mostrados na Tabela 4.3.

Quanto ao algoritmo SCO-ODiEP, mais uma vez procurou-se, através de um conjunto de testes preliminares, efectuar a afinação no valor dos parâmetros, especialmente *w*, *Vmax*, *c1* e *c2*. Como muita da informação disponível, relativa a valores típicos para estes parâmetros, se referiam à versão contínua da OEP, este processo foi algo mais delicado.

Assim, usando um número de iterações suficientemente grande (para que este parâmetro não pudesse ser factor limitativo na aquisição de soluções) e um valor inócuo de 1 para *w*, usaram-se valores de *Vmax*=5, 10 e 100, para avaliar a qualidade das soluções conseguidas. O gráfico da Figura 4.10 (à esquerda), mostra os valores obtidos. Verificou-se que, para os valores testados, este

parâmetro não tinha um impacto muito marcante na qualidade das soluções obtidas. Se um valor muito grande (Vmax=100) conduzia a uma procura lenta da solução, os valores 5 e 10 pareciam boas opções. Em termos da qualidade final das soluções, o melhor valor foi obtido para Vmax=10, pelo que se seleccionou este valor, possivelmente aquele que corresponderá a um melhor balancear entre a exploração e a focalização do algoritmo. Na verdade, à medida que *Vmax* aumenta, menor será a probabilidade de que a partícula mude de estado, o que explica a lentidão da convergência para Vmax=100.

Tabela 4.3. Parâmetros genéticos utilizados nos testes experimentais.

Tipo de Selecção	Competição. Buscar aleatoriamente dois indivíduos e, com 85% de probabilidade, o indivíduo mais apto é seleccionado, e reciprocamente.
% Cruzamentos	90
Crossover	Um ponto
Ponto de *Crossover*	Seleccionado aleatoriamente em cada geração
% Mutação	5
Probabilidade de Mutação	1 em cada 8 bits (12.5%)
Forma de lidar com soluções inválidas	Heurística de Reparação Aleatória (Algoritmo 4.5)

Num teste preliminar procurou-se avaliar o impacto de *w* (factor de inércia) no desempenho do algoritmo. Para OEP contínua, sugere-se, por exemplo em [Eberhart & Shi, 2001] que o valor de *w* seja linearmente decrementado com o número de iterações, desde um valor inicial de 0.9 até 0.4. Mas em OEP discreta, o efeito de *w* é oposto. Portanto, o mecanismo teria de ser invertido. Assim, usaram-se os valores 0.4 e 0.9. Os resultados foram, inicialmente, desanimadores. O algoritmo, simplesmente, não convergia, devido a fenómenos de oscilação. Procurada a causa, percebeu-se que valores baixos de *w* implicam uma elevada probabilidade de mudança de posição da partícula, mesmo na ausência de modificações na velocidade. Foram assim tentados outros valores. Para $0.95 \leq w \leq 1$, a convergência era assegurada. O gráfico da Figura 4.10 (à direita) mostra o comportamento evidenciado.

Como se pode observar, para este intervalo de valores, mais uma vez o impacto na qualidade das soluções é pequeno (as linhas quase se sobrepõem). Não foram patentes valores definitivamente melhores para w, já que, em alguns intervalos, um é melhor, mas, noutro intervalo, a situação inverte-se. Dado o comportamento mais uniforme para w=0.98, foi este o valor escolhido, embora qualquer outro, dentro dos valores testados, teria sido igualmente válido. Finalmente, quanto a c1 e

c2, verificou-se que o seu impacto era relativamente limitado para valores entre 1 e 2 (valores típicos referidos na literatura). Desta forma, utilizou-se um valor de 1.74, igual para ambos.

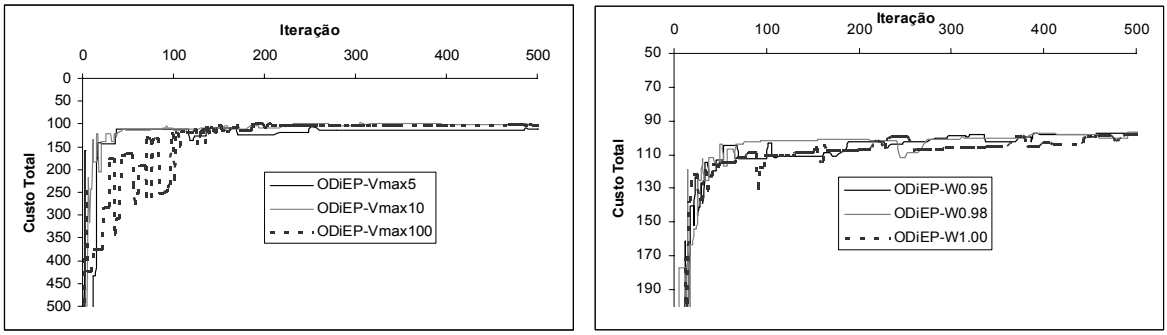


Figura 4.10. Resultados dos testes preliminares para ajuste dos parâmetros *Vmax* e *w*.

4.10.3 Testes Efectuados

Para a avaliação experimental dos algoritmos foi executado um conjunto de testes, de forma a estimar a qualidade das soluções, a rapidez na obtenção de boas soluções e também a escalabilidade. Como se pretende, especialmente, avaliar o desempenho do novo algoritmo proposto (SCO-ODiEP), organizaram-se os testes em três fases: na primeira, procura mostrar-se o comportamento do algoritmo para situações onde a análise comparativa não se coloque; em cada uma das outras, é efectuada a avaliação comparativa face a um dos outros algoritmos. Permite-se assim uma menor sobrecarga de elementos nos gráficos, além de seguir, mais de perto, os artigos referenciados [Loureiro & Belo, 2006c] e [Loureiro & Belo, 2005].

Na fase 1, procurou-se avaliar a evolução da qualidade das soluções com o número de iterações e o impacto do tamanho do enxame (número de partículas). Usou-se o cubo A para dados de teste e uma frequência de interrogações aleatória, normalizada para um número total de subcubos igual ao do cubo A. A Figura 4.11 (à esquerda) mostra os resultados do primeiro teste e a Figura 4.11 (à direita) os resultados para o segundo. É claro um impacto residual na qualidade das soluções de ambos os factores em análise. Mesmo com 5 partículas e 20 iterações, foi observado um aumento de apenas 3% no custo das soluções obtidas. Isto está de acordo com o dito em [Shi & Eberhart, 1999], onde é mostrado que a OEP contínua, com tamanhos de populações de partículas diferentes, revela um desempenho quase idêntico. O gráfico da esquerda mostra também um decréscimo importante de custos até percentagens de espaço de materialização de 10%; acima desse espaço, o ganho é quase nulo, devido ao impacto progressivo dos custos de manutenção, que não se revelam grandemente

benéficos em termos dos ganhos em custos de interrogação (os novos subcubos a materializar podem ser substituídos, sem grandes perdas, por outros já materializados, sendo, assim, quase redundantes).

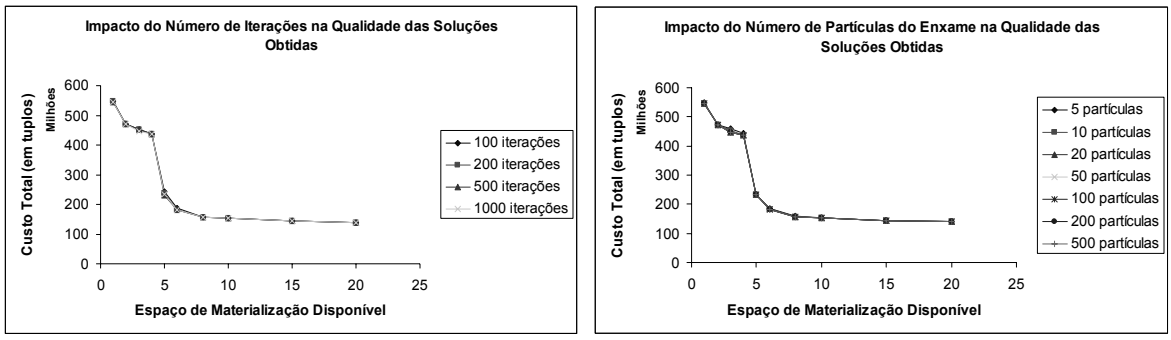


Figura 4.11. Impacto do número de iterações e do número de partículas do enxame na qualidade das soluções obtidas.

Relativamente à velocidade de execução do algoritmo (Figura 4.12, à esquerda), o teste mostrou que o espaço de materialização, bem como o número de subcubos seleccionados, têm um impacto nulo no tempo de execução, mas que este conhece um aumento com o número de iterações, com um rácio quase idêntico (para 200/100 no número de partículas, um rácio de 1.88 e para 2000/100, um rácio de 16.66). Este comportamento é expectável, já que, em cada ciclo, as operações a executar são as mesmas. Já o aumento do número de partículas do enxame também faz aumentar o tempo de execução (Figura 4.12, à direita), mas com um rácio bastante inferior: um aumento de 100x no número de partículas importa num aumento de tempo de execução de apenas 5.43x; duplicando o número de partículas (de 5 para 10), o tempo de execução conheceu um aumento de apenas 8%. Em termos reais, o tempo de execução do algoritmo foi de 3.4 segundos para 200 iterações com 5 partículas X 200 iterações com 500 partículas, em 25 segundos.

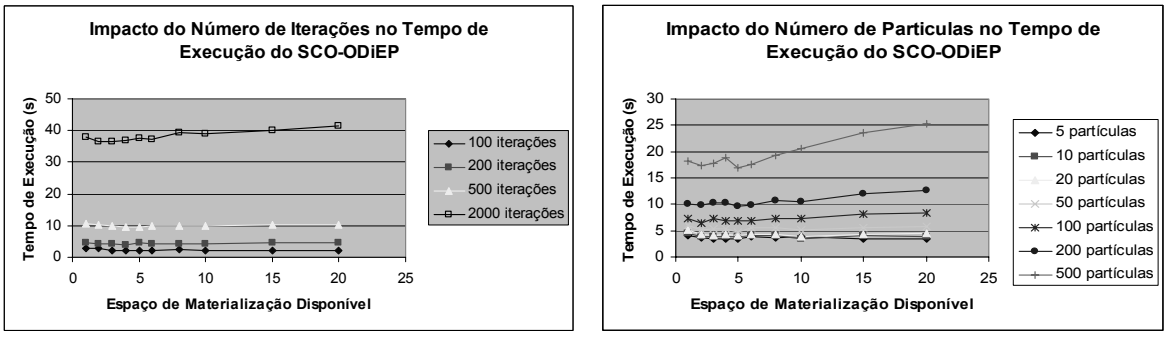


Figura 4.12. Impacto do número de Iterações e partículas no tempo de execução do SCO-ODiEP.

Este conjunto de testes terminou com a avaliação da evolução da qualidade da solução conseguida com o número de iterações (Figura 4.13). Usou-se um enxame com 100 partículas e 10% de espaço de materialização. O gráfico evidencia que o SCO-ODiEP é bastante rápido na procura de boas soluções.

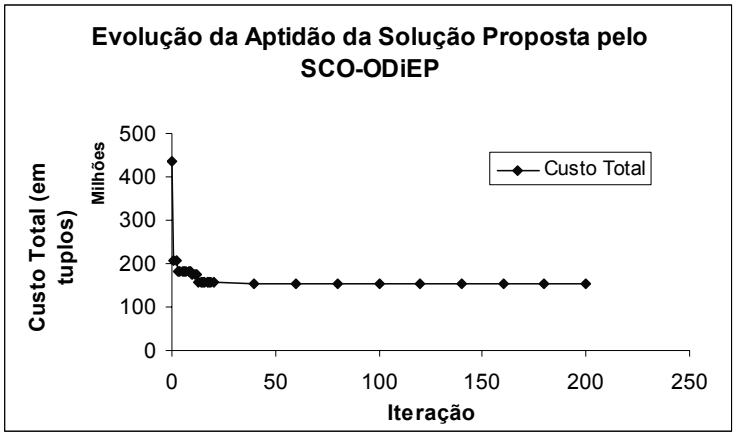


Figura 4.13. Melhor aptidão *versus* número de iterações.

O comportamento evidenciado é bastante importante, uma vez que: 1) o aumento do número de subcubos seleccionados tem um impacto quase nulo no tempo de execução; 2) o algoritmo revelou facilidade e rapidez na procura de boas soluções, mesmo com um número pequeno de partículas no enxame e ao fim de poucas iterações. Estas características permitem concluir que o algoritmo deverá suportar esquemas dimensionais bastante mais complexos, já que, nas condições actuais, parece ter reservas abundantes de "potência" para atacar problemas maiores: mesmo necessitando de um enxame mais populoso e mais iterações para encontrar boas soluções, ainda assim, o tempo de execução não será exagerado.

Outro conjunto de testes usou uma frequência uniforme, tendo os resultados confirmado o comportamento evidenciado atrás.

Na fase 2, efectuou-se a avaliação comparativa do SCO-ODiEP X SCO-AGR. Quanto ao tempo de execução, para o caso de teste, o SCO-AGR foi bastante mais rápido, pois que o número de subcubos seleccionados é pequeno. Mas, para esquemas dimensionais mais complexos, o balanço já não lhe será tão favorável, conhecida a sua complexidade $O(K.n^2)$, onde k é o número de subcubos seleccionados e n o número de subcubos total do cubo.

O outro teste deste grupo foi a avaliação do desempenho comparativo (em termos da qualidade das soluções conseguidas) do SCO-ODiEP X SCO-AGR (Figura 4.14). Pode ver-se um desempenho consistentemente superior do primeiro algoritmo (marginal para espaços de materialização até 4%), mas claro para valores superiores. Especialmente no intervalo [5-10%], o SCO-ODiEP é largamente melhor, o que é muito interessante em DW reais, já que o cubo a materializar tem de ser restringido a valores percentuais dessa ordem.

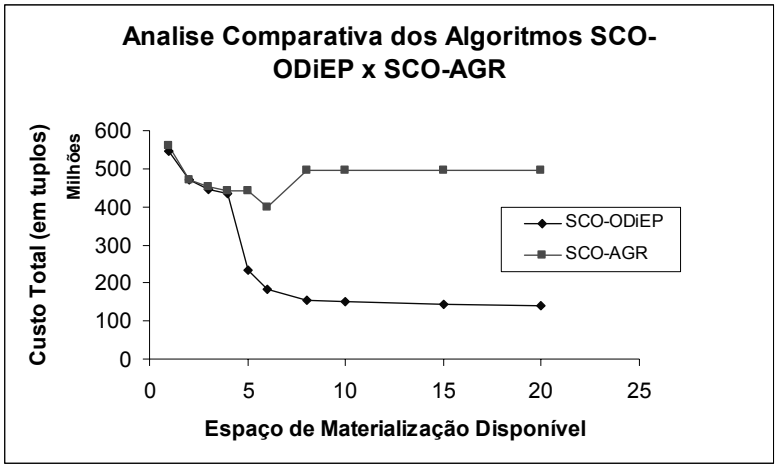


Figura 4.14. Comparação dos algoritmos SCO-AGR e SCO-ODiEP.

Na última fase dos testes, vai efectuar-se uma avaliação comparativa do SCO-ODiEP face ao SCO-AGP. A abordagem genética já mostrou, em várias implementações (por exemplo [Lin & Kuo, 2004]), um bom desempenho comparativamente aos algoritmos *greedy*, fazendo, assim, todo o sentido utilizar-se também para avaliar comparativamente o algoritmo SCO-ODiEP.

O teste inicial compara as soluções geradas por ambos os algoritmos. Para o SCO-AGS, usou-se uma população de 200 indivíduos, 200 gerações, %cruzamento=90, %mutação=5 e probabilidade de mutação=12.5%. Para o SCO-ODiEP, usaram-se 20 partículas e 200 iterações. A Figura 4.15 mostra o resultado do teste, considerando frequências uniformes (gráfico da esquerda) e frequências aleatórias (gráfico da direita). A similaridade da qualidade das soluções conseguidas por ambos os algoritmos é evidente. Inspeccionados os próprios valores, nota-se uma ligeira vantagem do SCO-ODiEP (cerca de 6.3%).

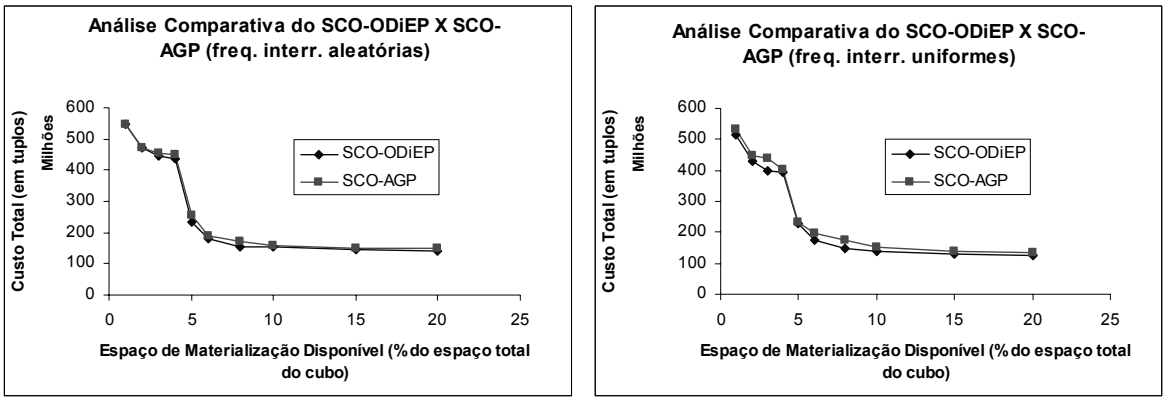


Figura 4.15. Avaliação comparativa do SCO-ODiEP e SCO-AGP.

Relativamente ao tempo de execução, observou-se um impacto quase nulo do espaço de materialização em ambos os algoritmos, apesar de ser observado um rácio de 1:1.5 (a favor do SCO-ODiEP). Este valor do rácio pode ser explicado pelo maior número de indivíduos da população do SCO-AGP X número de partículas no enxame. Apesar disso, a qualidade das soluções tende um pouco para o SCO-ODiEP.

Na próxima experiência avalia-se o impacto do número de elementos da população na qualidade das soluções obtidas e na convergência. Usou-se um espaço de materialização de 10% do tamanho total do cubo, 200 gerações (ou iterações) e os mesmos valores da experiência anterior para os restantes parâmetros. Relativamente ao parâmetro cujo impacto se pretende avaliar, usaram-se 4, 10, 20, 50, 100, 200 partículas e multiplicou-se por cinco cada um destes números para se obter o número de indivíduos na população do SCO-AGP (conhecida a necessidade de um maior número de indivíduos para o AG ter um bom desempenho e o impacto reduzido do número de partículas na qualidade das soluções obtidas para a versão contínua da OEP) [Shi & Eberhart, 1999]. A Figura 4.16 mostra o resultado deste teste. Se é óbvio o impacto quase nulo do número de partículas na qualidade das soluções (confirmando o visto na secção anterior), o mesmo não é válido para o SCO-AGS: quando o número de indivíduos é 20, o custo total é 190E6; já para uma população de 1000 indivíduos, o custo é de 160E6 (revelando um ganho de 18%). Também é importante salientar que, mesmo com um número pequeno de partículas, o SCO-ODiEP mostra ainda grande facilidade em obter boas soluções, convergindo rápida e eficazmente. O mesmo não é válido para o SCO-AGP, onde é óbvia a sua maior dificuldade em encontrar uma boa solução, especialmente para uma população pequena.

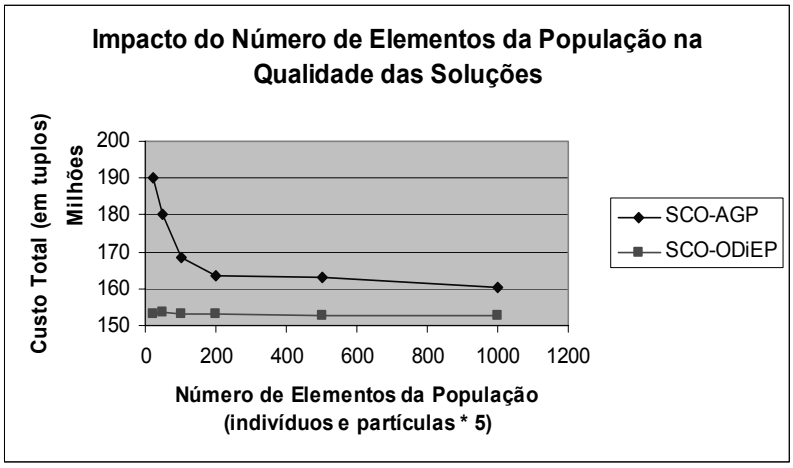


Figura 4.16. Impacto do número de elementos da população na qualidade das soluções obtidas.

Outra característica que importa avaliar é a escalabilidade de ambos os algoritmos. Para isso, usaram-se os cubos A e B (Tabela 2.1 e Tabela 2.3). Com esses dois *lattices*, executou-se um novo teste, com uma restrição espacial de 10% (relacionado com cada cubo), 200 iterações (gerações), um perfil de interrogações aleatório, 20 partículas / 100 indivíduos e mantendo-se o resto dos parâmetros usados no teste anterior.

Os resultados mostram-se na Tabela 4.4, com o correspondente gráfico da Figura 4.17. Um aumento de 4 vezes no número de subcubos implicou um aumento no tempo de execução de 7.13 para o SCO-AGS e de apenas 2.76 para o SCO-ODiEP. Isto mostra uma maior complexidade do primeiro e boas perspectivas de escalabilidade do segundo, com um aumento de tempo de execução de declive mais baixo do que o rácio de número de subcubos. Observou-se também que o SCO-ODiEP, quando o cubo B é utilizado, continua a obter soluções de melhor qualidade (13.5% melhores).

Tabela 4.4. Impacto do número de subcubos no tempo de execução do SCO-ODiEP e SCO-AGS.

	64 SubCubos	256 SubCubos	Rácio de Crescimento
SCO-AGP	4,945	35,253	7.13
SCO-ODiEP	4,058	11,200	2.76

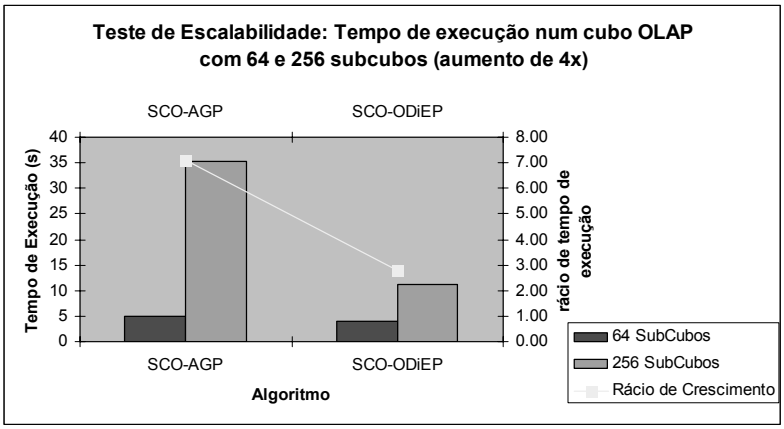


Figura 4.17. Escalabilidade do SCO-AGP e SCO-ODiEP.

A velocidade de ambos os algoritmos na obtenção das soluções é mostrada na Figura 4.18, utilizando o cubo B (o de maior complexidade e, portanto, com uma presumivelmente maior dificuldade de obtenção de soluções). Os valores mostrados referem-se à diferença percentual relativa ao valor do custo da qualidade final atingida. Pode ver-se que os dois algoritmos mostram uma rápida evolução para boas soluções, conseguindo chegar a uma vizinhança 2-3 % de diferença da solução final, logo cerca das 50 iterações (ou gerações). Também neste caso o SCO-ODiEP ultrapassa claramente o SCO-AGS: o primeiro entra numa vizinhança de 10% logo após 5 iterações, contra 29 para o segundo.

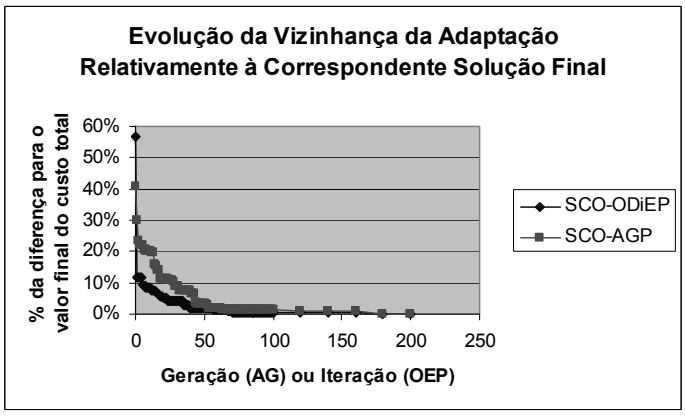


Figura 4.18. Velocidade de pesquisa de soluções para o SCO-ODiEP e SCO-AGP.

4.11 Avaliação Crítica do Algoritmo SCO-ODiEP

Neste capítulo foi proposto um novo método de selecção de cubos em ambiente OLAP centralizado que utiliza optimização por enxame de partículas - versão discreta, denominado SCO-ODiEP. Foram concebidos e descritos também outros dois algoritmos: SCO-AGS, baseado numa heurística *greedy* e o SCO-AGP, baseado num algoritmo genético padrão, ambos, embora com algumas variações, utilizando métodos de optimização já profusamente utilizadas para resolução do mesmo problema, sendo aqui aplicados para efeitos de análise comparativa face ao SCO-ODiEP.

O problema da selecção dos cubos foi descrito de uma forma mais apropriada, sendo igualmente descritos os três métodos de optimização utilizados, permitindo a fácil compreensão da forma como foi depois embutido na heurística de cada um dos métodos, especialmente no que respeita à codificação do problema no AG e OEP discreta.

A avaliação experimental efectuada parece mostrar a eficácia da aplicação da OEP discreta ao problema da selecção de cubos, dado um perfil de interrogações e um constrangimento especial, procurando minimizar o custo de interrogação e manutenção. A avaliação comparativa do SCO-ODiEP face aos dois outros algoritmos mostrou uma clara vantagem face ao SCO-AGR para espaços de materialização acima de 4% e uma vantagem, ainda que não tão marcada, face ao SCO-AGS, considerando a qualidade das soluções obtidas.

Mais do que um bom desempenho na qualidade das soluções propostas, o SCO-ODiEP não teve dificuldades em obter bons e sólidos resultados. Evidenciou também uma boa escalabilidade, já que o seu tempo de execução cresce linearmente com o número de partículas com um baixo declive, e mesmo o número de subcubos do cubo e subcubos seleccionados tem um impacto reduzido no tempo de execução (por exemplo 13 subcubos em 17.7 segundos x 36 subcubos em 25.9 segundos). Relativamente a este particular, verificou-se que o SCO-AGR apresentou o tempo de execução mais rápido, mas não é sustentável em casos de maior complexidade, dada a sua complexidade exponencial com o número de subcubos do cubo. Comparativamente, o SCO-ODiEP foi o algoritmo que mostrou uma complexidade mais baixa, como foi mostrado pela análise do gráfico da Figura 4.17.

Viu-se também que o SCO-ODiEP, com uma pequena população de partículas e um número reduzido de iterações, conseguiu boas soluções. Todas estas evidências mostraram uma grande "reserva de poder" do SCO-ODiEP, capaz de lidar com esquemas dimensionais mais complexos.

Especialmente para o caso dos dois algoritmos baseados em população, diversas variantes podem ser intentadas (por exemplo hibridação genética, variante cooperativa e multi-fase do OEP e a variante

co-evolucionária do AG), com expectáveis impactos positivos no seu desempenho. Já no próximo capítulo, algumas dessas variações vão ser testadas, até porque vai passar a lidar-se com um problema intrinsecamente mais complexo, pois que o âmbito do repositório OLAP vai ser distribuído, havendo assim um número N vezes superior de subcubos possíveis, onde N é o número de nós da arquitectura M-OLAP.

Capítulo 5

Selecção de Cubos em OLAP Distribuído - Arquitectura M-OLAP com Modelo de Custos Linear

5.1 O Problema da Selecção e Alocação de Cubos em Ambientes OLAP Distribuídos

O âmbito da arquitectura OLAP é aqui alargado segundo a dimensão espacial. Dispondo-se de uma arquitectura OLAP de nós múltiplos (Figura 3.5) os subcubos podem ser materializados e mesmo replicados em qualquer dos nós. O problema da selecção dos cubos definido na secção 4.6 tem de ser adaptado à nova realidade. O constrangimento espacial e perfil de consultas é agora individualizado por nó, importando definir claramente o problema a tratar. A definição 5.1 expõe o problema de forma clara.

Definição 5.1: Selecção de M em OLAP distribuído com constrangimento espacial

Seja $I = \{I_{11}, I_{12}, ..., I_{Nn}\}$ o conjunto de interrogações com frequências de acesso $F = \{F_{11}, F_{12}, ..., F_{Nn}\}$ e extensões de interrogação $Ei = \{Ei_{11}, Ei_{12}, ..., Ei_{Nn}\}$; seja a frequência de actualização e extensão de actualização Fm e $Em = \{Em_1, Em_2, ..., Em_n\}$ respectivamente e seja S_{N_i} o espaço disponível para materialização de subcubos em cada nó OLAP *i*. A solução para o problema de selecção do cubo a materializar é um conjunto $M = \{su_{11}, su_{21}, ..., su_{nN}\}$ com constrangimento $\sum_j |su_{jN_i}| \leq S_{N_i}$, tal que o custo de resposta às interrogações I e de manutenção de M, $Ci(I, M) + Cm(M)$, seja mínimo.

Para estimativa de $Ci(I, M)$ e $Cm(M)$, vai utilizar-se o modelo linear distribuído descrito em 3.2.1, e algoritmos de estimativa de custos descritos anteriormente nas secções 3.2.2 e 3.2.3, pois que correspondem à definição dos custos especificados na definição 5.1. Os custos estimados por estes algoritmos serão utilizados como cálculo do benefício do algoritmo *greedy* e função de aptidão dos AG e ODiEP, tal como no capítulo anterior.

5.2 Variantes dos Algoritmos Genéticos e de Optimização por Enxame de Partículas

No capítulo anterior, introduziu-se uma nova heurística de optimização em problemas combinatórios de carácter *NP-hard*: optimização por enxame de partículas na sua versão discreta, aplicada, neste caso, ao problema da selecção de cubos em ambiente OLAP centralizado. A utilização de abordagens genéticas e *greedy* no mesmo problema foram igualmente descritas e avaliadas comparativamente com a anterior.

O problema a solucionar foi agora alargado, uma vez que é considerada agora uma arquitectura M-OLAP. As mesmas heurísticas podem ser utilizadas, mas, dada a natureza do problema (a existência de vários NSOs), abordagens múltiplas farão, porventura, sentido, sob a forma de populações múltiplas de agentes de busca. Na verdade, a inclusão de novos nós faz aumentar

correspondentemente a complexidade do problema a resolver. Cada população pode encarregar-se da optimização da selecção e alocação de subcubos em cada um dos nós. Esta abordagem é extensiva aos algoritmos genéticos, surgindo como a variante co-evolucionária [Poter & Jong, 1994], e algoritmos de enxame de partículas, onde a abordagem cooperativa foi proposta para a variante contínua [Bergh & Engelbrecht, 2004], mas aqui estendida à versão discreta. Para este último método de optimização, uma outra variante foi igualmente proposta: a variante multi-fase [Al-Kazemi & Mohan, 2002].

5.2.1 Versão Co-Evolucionária do Algoritmo Genético (Co-Evol-AG)

Na secção 4.4, ao descrever-se um algoritmo genético e a forma de codificação do problema e de avaliar cada solução, foi fácil compreender que, havendo uma relação da complexidade do problema com o tamanho do genoma, um aumento da complexidade daquele, fazia aumentar o tamanho deste. Consequentemente deverá assistir-se a uma maior dificuldade na obtenção de boas soluções. Veja-se porquê: uma vez que a avaliação é global, uma melhoria num gene do cromossoma pode ser submergida por degradações em outro ou outros, e o gene melhorado pode ser perdido, uma vez que o indivíduo respectivo terá uma menor probabilidade de ser seleccionado. A situação é visível na Figura 5.1. À esquerda, nessa figura, as evoluções positivas observadas no primeiro e segundo genes do indivíduo um e dois, respectivamente, foram submergidas pela degradação havida em um ou mais dos outros genes desses indivíduos, já que a avaliação é efectuada globalmente. Mas esta situação será diversa se cada gene for o genoma de um indivíduo: qualquer melhoramento pode gerar descendência e a sua perpetuação. Há que pagar um custo adicional correspondente ao mais elevado número de avaliações de aptidão (tantas vezes maior quantas as espécies), mas o balanço final pode ser positivo.

Esta constatação fez surgir a ideia da denominada Abordagem Cooperativa Co-Evolucionária [Poter & Jong, 1994] do algoritmo genético (Co-Evol-AG). A sugestão para a solução advém da percepção de que, para a evolução de estruturas progressivamente mais complexas, noções explícitas de modularidade precisam de ser introduzidas, de maneira a proporcionar uma oportunidade razoável de evolução a soluções complexas, sob a forma de sub-componentes interactuantes co-adaptados. Na prática:

1. vai haver não uma população de soluções, mas várias subpopulações (denominadas espécies), cada uma representando um sub-componente da solução potencial;

2. uma solução completa é obtida por congregação de membros representativos de cada uma das espécies presentes (gerando-se um denominado "genoma completo" [Poter & Jong, 1994] ou "genoma global");
3. a atribuição de crédito a cada espécie é definida em termos da adaptação das soluções completas nas quais membros da espécie participam;
4. a evolução de cada espécie é efectuada por um algoritmo genético padrão.

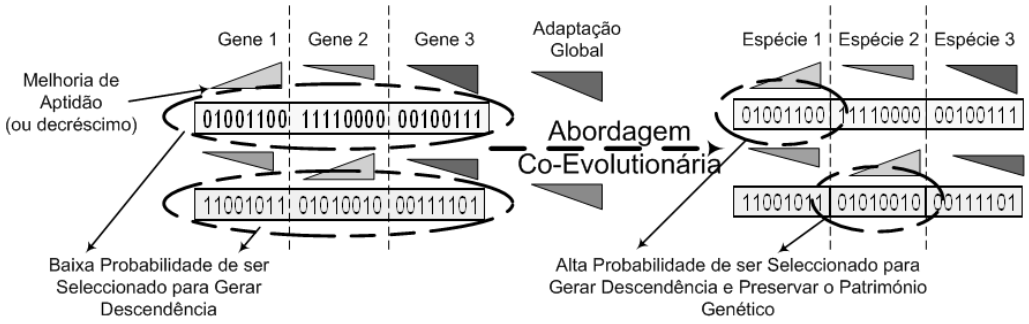


Figura 5.1. Abordagem Co-Evolucionária onde a população é dividida em subpopulações, conhecidas como espécies.

A avaliação da aptidão referida em 3) tem de ser efectuada globalmente, ou seja, é necessário congregar genomas de indivíduos de cada uma das espécies, de acordo com 2). Recordando o esquema de operação de um algoritmo genético esquematizado na Figura 4.1 e mostrado no Algoritmo 4.2, há a considerar duas fases principais: a geração da população e a evolução. Por outro lado, a eleição de membros representativos deve ser necessariamente diversa nas duas fases, já que ao ser gerada a população há que avaliá-la e, inicialmente, não há qualquer informação relativamente à qualidade de qualquer das soluções e, assim, não se dispõe de uma forma de "eleger" os membros mais representativos. Na fase da evolução, poderá já adoptar-se uma política relacionada com a aptidão de cada indivíduo. Desta discussão resulta que a criação do genoma para avaliação de cada indivíduo vai ser realizada distintamente para cada fase: 1) na fase inicial, quando é gerada a população (geração 0), o genoma para avaliação é formado adicionando ao genoma do indivíduo (no *locus* correspondente à sua espécie) o genoma seleccionado aleatoriamente de um indivíduo de cada uma das outras espécies (Figura 5.2, à esquerda), que encaixarão nos respectivos *locus*; 2) na fase evolucionária, o genoma global utiliza não indivíduos aleatórios de outras espécies, mas o melhor indivíduo de cada uma (encaixando no *locus* correspondente). E é precisamente aqui que a

abordagem co-evolucionária pode revelar algumas limitações. Se houver algumas inter-dependências específicas, como é o caso, dadas as relações entre subcubos inter e intra-nós, uma solução melhor pode ser invalidada, se em outro nó o melhor indivíduo implicar custos de manutenção mais elevados, devidos à melhoria da solução, sem ultrapassar a melhoria nos custos de interrogação. Como a evolução opera espécie a espécie, o algoritmo vai cair num ciclo de nó cego. Um melhoramento no melhor indivíduo da espécie E1 é invalidado pelo melhor indivíduo de E2 e reciprocamente. Então, torna-se impossível ao algoritmo "visitar" algumas das áreas e a pesquisa de áreas promissoras novas vai sendo progressivamente restringida, tendo como consequência a ocorrência de fenómenos de convergência prematura. O algoritmo cai em soluções sub-óptimas das quais é impossível sair (nem mesmo uma mutação cirúrgica casual permitirá ultrapassar o mínimo local, já que as mutações vão operar também ao nível da espécie).

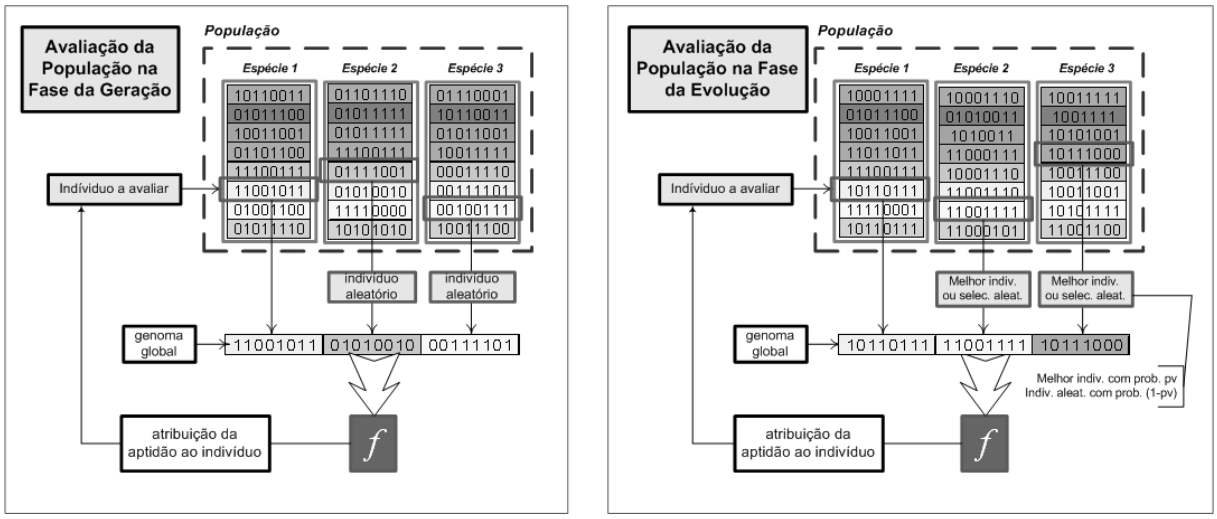


Figura 5.2. Mecanismo de geração do "genoma global".

Para resolver este problema, a selecção de indivíduos das outras espécies para formar o genoma global tem de ser menos determinista: implementar um mecanismo do tipo utilizado na fase inicial seria uma solução, com selecção aleatória de indivíduos. Mas este mostrou-se, compreensivelmente, lesivo do desempenho do algoritmo, originando fenómenos de não convergência. Assim, adoptou-se uma solução intermédia, introduzindo um mecanismo probabilístico (Figura 5.2, à direita). Assim, com uma probabilidade *pv*, o melhor indivíduo da espécie é seleccionado; com uma probabilidade *1-pv*, é seleccionado um indivíduo aleatório dessa mesma espécie. Esta probabilidade *pv* vai funcionar como

um patamar cuja variação permite que o "mecanismo de geração do genoma global" esteja mais ou menos perto da proposta de [Poter & Jong, 1994].

Outra modificação foi ainda realizada no algoritmo co-evolucionário original, além da acabada de descrever: introduziu-se um mecanismo de cruzamento inter-específico, com produção híbrida probabilística, tendente a gerar uma maior diversidade genética.

5.2.2 Variantes Cooperativas (ODiEP-C) e Multi-Fase (ODiEP-MF) da Versão Discreta da Optimização por Enxame de Partículas (ODiEP)

O aumento de complexidade do problema, que fez pensar na utilização de uma variante co-evolucionária do AG, também induziu uma direcção semelhante quanto à ODiEP. Muitas variações foram propostas à versão OEP base, nomeadamente 1) a inclusão de cooperação [Bergh & Engelbrecht, 2004] e 2) multi-fase com *hill-climbing* [Al-Kazemi & Mohan, 2002]. Algumas nuances e hibridações foram igualmente propostas, tais como 3) hibridação genética [Løvbjerg et al., 2001], [Angeline, 1998] e 4) extinção em massa [Xie et al., 2002].

O número de enxames é a principal diferença entre a variante cooperativa da optimização discreta por enxame de partículas (ODiEP-C) e a versão original da OEP: vários na primeira e um único na segunda. O seu nome surgiu em [Bergh & Engelbrecht, 2004], onde "enxames cooperativos" mostram a cooperação entre os diversos enxames. Aqui, é migrada para a versão discreta. Já foi dito que cada partícula "deambula" no espaço OEP multidimensional, procurando soluções. Na versão cooperativa, em vez de ser permitido a cada partícula voar em todo o espaço OEP, fica restrita às dimensões correspondentes a uma parte do problema (neste caso, as fronteiras do NSO). Mas, à semelhança do evidenciado no Co-Evol-AG, a aptidão de cada partícula pode ser calculada apenas em termos globais, ou seja, é necessário um esquema para gerar a posição global da partícula. Isto é conseguido com o denominado "vector contexto": a posição global virtual das partículas, onde a posição de cada partícula de cada enxame será incluída (nas dimensões correspondentes ao NSO). Detalhes acerca da forma de geração deste vector contexto serão dados um pouco mais à frente.

Na variante multi-fase da optimização discreta por enxame de partículas (ODiEP-MF) é proposta a divisão do enxame em vários grupos, cada um começando a sua evolução espacial em uma de várias fases possíveis, comutando entre fases, utilizando uma heurística adaptativa: a mudança de fase ocorre sempre que não se observar um melhoramento na aptidão global nas rI iterações mais

recentes. Tal como em [Al-Kazemi & Mohan, 2002], optou-se pela utilização do menor número de grupos e fases: dois para cada um. Neste caso, a equação 4.1 (secção 4.5.2) é substituída por:

$$\vec{v}_i^{k+1} = c_v.w\vec{v}_i^{k} + c_x.c2.rnd().\vec{s}_i^{k} + c_g.c2.rnd().gbest \qquad \text{eq. 5.1,}$$

na qual os sinais dos coeficientes (c_v, c_x e c_g) determinam a direcção do movimento das partículas. Olhando as duas equações (4.1 e 5.1), salta à vista, imediatamente, a inexistência do termo da eq. 4.1 correspondente a *pbest* (a razão será evidente já no parágrafo seguinte, ao discutir-se a utilização de *hill climbing*). Em cada instante, cada partícula pode estar numa de duas fases possíveis, determinada pela fase precedente e o número de iterações executadas até ao momento. Em cada fase, as partículas caem em grupos diferentes com coeficientes diversos para cada grupo. Nos testes efectuados, usaram-se, para a fase 1, (1, -1, 1) para o grupo 1 e (1, 1, -1) para o grupo 2. Na fase 2, os coeficientes são comutados. Na prática, numa fase, um grupo move-se em direcção a *gbest*, enquanto que o outro se move na direcção oposta.

Esta versão do optimizador utiliza também *hill climbing*, permitindo apenas mudanças da posição da partícula, se a nova localização implicar uma melhoria da aptidão. Desta forma, a posição corrente de cada partícula é necessariamente melhor do que a prévia; a posição de cada partícula é sempre *pbest*, compreendendo-se a inexistência deste termo na eq. 5.1. Mas, se o *hill climbing* pode ser interessante, ao apenas admitir melhores posições da partícula, sendo assim potencialmente benéfico do ponto de vista do desempenho global do algoritmo, há o reverso da medalha, na necessidade de um teste de aptidão para cada dimensão que possa ser actualizada. Em casos em que o número de dimensões é elevado (como no problema presente), isso requererá um número enorme de avaliações, o que implicará um tempo de execução incomportável. Desta forma, optou-se por uma solução de compromisso: só será feita uma avaliação de aptidão depois da actualização transitória de uma fracção *s* (um valor seleccionado aleatoriamente num intervalo *si=[1, sh]*, um parâmetro do algoritmo), de componentes consecutivos de velocidade. Além disso, a actualização só será efectiva se a alteração melhorar a aptidão. Procurando a focalização do algoritmo, quando o processo de busca se aproxima do fim, permite-se que *sh* decresça com o número de iterações do algoritmo.

Uma variação a este esquema foi tentada. Se nenhuma melhoria fosse conseguida com a evolução da partícula nas *s* dimensões, *s* era alargado e era executada a re-evolução da partícula nas *s* dimensões acrescidas. No entanto, esta modificação revelou um desempenho fraco e, assim, foi abandonada.

Os algoritmos ODiEP foram também dotados de competências genéticas, utilizando operadores de selecção e cruzamento. A selecção é implementada por simples substituição de partículas. Em cada

iteração, um número especificado *S* de partículas é substituído por um igual número de outras partículas com melhor aptidão, previamente clonadas. O processo de substituição pode ser executado de diversas formas, com um nível de elitismo [Mitchel, 1996] decrescente:

1. tomando aleatoriamente duas partículas em *S* , substituindo aquela que revela menor aptidão pela partícula *gbest*;
2. substituindo uma qualquer partícula aleatória em *S* por *gbest*, e, para as restantes, seleccionar aleatoriamente cada par de partículas, substituindo aquela que mostrou menor aptidão por uma qualquer terceira partícula;
3. buscar aleatoriamente pares de partículas em *S* e substituir a menos apta em cada par pela sua parceira.

Em qualquer dos métodos descritos, o *pbest* original de cada partícula é preservado, tornando-se a posição inicial da nova partícula.

Já quanto ao processo de cruzamento, a posição e a velocidade são cruzadas sempre entre dimensões, correspondendo ao mesmo nó. Considerando todo o enxame, com *pr* (*probabilidade de reprodução*), uma partícula é marcada para reprodução. Do grupo de partículas marcadas para reprodução (*gr*), um par é tomado aleatoriamente (pai1 e pai2), aplicando-se as seguintes fórmulas para gerar a posição e velocidade de filho1 e filho2:

$$\begin{aligned} filho_1(x_i) &= p_i.pai_1(x_i) + (1-p_i).pai_2(x_i) \\ filho_2(x_i) &= p_i.pai_2(x_i) + (1-p_i).pai_1(x_i) \end{aligned} \quad \text{eq. 5.2,}$$

em que p_i é um valor uniformemente distribuído $\in [0,1]$.

$$\begin{aligned} filho_1(v_i) &= pai_1(v_i) + pai_2(v_i) / |\, pai_1(v_i) + pai_2(v_i) \,|\, .|\, pai_1(v_i) \,| \\ filho_2(v_i) &= pai_1(v_i) + pai_2(v_i) / |\, pai_1(v_i) + pai_2(v_i) \,|\, .|\, pai_2(v_i) \,| \end{aligned} \quad \text{eq. 5.3.}$$

A operação é repetida até que *gr* esteja vazio. As partículas geradas substituem os pais, mantendo-se assim a população constante.

A extinção em massa [Xie et al., 2002] é a simulação artificial das extinções em massa que desempenharam um papel chave na história da vida na Terra. No algoritmo, ela é executada simplesmente por reinicialização das velocidades de todas as partículas com uma periodicidade ditada por um intervalo de extinção (*Ie*), um determinado número de iterações.

5.3 Algoritmo Greedy M-OLAP

O algoritmo Greedy M-OLAP é uma extensão multi-nó do algoritmo SCO-AGR (ver secção 4.7). A sua descrição formal apresenta-se em Algoritmo 5.1. Este algoritmo é uma adaptação do algoritmo *distributed node set greedy* proposto em [Bauer & Lehner, 2003], considerando o modelo linear distribuído e podendo utilizar duas métricas de benefício: ganho e densidade de ganho. Vai ser usado como termo de comparação de desempenho nos testes experimentais.

```
Input: L  // lattice com todas as combinações de granularidade, dependências,
          // tamanhos, dimensões e hierarquias
       I={I1, ..., In}, Ei={Ei1, ..., Ein}, Em={Em1, ..., Emn}, Fm, E=(E1... En.)
          // perfil de interrogações e actualizações: frequência e extensão
       Pb // parâmetros base: P=(P1... Pn.); X=(X1... Xn.), S=(S1... Sn.)
          // cap. Process. e espaço por nó e parâmetros de conexões
Output: M  // conjunto de subcubos a materializar e sua localização

Begin
 1. Inicialização:
  Inicializar parâmetros e serviços do sistema; // carregar perfil de carga,
   // lattice e esquema dimensional
  M ← { }; // inicializar com todos os nós vazios;
   // no nó 0 residem as relações base: tamanho = 3x o subcubo com menor granularidade
 2. Procurar sucessivos subcbos a materializar:
  Enquanto (∃En > 0 E ∃s : B(M ⋃ {c}) > B(M))  Fazer: // enquanto houver espaço
   // disponível num qq. nó e um subcubo s capaz de conferir benefício
   sopt ←   Ø ; B ( { sopt },M) : ←  0; // procurar de um subcubo s em qualquer nó
    // com o benefício máximo
   ParaCada nó n Fazer:
    ParaCada (s ∈ {L − M} )  Fazer: // para cada subcubo ainda não
     // materializado no nó n, calcular o ganho para cada um dos seus descendentes
     ParaCada ( i ∈ { descendente (s) } ) Fazer:
        B ({s},M,P,X) ← B({s}, M,P,X) + (C (i, M,P,X) − C (i, M ⋃{c},P,X));
     Próximo i
     B ({s},M,P,X) ← B({s}, M,P,X) − CM(s, M,P,X); // subtrai custos de
        // manutenção de s
    Próximo s
    Se (B{s}, M,P,X) > B({sopt}, M,P,X) Então // testa s como o subcubo
     // que proporciona o ganho máximo
     sopt ←  s;
   Próximo n
   Se ( Sn − Tamanho( {sopt} ) > 0) Então // se houver espaço disponível no nó n
    M ←  M ⋃ {sopt}; En ←  En − Tamanho( {sopt}); //  adiciona o subcubo óptimo a M
     // e actualiza o espaço disponível no NSO n
  FimEnquanto
 3. Devolver o resultado:
  Return (M)
End
```

Algoritmo 5.1. Algoritmo Greedy M-OLAP.

Basicamente, este algoritmo utiliza o *lattice* de agregação distribuído, o limite de armazenamento para materialização de subcubos por NSO, o perfil de interrogações e manutenção, e os parâmetros de processamento e comunicação como entradas. Enquanto houver espaço de materialização disponível num qualquer nó, o algoritmo vai seleccionar o par NSO e subcubo cuja materialização assegure, em cada momento, o maior benefício. Além disso, foi incluída uma estratégia adicional para resolver situações onde há dois ou mais subcubos que conduzem a benefícios iguais: seleccionar aquele residente no NSO com maior espaço de materialização ainda disponível.

5.4 Abordagem Evolucionária e Co-Evolucionária

5.4.1 Mapeamento do Problema no Genoma

A distribuição de subcubos a materializar *M* é agora distribuída pelos vários nós da arquitectura M-OLAP. Assim, em cada nó da arquitectura, cada subcubo pode (representado por um "1" no genoma) ou não (um "0" do genoma) estar materializado. Na Figura 5.3, onde cada *lattice* tem oito subcubos, ter-se-á, então, um número total de subcubos possíveis de 8 * 3 = 24, ou seja, genericamente, um número de subcubos total de

$$NSubC = nL.nNos \qquad \text{eq. 5.4,}$$

onde nL é o número de subcubos do *lattice* e nNos, o número de NSOs. A este *M* corresponderá um número idêntico de bits na cadeia binária do genoma de cada indivíduo do algoritmo genético padrão (na Figura 5.3, em cima, à esquerda). Para o caso do mapeamento no algoritmo co-evolucionário, a questão é ainda mais directa: cada M', correspondente aos subcubos a materializar em cada NSO, é representado pelo genoma do indivíduo de cada espécie, situação mostrada na Figura 5.3 (em cima, à direita).

Em resumo, o genoma de cada indivíduo do algoritmo genético padrão será uma cadeia de bits, em que, cada um vai representar um subcubo do M distribuído. Já o genoma do indivíduo de cada espécie do algoritmo genético co-evolucionário (uma vez restrito às fronteiras de cada NSO) irá ser uma cadeia binária com um número de bits igual ao número de subcubos possíveis a materializar em cada NSO.

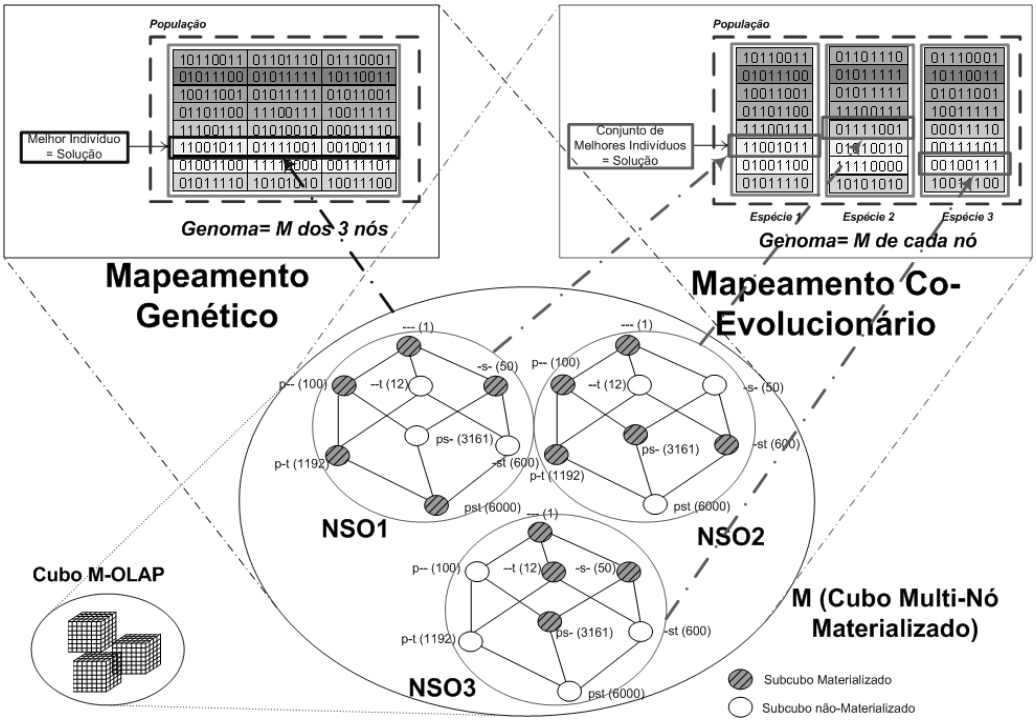


Figura 5.3. Codificação genética do problema da selecção e alocação de cubos no genoma do algoritmo genético padrão e co-evolucionário.

5.4.2 Algoritmos Genéticos Propostos

O algoritmo Genético Co-Evolucionário para a Selecção e Alocação de Cubos numa Arquitectura M-OLAP (Co-Evol-AG M-OLAP), a versão co-evolucionária do algoritmo genético, apresenta-se em Algoritmo 5.2.

O algoritmo para a versão padrão não é apresentado, pois que, por um lado, é semelhante ao mostrado em Algoritmo 4.7. Por outro lado, é também uma simplificação do agora mostrado, eliminando as estruturas relativas à geração do genoma global e ao ciclo adicional, correspondente à evolução de cada espécie.

Olhando o Algoritmo 5.2, podem ver-se distintamente as suas três fases de execução:

1. a geração da população inicial de cada espécie com a avaliação de cada indivíduo com geração do genoma global, utilizando o esquema descrito na secção 5.2.1, correspondente à fase inicial;

2. o ciclo correspondente à evolução das espécies, onde a avaliação é efectuada utilizando o esquema correspondente à fase da evolução descrito na secção 5.2.1;
3. a disponibilização da solução encontrada, correspondente, na prática, ao genoma global do final da última geração.

```
Input: L  // lattice com todas as combinações de granularidade, dependências,
          // tamanhos, dimensões e hierarquias
       I={I1, ..., In}, Ei={Ei1, ..., Ein}, Em={Em1, ..., Emn}, Fm, E=(E1... En.)
          // perfil de interrogações e actualizações: frequência e extensão
       Pg // parâmetros do algoritmo genético: NumIndiv, NumGerac, TpSelec,
          // PatamarSelecTpC, TxMutac, ProbMutac
       Pb // parâmetros base: P=(P1... Pn); X=(X1... Xn.), S=(S1... Sn.), Tr
          // cap. process. e espaço por nó, parâmetros de conexões e tipo de reparação
Output: M  // conjunto de subcubos a materializar e sua localização

Begin
 1. Inicialização:
  Inicializar parâmetros e serviços do sistema; // carregar parâmetros genéticos,
   // perfil de carga, lattice e esquema dimensional
  Pope ← { }; // população de indivíduos vazia (uma população por cada espécie=ao
   // número de nós da arquitectura M-OLAP )
 2. Geração, reparação e avaliação da população inicial com mostra do estado:
  Para Cada espécie e Em Popn, Fazer: // vai gerar uma população para cada espécie
   // (o genótipo de cada espécie representa os subcubos a material. em cada NSO)
   Repetir NumIndiv Vezes: // vai gerar a pop. P de NumIndiv indivíduos e repará-la
    Gerar indivíduo i ; // cada indivíduo tem um genoma com n bits, em que
     // n=número de subcubos do lattice
    Se (Tamanho(i)>Sn) Então ReparaIndiv (i, Tr); // n=e; se indivíduo gerado
     // com  genoma cuja solução de materialização viola constrangimento,
     // vai repará-lo, aplicando o Algoritmo 4.5 ou Algoritmo 4.6
    Pope ← Pope ∪ i ;
   FimRepetir
  Próximo e
  // calcular a aptidão: é necessário gerar o genoma global
  Para Cada espécie e Em Pope, Fazer: // vai fazer o cálculo espécie a espécie
   Para Cada Indivíduo i Em Pe, Fazer:
    // gerar o genoma global=genoma do indivíduo em avaliação ∪
    // genoma de indivíduos aleatórios das outras espécies
    genoma_global ← gen(i); // gen(i) devolve o genoma do indivíduo a avaliar
    Para Cada espécie ee Em Pope \{e} Fazer:
     genoma_global ← genoma_global ∪ gen(aleat(Popee)); // gen(xx) devolve o
      // genoma de xx; aleat{yy} devolve um elemento aleatório de {yy}
    Próximo ee
    Fitness(i)= f (genoma_global, I, Ei, Fi, Em, Fm, X); // calcula a aptidão de
     // cada indivíduo gerado  ( f  calculada aplicando algoritmos (3.4 ou 3.5) e 3.6)
   Próximo i
  Próximo e
  MostrarEstado (Pope); // mostra estado instantâneo da população e melhor solução
   // gerada: quantos indivíduos propõem a materialização de um determinado subcubo
   // (algo indicativo da diversidade genética ) e melhor genoma, correspondendo ao M
   // que mostrou um menor custo
```

Algoritmo 5.2. Algoritmo Co-Evol-AG M-OLAP com geração semi-aleatória do genoma global.

Algoritmo 5.2, Co-Evol-AG M-OLAP, continuado da página anterior.

```
3. Evolução (sucessivas gerações):
 // gerada, reparada e mostrada a população inicial, segue-se a evolução
 Gerac←0; // contador do número de gerações
 Enquanto ( Gerac<NumGerac))  Fazer:  // a condição de finalização é o
  // número de gerações
  PontoCross← rnd (1, nScub(L)-2); // selecciona o ponto de crossover a usar para
   // esta geração: nScub(L) devolve o número total de subcubos no lattice; supõe-se
   // que o subcubo 0 é o primeiro, e portanto o último será nScub(L)-1
  Pop_e' ← { } // cria a nova população de espécies vazia
  Para Cada espécie e Em Pop_e, Fazer:
   Enquanto |Pop_e'|<=NumIndiv Fazer: // executar cruzamentos até gerar uma nova
    // população com número de indivíduos igual a P
    Pop_e' ← Pop_e' ∪ melhor (Pop_e); // princípio do elitismo: o melhor indivíduo
       // é preservado na nova população e fica disponível para futuros cruzamentos
    // Cruzamentos
    Pais←{ }; Filhos←{ } // cria pais e filhos: seleccionar dois pais da população
       // a ser objecto de cruzamento gerando dois filhos
    Repetir 2 Vezes:
     Pais←SelecIndiv (Pop_e, TpSelec, PatamarSelecTpC); // selecciona indiv.
       // conforme tipo de selecção (proporcional ou competição)
    FimRepetir
    Filhos←Crossover(Pais, PontoCross);  // gera os dois filhos, aplicando o
       // operador crossover de um ponto
    ParaCada filho Em Filhos, Fazer: // vai reparar cada filho gerado
     Se (Tamanho(filho)>S_n) Então ReparaIndiv (filho, Tr); // se indivíduo
       // gerado com  genoma cuja solução de materialização viola constrangimento,
       // vai repará-lo, aplicando o Algoritmo 4.5 ou Algoritmo 4.6
       P_e'←P_e' ∪ filho;
    FimPara
   FimEnquanto
   // Mutação
   Repetir NumIndiv*TaxMutac Vezes: // vai operar a mutação sobre um número
    // de indivíduos de TaxMutac
    IndivMutar← rnd (0, NumIndiv-1) // selecciona aleatoriamente o indivíduo
     // a sofrer mutação, supõe-se que o primeiro indivíduo é o 0
    Se metod_lidar_indiv_inval='R' Então // método de lidar com indivíduos
     // inválidos típico: depois da mutação, a reparação aplica a cada bit do
     // genoma do indivíduo a mutar a probabilidade de mutação
     // ProbMutac (probabilidade em 100)
     ParaCada bit b Em genoma (P_e', IndivMutar) Fazer:
      Se rnd (0,100)<ProbMutac Então
       Se b=0 Então b ←1; Senão b ←0;
     FimPara
     Se (Tamanho (P_e', IndivMutar)>S_n) Então
      ReparaIndiv ((P_e',IndivMutar), Tr); // n=e; se indivíduo mutado tiver
       // genoma cuja solução de materialização viole constrangimento, vai
       // repará-lo, aplicando o Algoritmo 4.5 ou Algoritmo 4.6
     SenãoSe metod_lidar_indiv_inval='M' Então // só permite mutações
       // que não violem constrangimento
      Fazer
      // aplica a cada bit do genoma do indivíduo a mutar a probabilidade de
      // mutação ProbMutac (probabilidade em 100)
       ParaCada bit b Em genoma (P_e', IndivMutar) Fazer:
        Se rnd (0,100)<ProbMutac Então
         Se b=0 Então b ←1; Senão b ←0;
       FimPara
```

Algoritmo 5.2, Co-Evol-AG M-OLAP, continuado da página anterior.

```
        Enquanto Tamanho (Pe', IndivMutar)>Sn
      FimRepetir
      // nova população gerada e reparada: aplicar operador substituição
      Pe←{ } // limpa a população antiga
      Para Cada Indiv Em Pe', Fazer:
       Pe←Pe ∪ Indiv;
      FimPara
      // avaliação da aptidão da nova população
      Para Cada Indivíduo i Em Pe, Fazer:
      // gerar o genoma global=genoma do indivíduo em avaliação ∪ genoma
      // de indivíduos aleatórios das outras espécies
       genoma_global ← gen(i); // gen(i) devolve o genoma do indivíduo a avaliar;
          // coloca em genoma_global o genoma do individuo a avaliar
       Para Cada espécie ee Em Pope \{e} Fazer: // adic. os genomas do melhor
        // ou um indiv. aleat. de outras espécies ao genoma do indiv. a avaliar
        genoma_global← genoma_global∪gen(indiv_max_fit(Popee OR aleat(Popee)));
          // gen(xx) devolve o genoma de xx; aleat{yy} devolve um elemento aleatório de
          // {yy}; indiv_max_fit(zz) devolve o indiv. com maior aptidão da espécie zz
       Próximo ee
       Fitness(i)=f (genoma_global, I, Ei, Fi, Em, Fm, X); // calcula a aptidão
        // de cada indiv. gerado ( f  calculada aplicando algortmos (3.4 ou 3.5) e 3.6)
      Próximo i
      ParaCada indivíduo i Em Pe, Fazer:
       Fitness(i)=f (genoma_global, I, Ei, Fi, M, Em, Fm, X); // calcula a
        // aptidão. de cada indivíduo gerado  ( f  calculada aplicando algoritmos (3.4
        // ou 3.5) e 3.6)
      Próximo i
      MostrarEstado (Pe); // mostra estado instantâneo da espécie e
       // melhor solução gerada
    Próximo e
    Gerac←Gerac+1; // incrementa o número da geração
  FimEnquanto
 4. Devolver o resultado:
  Return M (melhor (P)); // retorna o melhor genoma global conseguido
End
```

5.5 Optimização por Enxame de Partículas

5.5.1 Mapeamento do Problema no Espaço ODiEP

O mesmo raciocínio aplicado na secção 5.4.1 vai permitir mapear o problema da selecção e alocação do cubo no espaço ODiEP. A Figura 5.4 mostra o esquema implementado.

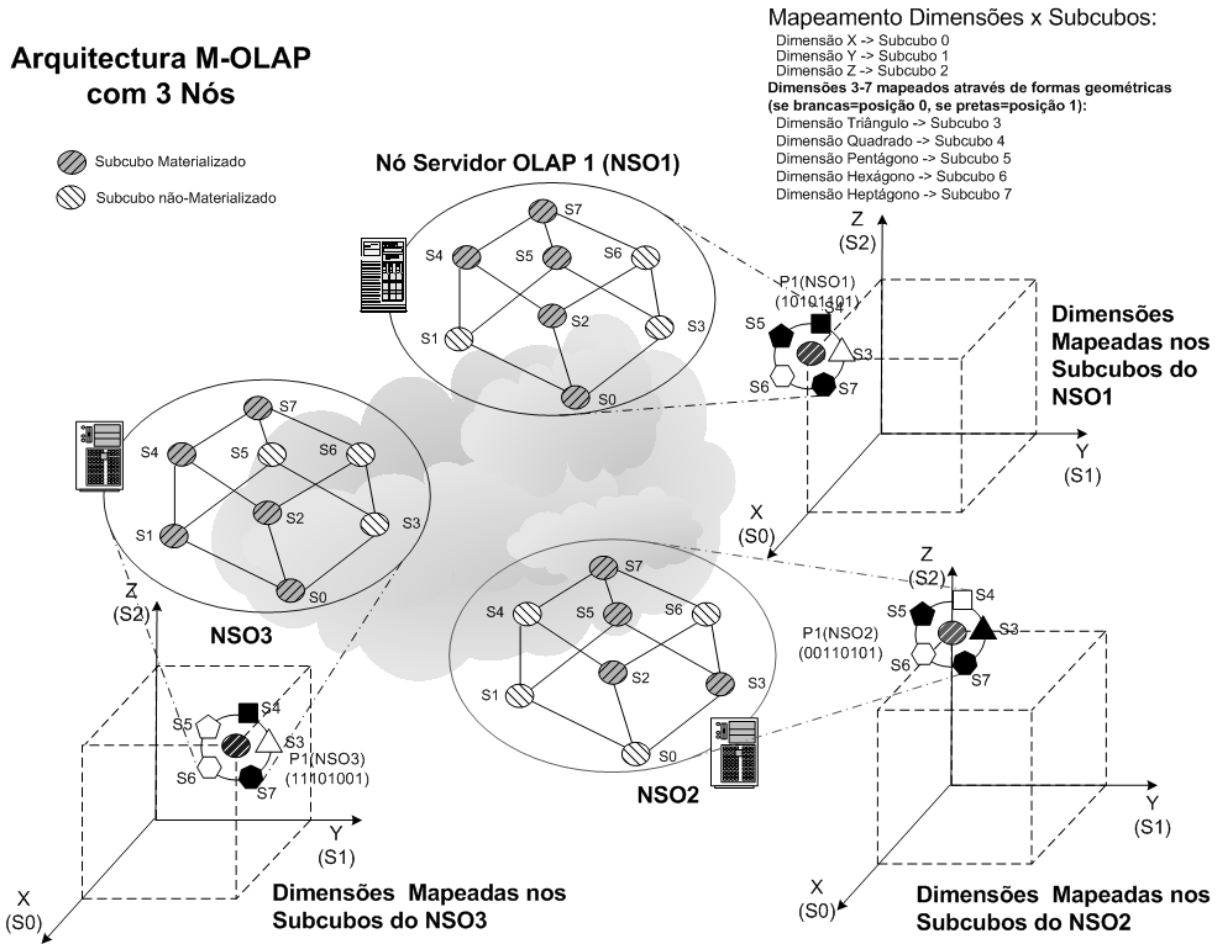


Figura 5.4. Mapeamento do problema da selecção e alocação do cubo num ambiente M-OLAP.

Por analogia, tem-se também um número total de subcubos dado pela eq. 5.4, neste caso, 24, já que cada *lattice* tem 8 subcubos possíveis. Como cada possível subcubo é mapeado para uma dimensão ODiEP, teremos: 1) Para a versão normal do ODiEP, 24 dimensões, onde cada grupo de 8 corresponderá aos subcubos possíveis em cada NSO; na figura, o conjunto dos três sistemas dimensionais representados perfarão o total das 24 dimensões. 2) Para o caso da versão multi-enxame (por exemplo, cooperativa), cada partícula mover-se-á num espaço ODiEP com oito dimensões, correspondendo então aos possíveis subcubos de um NSO. Na Figura 5.4, cada um dos sistema de eixos representado será o espaço ODiEP onde se movimentará cada um dos enxames.

5.5.2 Algoritmos ODiEP Propostos

Para a ODiEP, apresentam-se todos os algoritmos, pois que são agora incluídas hibridações genéticas e variante multi-fase, além da versão cooperativa do algoritmo original. Assim, mostram-se: no Algoritmo 5.3, a versão normal (ODiEP M-OLAP), no Algoritmo 5.4, a versão com enxames cooperativos (ODiEP-C M-OLAP) e no Algoritmo 5.5, a versão multi-fase com *hill climbing* e extinção em massa (ODiEP-MF M-OLAP).
Relativamente ao algoritmo ODiEP M-OLAP, não haverá grandes considerações adicionais a fazer. O número de dimensões do espaço OEP é agora incrementado pelo factor igual ao número de NSOs na arquitectura M-OLAP, como já se viu no mapeamento apresentado na Figura 5.4.

Uma análise comparativa com o Algoritmo 4.9 permite verificar que foram adicionados os módulos que implementam as hibridações genéticas. O primeiro implementa o operador *crossover*, implementado através da aplicação das eq. 5.2 e 5.3 e esquema descrito na secção 5.2.2. O ponto de *crossover* é gerado aleatoriamente, sendo os *crossovers* efectuados entre alelos correspondentes (neste caso, entre dimensões mapeadas para o mesmo NSO). Já o segundo implementa um esquema de selecção, procurando incluir o princípio da sobrevivência do mais apto. Diversas variantes foram implementadas que podem ser seleccionadas através de um parâmetro do algoritmo ODiEP. São elas:

1. Substituição elitista-2F executada em duas fases: a) é seleccionada uma partícula aleatória que é depois substituída por gbest (posição e velocidade); b) seleccionadas três partículas, uma das quais vai substituir a menos adaptada das outras duas.
2. Substituição competição-elitista: a menos adaptada de entre duas seleccionadas aleatoriamente é substituída por gbest.
3. Substituição competição: estando seleccionadas duas partículas aleatoriamente, a menos adaptada é substituída pela mais adaptada.

Quanto ao algoritmo ODiEP-C M-OLAP, mantém as hibridações genéticas, mas como versão cooperativa é, agora, multi-enxame. No entanto, o aumento de complexidade não é grande, já que o paradigma é perfeitamente adaptado ao caso: cada enxame é mapeado para um NSO. Há que gerar o vector contexto, para permitir a avaliação da posição de cada partícula e gerir os enxames múltiplos, mas, no restante, tudo é semelhante.

```
Input: L  // lattice com todas as combinações de granularidade, dependências,
          // tamanhos, dimensões e hierarquias
       I={I1, ..., In}, Ei={Ei1, ..., Ein}, Em={Em1, ..., Emn}, Fm, E=(E1... En.)
          // perfil de interrogações e actualizações: frequência e extensão
       Pe // parâmetros do algoritmo de optimização por enxame de partículas discreto:
          // NumPartic, NumIterac,. Vmax, wini, wfim, perc_cruz, perc_subs
       Pb // parâmetros base: S={S1, ..., Sn} (Espaço disponível para materialização),
          // Tr (tipo de reparação), nNSO (número de NSOs)
       P=(P1... Pn.); X=(X1... Xn.), E=(E1... En.)  // Capacidade de Processamento
          // e espaço por nó e parâmetros de conexões
Output: M  // conjunto de subcubos a materializar e sua localização
Begin
 1. Inicialização:
  Inicializar parâmetros e serviços do sistema; // carregar parâmetros OEP,
     // perfil de carga, lattice e esquema dimensional
  E←{ }; d←NSCubos(L)*nNSO // enxame vazio e d=número de dim. do espaço OEP discreto
     // (= número de subcubos possíveis*número de NSOs)
  D←{d1, ..., dd}; // espaço OEP d dimensional
 2. Geração, reparação, avaliação e mostra do estado do enxame inicial:
  Repetir NumPartic Vezes: // vai gerar o enxame com NumPartic partíc. e repará-lo
   p←GeraPartic(Vmax, d) // gera velocidade aleatoriamente (entre –Vmax e Vmax)
    // e aplica regra 4.2 para calcular posição para cada dimensão (igual ao número
    // de subcubos possíveis, dado o mapeamento do problema)
   ParaCada nó n Em M-OLAP, Fazer:
    pos_nó=pos(p, n); // pos(p,n) devolve as coordenadas da partícula
     // correspondente às dimensões mapeadas no nó n da arq. M-OLAP
    Se (Tamanho(M (pos_no))>Sn) Então ReparaPartic (pos_no, Tr); // se
     // partícula gerada em posição cuja solução de materialização viola
     // constrangimento, vai repará-lo, aplicando Algoritmo 4.8 ou correspondente
     // OEP de Algoritmo 4.6
   Próximo n
   E←E∪p;
   Fitness(p)=f (p, I, Ei, Fi, Em, Fm, X); // calcula a aptidão da partícula (f
    // calculada aplicando (3.4 ou 3.5) e 3.6)
   pbest(p) ← Fitness(p); // pbest inicial de cada partícula é a aptidão da
    // posição inicial da partícula
   Se Fitness(p)>gbest Então gbest ← Fitness(p); // calcula gbest relativo
    // ao enxame inicial
  FimRepetir
  MostrarEstado (E); // mostra estado instantâneo do enxame e gbest: quantas
   // partículas propõem a materialização de um determinado subcubo em cada nó da
   // arq. M-OLAP e gbest, correspondendo à melhor posição de uma qq. partícula
 3. Movimentação do enxame: // gerado, reparado e mostrado o enxame inicial,
   // segue-se a movimentação
  Iter←0; // contador do número de iterações
  Enquanto ( Iter<NumIter) Fazer:// condição de finalização é o número de iterações
   Actualiza w; // faz variar w de wini até wfim, com o número de iterações
   ParaCada p Em E, Fazer: // vai movimentar cada partícula, aplicando equações
    // e regras da dinâmica das partículas
```

Algoritmo 5.3. Algoritmo de Optimização Discreta com Enxame de Partículas para a Selecção e Alocação de Cubos numa Arquitectura M-OLAP (ODiEP M-OLAP).

Algoritmo 5.3., ODiEP M-OLAP, continuado da página anterior.

```
      ParaCada d Em D, Fazer:
       // calcular nova velocidade da partícula para dada dimensão d
       v_Pd^(iter+1) = w.v_Pd^iter + c1.rnd().(pbest_Pd - s_Pd^iter) + c2.rnd().(gbest - s_Pd^iter)   // aplicar eq. 4.1; s_Pd^iter é a posição da
         // partícula p na dimensão d na iteração iter
       Se v_Pd^(iter+1) > v_max Então v_Pd^(iter+1) = v_max SenãoSe v_Pd^(iter+1) < v_max Então v_Pd^(iter+1) = -v_max ; // aplicar regra 4.1
       Se se rnd() < sig(v_Pd^(iter+1)) Então s_Pd^(iter+1) = 1 Senão s_Pd^(iter+1) = 0 ; // calcular localização da partícula:
        // aplicar regra 4.2
      Próximo d
      ParaCada nó n Em M-OLAP, Fazer:
       pos_nó=pos(p, n), Fazer: // pos(p,n) devolve as coordenadas da partícula
        // correspondente às dimensões mapeadas no nó n da arq. M-OLAP
       Se (Tamanho(M (pos_no))>Sn) Então ReparaPartic (pos_no, Tr); // se
        // partíc. gerada em posição cuja solução de materialização viola constrangi/o
        // vai repará-lo, aplicando Algoritmo 4.8 ou correspondente OEP de Algoritmo 4.6
      Próximo n
     Próximo p
     Repetir NumIndiv*Perc_Cruz Vezes: // vai operar os cruzamentos
      nParticPai1 ← rnd (0, NumPartic-1);  nParticPai2 ← rnd (0, NumPartic-1);
       // selecciona os pais, supõe-se que a primeira partícula é 0
      filho1,filho2 ← crossing(E(nParticPai1), E(nParticPai2));
      ParaCada nó n Em M-OLAP, Fazer: // gerados os filhos, vai repará-los
       ParaCada filho Em {filho1, filho2}, Fazer:
        pos_nó=pos(filho, n), Fazer: // pos(p,n) devolve as coordenadas da
         // partícula corr. às dimensões mapeadas no nó n da arq. M-OLAP
        Se (Tamanho(M (pos_no))>Sn) Então ReparaPartic (pos_no, Tr); // repara
         // partícula (identicamente ao feito acima, na fase de inicialização)
       Próximo filho
      Próximo n
      E(nParticPai1) ← filho1; E(nParticPai2) ← filho2; // substituição dos pais
       // pelos filhos gerados
     FimRepetir
     ParaCada p Em E, Fazer: // calcula aptidão de cada partícula e
      // actualiza pbest e gbest
      Fitness(p) = f (p, I, Ei, Fi, Em, Fm, X); // calcula a aptidão de
       // cada partícula gerada
      Se Fitness(p)>pbest(p) Então pbest(p) ← Fitness(p); // se nova aptidão é
       // melhor do que a melhor aptidão atingida pela partícula actualiza pbest
      Se Fitness(p)>gbest Então gbest ← Fitness(p); // actualiza gbest relativo à
       // nova posição das partículas do enxame, se for o caso
     Próximo p
     Repetir NumIndiv*Perc_Subs Vezes: // vai operar as substituições
      Seleccão e clonagem com aplicação do operador substituição; // aplica
       // selecção e substituição utilizando um de entre vários esquemas
     FimRepetir
     MostrarEstado (E); iter++; // mostra estado actual e incrementa número
      // da iteração corrente
   FimEnquanto
  4. Devolver resultado:
   Return M (gbest); // retorna o M correspondente à melhor posição atingida por
    // uma qualquer partícula do enxame
End
```

```
Input: L  // lattice com todas as combinações de granularidade, dependências,
          // tamanhos, dimensões e hierarquias
       I={I1, ..., In}, Ei={Ei1, ..., Ein}, Em={Em1, ..., Emn}, Fm, E=(E1... En.)
          // perfil de interrogações e actualizações: frequência e extensão
       Pe // parâmetros do algoritmo de optimização por enxame de partículas discreto:
          // NumPartic, NumIterac,. Vmax, wini, wfim, perc_cruz, perc_subs
       Pb // parâmetros base: S={S1, ..., Sn} (Espaço disponível para materialização),
          // Tr (tipo de reparação), nNSO (número de NSOs)
       P=(P1... Pn.); X=(X1... Xn.), E=(E1... En.)  // Capacidade de Processamento
          // e espaço por nó e parâmetros de conexões
Output: M  // conjunto de subcubos a materializar e sua localização
Begin
 1. Inicialização:
   Inicializar parâmetros e serviços do sistema; // carregar parâmetros OEP,
      // perfil de carga, lattice e esquema dimensional
   E←{ }; nd←NSCubos(L) // há um enxame por cada NSO da arquitectura M-OLAP;
      // d=número de dim. do espaço OEP discreto
   D←{d1, ..., dd}; // espaço OEP d dimensional
 2. Geração, reparação, avaliação e mostra do estado do enxame inicial:
   ParaCada enxame e Em En, Fazer: // gerar os vários enxames
    Repetir NumPartic Vezes: // vai gerar o enxame com NumPartic partíc. e repará-lo
    p←GeraPartic(Vmax, nd) // gera velocidade aleatoriamente (entre -Vmax e Vmax)
      // e aplica regra 4.2 para calcular posição para cada dimensão (igual ao número
      // de subcubos possíveis, dado o mapeamento do problema)
    Se (Tamanho(M (p))>Sn) Então ReparaPartic (p, Tr); // n=e; se partícula
      // gerada em posição cuja solução de materialização viola constrangimento, vai
      // repará-lo, aplicando Algoritmo 4.8 ou correspondente OEP de Algoritmo 4.6
    Ee←Ee ∪ p;
   Próximo e
   // calcular a aptidão: é necessário gerar o vector contexto
   ParaCada enxame e Em En, Fazer: // vai fazer o cálculo enxame a enxame
    ParaCada partícula p Em e Fazer:
      // gerar o vector contexto=posição do enxame ∪ posição de partículas
      // aleatórias dos outros enxames
      vector_contexto ← pos(p); // pos(p) devolve a posição da partícula p
      ParaCada enxame ee Em En \{e} Fazer:
       vector_contexto ← vector_contexto ∪ pos(aleat(Eee)); // pos(xx) devolve a
         // posição de xx; aleat{yy} devolve uma partícula aleat. de {yy}
      Próximo ee
      Fitness(p)=f (vector_contexto, I, Ei, Fi, Em, Fm, X); // calcula a aptidão
       // da partícula gerada  (f  calculada aplicando (3.4 ou 3.5) e 3.6)
      pbest(p,e) ← Fitness(p); // pbest inicial de cada partícula é a aptidão
       // da posição inicial da partícula em cada enxame
      Se Fitness(p)>gbest(e) Então
       gbest(e) ← Fitness(p); // calcula gbest relativo a cada enxame inicial
      Se Fitness(p)>gbestGlob Então
       gbestGlob ← Fitness(p); gbestPosGlob ← vector_contexto // calcula gbest
         // relativo a cada enxame inicial e melhor posição global
    Próximo p
   Próximo e
```

Algoritmo 5.4. Algoritmo de Optimização Discreta com Enxames de Partículas Cooperativos para a Selecção e Alocação de Cubos numa Arquitectura M-OLAP (ODiEP-C M-OLAP) com geração semi-aleatória do vector contexto.

Algoritmo 5.4., ODiEP-C M-OLAP, continuado da página anterior.

```
  vextor_contexto ← gbestPosGlob; // o vector contexto que inicia a fase de
    // movimentação é aquele que assegurou o melhor gbest
  MostrarEstado (Ee); // mostra estado instantâneo dos enxames e melhor
    // vector_contexto: quantas partículas propõem a materialização de um determinado
    // subcubo em cada nó da arq. M-OLAP e melhor vector_contetxo, correspondendo à
    //melhor posição global de uma qq. partícula
3. Movimentação do enxame: // gerados, reparados e mostrados os enxames iniciais,
    // segue-se a movimentação
  Iter←0; // contador do número de iterações
  Enquanto (Iter<NumIter)  Fazer: //condição de finalização é o número de iterações
    Actualiza w; // faz variar w de wini até wfim, com o número de iterações
    ParaCada enxame e Em En, Fazer: // vai movimentar cada enxame
      ParaCada p Em En, Fazer: // n=e; vai movimentar cada partícula, aplicando
        // equações e regras da dinâmica das partículas
        ParaCada d Em D, Fazer:
          // calcular nova velocidade da partícula para dada dimensão d
          v_Pd^(iter+1) = w.v_Pd^iter + c1.rnd().(pbest_Pd − s_Pd^iter) + c2.rnd().(gbest − s_Pd^iter)  // aplicar eq. 4.1; s_Pd^iter é a pos. da
            // partícula p na dimensão d na iteração iter
          Se v_Pd^(iter+1) > v_max Então v_Pd^(iter+1) = v_max SenãoSe v_Pd^(iter+1) < v_max Então v_Pd^(iter+1) = −v_max ; // aplicar regra 4.1
          // calcular localização da partícula
          Se se rnd() < sig(v_Pd^(iter+1)) Então s_Pd^(iter+1) = 1 Senão s_Pd^(iter+1) = 0 ; // aplicar regra 4.2
        Próximo d
        Se (Tamanho(M (p))>Sn) Então ReparaPartic (p, Tr); // repara partícula,
          // reposiciona-a em posição do espaço válida
      Próximo p
      Repetir NumIndiv*Perc_Cruz Vezes: // vai operar os cruzamentos
        nParticPai1 ← rnd (0, NumPartic-1); // selecciona os pais,
        nParticPai2 ← rnd (0, NumPartic-1); // supõe-se que a primeira partícula é 0
        filho1,filho2 ← crossing(En (nParticPai1), En (nParticPai2));
        ParaCada filho Em {filho1, filho2}, Fazer:
          Se (Tamanho(M (filho))>Sn) Então ReparaPartic (filho, Tr); // repara
            // partícula (identicamente ao feito acima, na fase de inicialização)
        Próximo filho
        En(nParticPai1) ← filho1; En (nParticPai2) ← filho2; // substituição dos
          // pais pelos filhos gerados
      FimRepetir
    Próximo e
    // calcular a aptidão: é necessário colocar a posição de cada partícula
    // na dimensões correspondentes ao vector contexto, gerando vector_aval
    ParaCada enxame e Em En, Fazer: // vai fazer o cálculo enxame a enxame
      ParaCada partícula p Em e Fazer:
        vector_aval ← vector_contexto ∩ pos(p); // pos(xx) devolve a pos. de xx;
        Fitness(p) = f (vector_aval, I, Ei, Fi, Em, Fm, X); // calcula a aptidão da
          // partícula gerada  ( f  calculada aplicando (3.4 ou 3.5) e 3.6)
        Se Fitness(p) >pbest(p,e) Então pbest(p,e) ← Fitness(p); // actualiza
          //  pbest da partícula no enxame
        Se Fitness(p)>gbest(e) Então gbest(e) ← Fitness(p); // act.gbest do enxame
        Se Fitness(p)>gbestGlob Então
          gbestGlob ← Fitness(p); vector_contexto ← vector_aval; // actualiza
            // gbest global e vector contexto
      Próximo p
```

Algoritmo 5.4., ODiEP-C M-OLAP, continuado da página anterior.

```
     Repetir NumIndiv*Perc_Subs Vezes: // vai operar as substituições
       Seleccão e clonagem com aplicação do operador substituição; // aplica
         // selecção e substituição utilizando um de entre vários esquemas
     FimRepetir
    Próximo e
    MostrarEstado (En); // mostra estado actual
    iter++; // incrementa número da iteração corrente
   FimEnquanto
  4. Devolver resultado:
   Return M (vector_contexto); // retorna o M corr. ao melhor vector contexto
End
```

Para finalizar esta secção, importa descrever informalmente o algoritmo ODiEP-MF M-OLAP, já que inclui características novas: enxames em fases múltiplas, *hill climbing* e extinção em massa.
Olhando o Algoritmo 5.5, percebe-se o aumento de complexidade face ao Algoritmo 5.3, ainda que não excessivo. Na verdade, se compararmos o tamanho (uma medida grosseira de complexidade), este é muito semelhante ao Algoritmo 5.3. As fases 1 e 2 do algoritmo são iguais ao Algoritmo 5.3. A fase das iterações é que inclui agora a movimentação multi-fase, o controlo do *hill climbing* e a extinção em massa (a sombreado mais escuro).

```
Input: L  // lattice com todas as combinações de granularidade, dependências,
          // tamanhos, dimensões e hierarquias
       I={I1, ..., In}, Ei={Ei1, ..., Ein}, Em={Em1, ..., Emn}, Fm, E=(E1... En.)
          // perfil de interrogações e actualizações: frequência e extensão
       Pe // parâmetros do algoritmo de optimização por enxame de partículas discreto:
          // NumPartic, NumIterac,. Vmax, wini, wfim, perc_cruz, perc_subs
       Pb // parâmetros base: S={S1, ..., Sn} (Espaço disponível para materialização),
          // Tr (tipo de reparação), nNSO (número de NSOs)
       P=(P1... Pn.); X=(X1... Xn.), E=(E1... En.)  // Capacidade de Processamento
          // e espaço por nó e parâmetros de conexões
Output: M  // conjunto de subcubos a materializar e sua localização

Begin
  1. Inicialização:
    Inicializar parâmetros e serviços do sistema; // carregar parâmetros OEP,
       // perfil de carga, lattice e esquema dimensional
    E←{ }; d←NSCubos(L)*nNSO  // enxame vazio e d=número de dim. do espaço OEP
       // discreto (= número de subcubos possíveis*número de NSOs)
    D←{d1, ..., dd}; // espaço OEP d dimensional
```

Algoritmo 5.5. Algoritmo de Optimização Discreta com Enxames de Partículas Multi-Fase para a Selecção e Alocação de Cubos numa Arquitectura M-OLAP (ODiEP-MF M-OLAP).

Algoritmo 5.5., ODiEP-MF M-OLAP, continuado da página anterior.

```
2. Geração, reparação, avaliação e mostra do estado do enxame inicial:
 Repetir NumPartic Vezes: // vai gerar o enxame com NumPartic partíc. e repará-lo
  p←GeraPartic(Vmax, d) // gera velocidade aleatoriamente (entre –Vmax e Vmax) e
   // aplica regra 4.2 para calcular posição para cada dimensão (igual ao número de
   // subcubos possíveis, dado o mapeamento do problema)
  ParaCada nó n In M-OLAP, Fazer:
   pos_nó=pos(p, n); // pos(p,n) devolve as coordenadas da partícula
    // correspondente às dimensões mapeadas no nó n da arq. M-OLAP
   Se (Tamanho(M (pos_no))>Sn) Então ReparaPartic (pos_no, Tr); // se
    // partícula gerada em posição cuja solução de materialização viola constrang.,
    // vai repará-lo, aplicando Alg. 4.8 ou OEP de Alg. 4.6.
  Próximo n
  E←E∪p;
  Fitness(p)=f (p, I, Ei, Fi, Em, Fm, X); // calcula a aptidão da partícula
   // (f calculada aplicando algoritmos (3.4 ou 3.5) e 3.6)
  pbest(p) ← Fitness(p); // pbest inicial de cada partícula é a aptidão da
   // posição inicial da partícula
  Se Fitness(p)>gbest Então gbest ← Fitness(p); // calcula gbest relativo
   // ao enxame inicial
 FimRepetir
 MostrarEstado (E); // mostra estado instantâneo do enxame e gbest: quantas
  // partículas propõem a materialização de um determinado subcubo em cada nó da arq.
  // M-OLAP e gbest, correspondendo à melhor posição de uma qq. partícula
3. Movimentação do enxame: // gerado, reparado e mostrado o enxame inicial,
 // segue-se a movimentação; o enxame é dividido em dois grupos, que em cada momento
 // estão um na fase 1 e o outro na fase 2, trocando de fase quando nas S recentes
 // iterações não tiver ocorrido uma melhoria da qualidade da solução
 Iter←0; // contador do número de iterações
 Enquanto ( Iter<NumIter) Fazer: //condição de finalização é o número de iterações
  Actualizar w; // faz variar w de wini até wfim, com o número de iterações
  Actualizar sHillClimbing // é seleccionado um número aleatório no intervalo
   // [lowValue, upValue], cujo upValue varia com a iteração corrente
  Se (numIterSMelhoram>rI) Então // se nas rI iterações mais recentes não tiver
   // havido melhoramento da solução, muda de fase
   Se faseCorr=1 Então faseCurr ← 2;
   Senão faseCorr ← 1;
  ParaCada p Em E, Fazer: // vai movimentar cada partícula, aplicando equações
   // e regras da dinâmica das partículas da versão multi-fase
   pTrans ← p; dAnt ← 0; // transfere partícula para partícula transitória que vai
    // ser movimentada e testada (hill climbing); inicializa dAnt
   ParaCada d Em D, Fazer:
    Se num(p)<nPartic/2 Então // partícula pertence ao grupo 1
    // calcular nova velocidade da partícula p para dimensão d
     Se faseCurr=1 Então v_Pd^(iter+1) = c_v1.w.v_Pd^iter + c_2.c_x1.s_Pd^iter + c_2.c_g1.rnd().gbest) ; // aplicar eq. 5.1; s_Pd^iter é a
      // posição da part. p na dim. d na iter. Iter
     Senão v_Pd^(iter+1) = c_v2.w.v_Pd^iter + c_2.c_x2.s_Pd^iter + c_2.c_g2.rnd().gbest) ;
    Senão // partícula pertence ao grupo 2 (em oposição de fase com o grupo 1)
     Se faseCurr=1 Então v_Pd^(iter+1) = c_v2.w.v_Pd^iter + c_2.c_x2.s_Pd^iter + c_2.c_g2.rnd().gbest) ;
     Senão v_Pd^(iter+1) = c_v1.w.v_Pd^iter + c_2.c_x1.s_Pd^iter + c_2.c_g1.rnd().gbest) ;
    Se v_Pd^(iter+1) > vmax Então v_Pd^(iter+1) = vmax SenãoSe v_Pd^(iter+1) < vmax Então v_Pd^(iter+1) = –vmax ; // aplicar regra 4.1
    Se se rnd() < sig(v_Pd^(iter+1)) Então s_Pd^(iter+1) = 1 Senão s_Pd^(iter+1) = 0 ; // calcular localização da partícula:
     // aplicar regra 4.2
```

Algoritmo 5.5., ODiEP-MF M-OLAP, continuado da página anterior.

```
     Se d-dAnt=sHillClimbing OU d=num(D) Então // se já movimentou a partícula
       // de um número sHillClimbing de dimensões ou chegou à
       // última dimensão, vai verificar efeito da movimentação feita
       Fitness(p)=f (pTrans, I, Ei, Fi, Em, Fm, X); // calcula a aptidão de cada
        // partícula gerada
       dAnt←d;
       Se Fitness(pTrans)>Fitness(p) OU (Tamanho(M (pos_n(p, d)))>Sn) Então
        // pos n(xx, y) devolve a posição da partícula xx, nas dimensões mapeadas
        // para um NSO, que contenham y;
        // a movimentação nas sHillClimbing dimensões não foi benéfica ou viola
        // constrang. espacial: repõe pos. vel. anterior da partíc. nessas dimensões
        repõe (p, pTrans, sHillClimbingDim, d, pos, vel)); // transfere, para
         // p, a posição e velocidade de ptrans, correspondente às dimensões
         // d-sHillClimbingDim
     Próximo d
    Próximo p
// vai operar os cruzamentos
    Repetir NumIndiv*Perc_Cruz Vezes:
     nParticPai1 ← rnd (0, NumPartic-1); // selecciona os pais,
     nParticPai2 ← rnd (0, NumPartic-1); // supõe-se que a primeira partícula é 0
     filho1,filho2 ← crossing(E(nParticPai1), E(nParticPai2));
     ParaCada nó n Em M-OLAP, Fazer: // gerados os filhos, vai repará-los
      ParaCada filho Em {filho1, filho2}, Fazer:
       pos_nó=pos(filho, n), Fazer: // pos(p,n) devolve as coordenadas da
        // partícula corr. às dimensões mapeadas no nó n da arq. M-OLAP
       Se (Tamanho(M (pos_no))>Sn) Então ReparaPartic (pos_no, Tr); // repara
        // partícula (identicamente ao feito acima, na fase de inicialização)
      Próximo filho
     Próximo n
     E(nParticPai1) ← filho1; E(nParticPai2) ← filho2; // substituição dos pais
      // pelos filhos gerados
    FimRepetir
    // vai operar a extinção em massa a cada Ie iterações
    Se restoDiv(iter/Ie) = 0 Então // a extinção em massa é executada por
     // reinicialização da velocidade das partículas
     p←GeraParticVel(Vmax, d) // gera velocidade aleatoriamente (entre -Vmax e Vmax)
      // para as d dimensões
    ParaCada p Em E, Fazer: // calcula aptidão de cada partíc. e act. pbest e gbest
     Fitness(p)=f (p, I, Ei, Fi, Em, Fm, X);
     Se Fitness(p)>pbest(p) Então pbest(p) ← Fitness(p);
     Se Fitness(p)>gbest Então gbest ← Fitness(p);
    Próximo p
    Repetir NumIndiv*Perc_Subs Vezes: // vai operar as substituições
     Seleccão e clonagem com aplicação do operador substituição; // aplica
      // selecção e substituição utilizando um de entre vários esquemas
    FimRepetir
    MostrarEstado (En); // mostra estado actual
    iter++; // incrementa número da iteração corrente
   FimEnquanto
  4. Devolver resultado:
   Return M (gbest); // retorna o M correspondente à melhor posição atingida
    // por uma qualquer partícula do enxame
End
```

5.6 Avaliação Experimental

Na continuação da implementação adoptada para a avaliação experimental dos algoritmos descritos no capítulo anterior, também aqui é utilizada uma abordagem orientada a objectos. A implementação dos algoritmos descritos nas secções 5.3 (Greedy M-OLAP), 5.4.2 (AG e Co-Evol-AG M-OLAP) e 5.5.2 (ODiEP M-OLAP, ODiEP-C M-OLAP e ODiEP-MF M-OLAP) foi integrada no conjunto de classes e métodos já implementados, já que utiliza, para função de elegância, os algoritmos descritos e implementados nas secções 3.2.2 e 3.2.3. São também utilizadas todas as outras classes relativas aos parâmetros base, parâmetros específicos de cada um dos algoritmos e classes relativas a serviços de entrada / saída.

Uma vez que se estão a avaliar duas abordagens à optimização da selecção e alocação de cubos para sistemas M-OLAP, vão também dividir-se os testes experimentais em dois grupos. No primeiro, será avaliada a abordagem genética e, no segundo, a abordagem enxame de partículas. Esta separação, além de motivada por razões de sistematização e legibilidade dos resultados experimentais, é também uma consequência da tentativa de acompanhar de perto a descrição reportada em [Loureiro & Belo, 2006f] e [Loureiro & Belo, 2006i].

5.6.1 Implementação

A Figura 5.5, mostra o diagrama de classes simplificado da implementação dos algoritmos genéticos na sua versão padrão e co-evolucionária, algoritmos ODiEP (versão padrão, cooperativa e multi-fase) e uma extensão para suporte do algoritmo *greedy* M-OLAP. Na continuação das implementações anteriores, também os componentes relativos aos vários algoritmos foram desenvolvidos em Java.

Uma breve análise aos algoritmos desenvolvidos permite salientar os principais componentes arquitecturais e classes correspondentes, relacionadas com:

1. o cubo de dados, a informação relativa aos subcubos, hierarquias dimensionais, frequência e extensão de actualizações (classes Cubo, DimParalel, DimHierarq e SubCubo);
2. interrogações, a sua identificação, nó receptor, frequência e extensão (classe InterrNo);
3. arquitectura M-OLAP, como OSNs, espaço de materialização disponível, capacidade de processamento, conexões de comunicação com os seus parâmetros funcionais e alguma informação adicional como hardware instalado, localização, etc. (classes Nos, Conexões e EspaçoNo);

4. conjunto de parâmetros globais e específicos que permitem ao sistema saber como inicializar as estruturas de dados e gerir a execução dos algoritmos (subsistema Parâmetros Base e Específicos relativos aos Algoritmos);
5. a implementação do Algoritmo 3.2, capaz de calcular e disponibilizar as relações de dependência intra-nó (classe SCubosDependencias);
6. os algoritmos genéticos e respectivas estruturas de dados que suportam o seu estado (classes População, Indivíduos e Indivíduos_Coop);
7. os algoritmos ODiEP e respectivas estruturas de dados (classes Enxame, Enxame_Coop e Enxame_MF);
8. o algoritmo Greedy M-OLAP (classe Greedy M_OLAP);
9. o cálculo dos custos de interrogação e manutenção de uma qualquer distribuição de subcubos (classe NoCubo), de acordo com o modelo e algoritmos referidos atrás (no início da secção 5.6);
10. o estado actual e melhor solução conseguida pelos algoritmos, efectuando a sua visualização (classe M_OLAP Estado);
11. a saída de dados, permitindo a persistência de estados internos ou muitas outras informações de caracter estatístico relativas ao desempenho dos algoritmos, importantes para estudo posterior do comportamento dos algoritmos (subsistema Serviços de Saída de Dados).

O diagrama não foi implementado de uma forma "pura". Optou-se pela utilização de *arrays* para armazenar as estruturas de dados de processamento intensivo, pois que o seu desempenho é considerado como o mais rápido, e um tempo de execução curto dos algoritmos é de interesse elevado. Assim, e a título de exemplo, o atributo Genoma_Coev da classe Indivíduos_Coop foi implementado como um *array* tridimensional. A primeira dimensão irá corresponder ao número de espécies, a segunda dimensão corresponderá à população de indivíduos de cada espécie e a terceira dimensão vai corresponder a cada possível bit do genoma de cada indivíduo. Para o caso de termos uma arquitectura M-OLAP com 10 nós, uma população de 100 indivíduos e um cubo com 256 subcubos (cubo B), o *array* será Genoma_Coev[10][100][256]. O tamanho não é excessivo, pelo menos, para esquemas dimensionais não muito complexos, pois que, como o genoma é uma cadeia de bits, cada elemento do *array* pode ser o tipo de dados mais pequeno que suporte o valor 0 e 1 (neste caso, o byte).

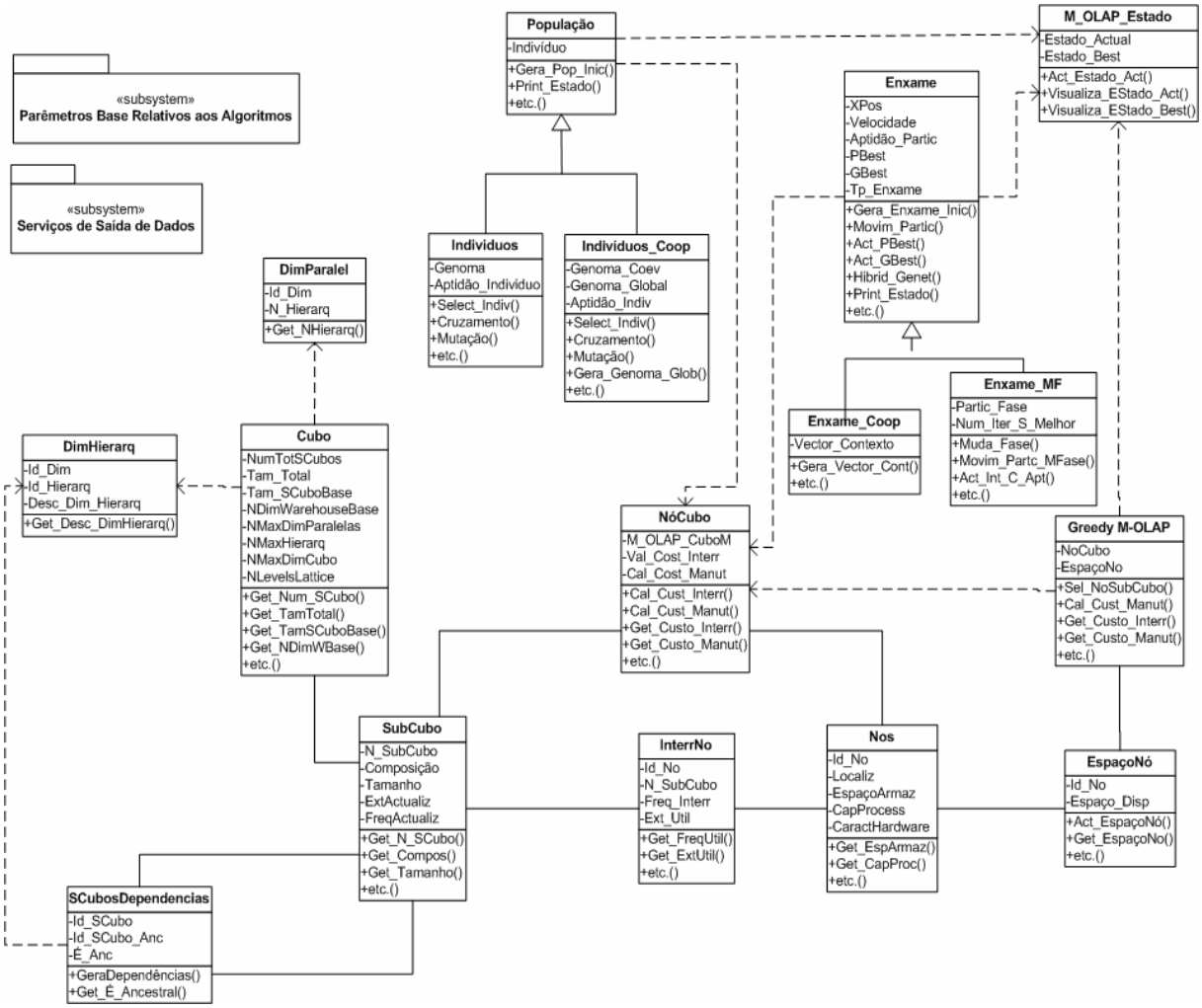


Figura 5.5. Diagrama de classes simplificado do sistema desenvolvido.

5.6.2 Ambiente de Teste

Para teste dos diversos algoritmos de selecção e alocação dos cubos numa arquitectura M-OLAP, utilizou-se, na maioria dos casos, o cubo A (Tabela 2.1), supondo um custo de utilização das relações base de três vezes o tamanho do subcubo de granularidade mais fina (cps); o cubo B (Tabela 2.3) foi também utilizado, quando considerações de escalabilidade estiveram em avaliação.

Para cálculo dos custos de interrogação foi utilizado o algoritmo ACCIEPST (secção 3.2.2), sendo considerado *ei* =1. O algoritmo Greedy2FPR (secção 3.2.3) foi utilizado para cálculo dos custos de manutenção, supondo *em*=0.1 para todos os NSOs.

A Figura 5.6 mostra a arquitectura utilizada na simulação: três NSOs mais a relação base (considerada como nó 0) e uma rede de comunicação. Um *middleware* OLAP distribuído encarrega-se da recepção das interrogações, providenciando a sua alocação ao NSO e seu redireccionamento. É suposto que os serviços de *middleware* residam em cada NSO, numa abordagem *peer-to-peer*, embora um ou mais servidores dedicados possam existir também.

Para simplificar os cálculos, supõe-se que os custos de comunicação entre NSOs e qualquer utilizador são iguais (utilizando-se uma LAN), podendo ser então desprezados. Embora possam ser considerados utilizadores com acesso remoto (através de uma WAN / Internet), caso onde teriam de ser considerados custos adicionais de comunicação (e, possivelmente, OSNs remotos, residentes em clientes [Kalnis et al., 2002a] ou servidores *proxy* [Kalnis & Papadias, 2001]), esta possibilidade não é considerada nos testes ora efectuados. Geraram-se perfis de interrogação aleatórios (normalizados de forma a ter-se um número total de interrogações igual ao número de subcubos do cubo, ou seja, para qualquer distribuição de interrogações $\sum_{i=1}^{n} f_i = n$, onde n é o número de subcubos, e f_i a frequência da interrogação i), perfis estes supostos capazes de induzir a distribuição M (conjunto de subcubos seleccionados e alocados).

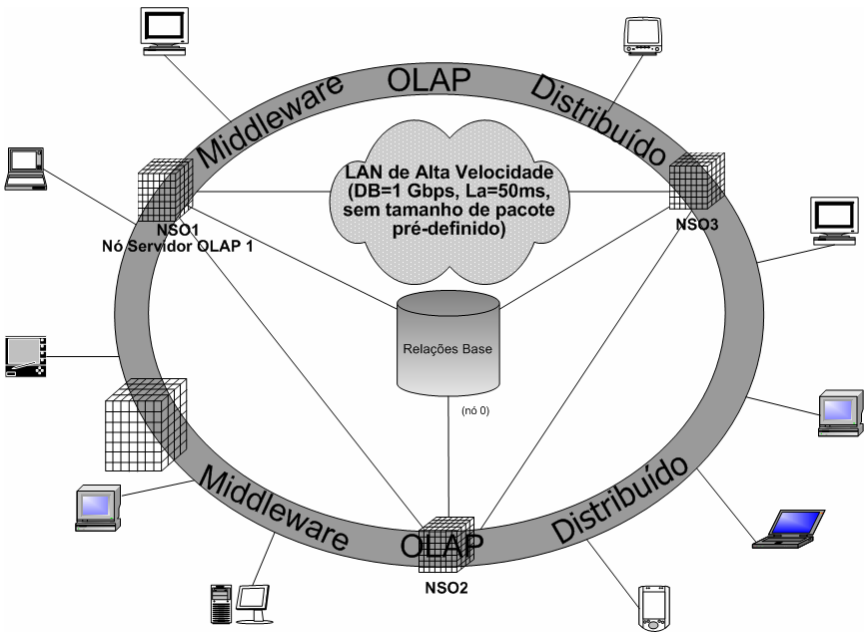


Figura 5.6. Arquitectura M-OLAP com *middleware* OLAP distribuído

5.6.3 Testes Efectuados à Abordagem Genética e Resultados Obtidos

No teste inicial, pretendeu-se obter alguma sensibilidade acerca da afinação de alguns parâmetros dos algoritmos genéticos para depois iniciar o conjunto de testes propriamente dito. Observou-se imediatamente a convergência fácil de ambos os algoritmos, mesmo com pequenas populações (por exemplo 50). Além disso, observou-se também a superioridade da variante co-evolucionária do algoritmo, confirmando a validade da opção tomada. Também se observou que ambos os algoritmos genéticos revelaram um desempenho superior ao algoritmo *greedy*.

No segundo teste avaliou-se comparativamente o desempenho de todos os algoritmos e o impacto do número de indivíduos da população na qualidade das soluções obtidas. Geraram-se três distribuições de interrogações normalizadas diferentes. Como os algoritmos genéticos são inerentemente estocásticos, o resultado de cada teste é a média de várias execuções dos algoritmos (um número entre três e sete). Para o teste inicial, usou-se uma população de 100 indivíduos e um máximo de espaço de materialização por nó (igual para todos) de 10% do espaço necessário à materialização de todos os subcubos. Os parâmetros genéticos utilizados mostram-se na Tabela 5.1.

Tabela 5.1. Parâmetros genéticos utilizados na maioria dos testes efectuados.

Tipo de Evolução	Geracional
Número de Gerações	500
Tipo de Selecção	Competição: buscar aleatoriamente dois indivíduos e com uma probabilidade de 85% seleccionar o indivíduo mais apto.
% Cruzamentos	90%
Crossover	Um ponto
Ponto de Crossover	Gerado aleatoriamente em cada geração
% Mutação	5
Probabilidade de Mutação	1 em cada 8 bits
Forma de lidar com soluções inválidas	Aleatória: comutar bits com valor 1 do genoma seleccionados aleatoriamente (equivalendo à desmaterialização do correspondente subcubo), operação repetida até que o espaço de materialização fique abaixo do espaço disponível no nó.
Geração quase-aleatória do Genoma Global	Com uma probabilidade P= 80%, o melhor indivíduo de cada uma das outras espécies é seleccionado; com uma probabilidade 1-P, um indivíduo aleatório das outras espécies é seleccionado.
Probabilidade de Cruzamentos Inter-Específicos	10%

Os resultados obtidos para a primeira parte deste teste estão apresentados na Figura 5.7. Como se pode ver nessa figura, no gráfico da esquerda, o algoritmo Co-Evol-AG M-OLAP (a versão co-evolucionária) revelou um desempenho superior em todos os três conjuntos de interrogações, com

um desempenho médio superior em 33% relativamente ao algoritmo genético normal e de 44% quando comparado com o algoritmo *greedy*.

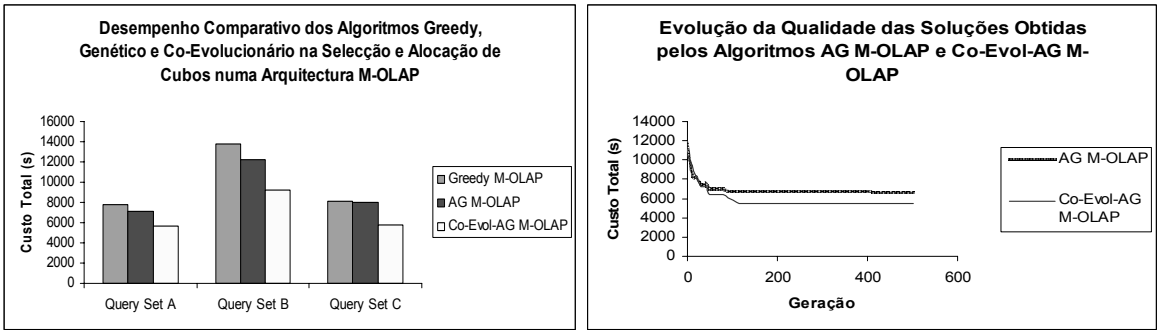


Figura 5.7. Custo e evolução das soluções propostas pelos algoritmos *greedy* e genéticos.

Uma outra característica interessante dos algoritmos que importa avaliar é a rapidez na obtenção da solução. Para o mesmo teste, a Figura 5.7 (gráfico da direita) mostra a evolução da qualidade da solução obtida por ambos os algoritmos genéticos: é clara uma evolução inicial rápida seguida por uma evolução mais lenta. Cerca da geração 100, percebe-se uma variação quase nula, significando que os algoritmos convergiram. Isto é um exemplo, e outras condições de teste poderiam induzir um comportamento diverso, mas estas três fases poderiam ser igualmente identificadas com clareza. Mais uma vez, este gráfico mostra também a superioridade do Co-Evol-AG M-OLAP.

Para avaliar o impacto do número de indivíduos da população no desempenho dos algoritmos genéticos, executou-se este mesmo teste com populações de 20, 50, 100 (a população do teste anterior), 200 e 500 indivíduos. A Figura 5.8 mostra o resultado deste teste. Observou-se que, com uma população de 20 indivíduos, o algoritmo Co-Evol-AG M-OLAP mostrou dificuldade em convergir (depois de 500 gerações, ainda não tinha conseguido a convergência, em várias tentativas, especialmente quando o conjunto de interrogações C era utilizado para indução da solução M, o que explica os maus resultados obtidos para esse caso). Globalmente, observou-se que o número de indivíduos da população genética tem um impacto positivo na qualidade da solução obtida, mas não deveras pronunciado: um aumento de 20 para 500 indivíduos resultou numa melhoria, em média, de 22% na qualidade da solução obtida para o AG M-OLAP e de 30% para o Co-Evol-AG M-OLAP. Este comportamento concorda com a informação de [Lin & Kuo, 2004], embora o AG fosse então aplicado a um sistema OLAP centralizado.

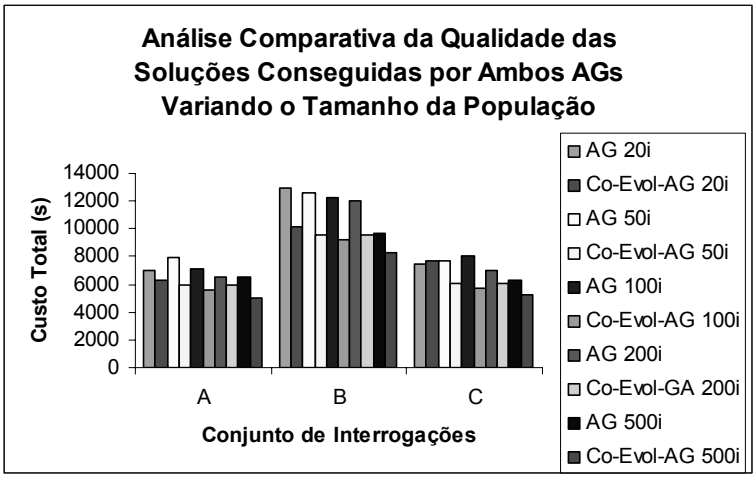


Figura 5.8. Impacto do número de indivíduos da população na qualidade das soluções obtidas pelo algoritmo AG M-OLAP e Co-Evol-AG M-OLAP.

Um terceiro teste tentou avaliar o comportamento dos algoritmos relativamente à complexidade dimensional do esquema OLAP. Utilizaram-se assim os cubos A e B, conforme referido anteriormente. Os resultados com o cubo B confirmaram os obtidos com o cubo A: o Co-Evol-AG M-OLAP mostra o melhor desempenho, conforme pode ser verificado pela inspecção da última coluna da Tabela 5.2, sendo utilizada, neste caso, uma população de 100 indivíduos.

Este mesmo teste permite também verificar outra característica muito importante dos algoritmos: a sua escalabilidade. Para isso, importa analisar o tempo de execução dos algoritmos, e, assim, a Tabela 5.2 mostra também o tempo de execução (em ms) necessário para atingir uma solução de uma determinada qualidade. A análise da tabela mostra que o algoritmo *greedy* é o mais rápido quando o cubo A é utilizado. Mas o mesmo não se verifica quando se usa o cubo B: o tempo de execução aumentou de forma exponencial. Um incremento de quatro vezes no número de subcubos do esquema OLAP implicou um aumento do tempo de execução de (3,103,992 / 72,965 = 42.53). Este comportamento está de acordo com a complexidade do algoritmo *greedy* $O(kn^2)$ referida em [Kalnis et al., 2002b], quadrática em *n*, mas dependente do número de subcubos seleccionados. No algoritmo *greedy* M-OLAP, como há a considerar N nós, a complexidade deverá ser multiplicada por N: ter-se-á então $k.4^2.3 = 48.k$. Neste caso, mesmo sem contar com k, o algoritmo *greedy* M-OLAP apresentou uma complexidade algo menor do que o algoritmo *greedy* original (43 vs. 48.k). Ora supondo um k>1, como será normal, esta diferença será bastante mais marcada.

Tabela 5.2. Custo total, número de gerações e tempo de execução relativos ao algoritmo greedy, AG e Co-Evol-AG M-OLAP.

Arq. M-OLAP com 3 NSOs + Nó base		Cubo A (64 subcubos)	Cubo B (256 subcubos)
Algoritmo Greedy	Qualidade da Solução Final	7,790	801,174
	Tempo de Execução	72,965	3,103,592
AG M-OLAP	Qualidade da Solução Final	7,152	628,134
	Gerações Necessárias	353	299
	Tempo de Execução	237,739	771,319
Co-Evol-AG M-OLAP	Qualidade da Solução Final	5,501	505,285
	Gerações Necessárias	187	295
	Tempo de Execução	323,906	2,802,646

O mesmo tipo de análise pode ser feita para os AGs. De acordo com [Lin & Kuo, 2004], um AG original com reparação *greedy* inversa teria uma complexidade de (grn^2logn), onde *g* é o número de gerações, *r* o tamanho da população e *n* o número de subcubos. Aqui, não foi utilizada a reparação *greedy* inversa, pelo que a complexidade observada foi bastante inferior: apenas linear. O AG M-OLAP mostrou um aumento de tempo de execução (771,319 / 237,739 = 3.2), mesmo inferior ao aumento do número de subcubos; já no que se refere ao Co-Evol-AG M-OLAP, um aumento de (2,802,646 / 323,906 = 8.6) é duplo do incremento do número de subcubos, mas a meio caminho do incremento se a dependência fosse quadrática.

Porém, a última análise é algo falaciosa: compararam-se tempos de execução para atingir soluções finais, não o tempo de execução necessário a atingir uma solução com uma determinada qualidade, neste caso, a do elemento de comparação proporcionado pelo tempo de execução do algoritmo *greedy* e respectiva qualidade da solução. Com esta nova abordagem, produziu-se a Tabela 5.3.

Tabela 5.3. Número de gerações e tempo de execução necessários aos algoritmos AG e Co-Evol-AG M-OLAP para atingir uma solução de qualidade comparável à conseguida pelo algoritmo greedy.

Arq. M-OLAP com 3 NSOs + Nó base		Cubo A (64 subcubos)	Cubo B (256 subcubos)
Algoritmo Greedy	Qualidade da Solução Final	7,790	801,174
	Tempo de Execução	72,965	3,103,592
AG M-OLAP	Qualidade de Solução Comparável	7,622	743,556
	Gerações Necessárias	160	70
	Tempo de Execução	116,077	197,534
Co-Evol-AG M-OLAP	Qualidade de Solução Comparável	6,975	766,542
	Gerações Necessárias	27	60
	Tempo de Execução	55,833	582,969

A superioridade dos AGs é patente; neste caso, o AG padrão parece exibir a melhor relação velocidade X qualidade de solução. A razão pode estar relacionada com o cruzamento inter-específico que, favorecendo a diversidade genética, é lesiva da rapidez de convergência. Além disso, o facto do Co-Evol-AG M-OLAP necessitar de um número de avaliações de aptidão maior (num factor igual ao número de espécies) tem um impacto negativo no tempo de execução, levando a uma evolução mais lenta. Resulta, no entanto, numa maior eficácia global. É o preço a pagar pela superioridade observada relativamente à qualidade da solução final.

Uma outra medida importante quanto à escalabilidade dos algoritmos está relacionada com o comportamento do tempo de execução quando o número de NSOs da arquitectura M-OLAP é alterado. Assim, executou-se outro teste, utilizando uma arquitectura com 10 NSOs + nó base, também para um espaço de materialização de 10% e utilizando o cubo A.

Desta vez, mantendo os mesmos parâmetros genéticos usados nos testes antecedentes, o AG M-OLAP revelou o melhor desempenho. Observou-se também que o Co-Evol-AG M-OLAP mostrava alguma dificuldade em convergir. Procurando causas possíveis, pensou-se imediatamente nos cruzamentos inter-específicos como sendo a causa do distúrbio. Como o número de nós é elevado, a probabilidade de 10% para o cruzamento inter-específico, funcionando como um operador mutação em larga escala, perturbava o processo de evolução. Dessa forma, alterou-se para 2% e mudou-se também a taxa de mutação para 2%, tentando limitar os factores de perturbação. Os resultados obtidos no teste, com estes novos parâmetros, mostram-se na Tabela 5.4. A melhor escalabilidade do AG padrão é clara (73 → 517 para o *greedy* vs. 237 → 377 para o AG), o mesmo não é verdade para a versão co-evolucionária (73 → 517 para o *greedy* vs. 323 → 8,036 para o Co-Evol-AG). Mas esta conclusão é algo apressada: se forem comparados os tempos de execução necessários para atingir uma solução comparável para os AGs versão padrão e co-evolucionária quando aplicados à arquitectura 10NSOs+1, observou-se que, após 170 gerações, o Co-Evol-AG M-OLAP conseguiu uma solução de custo 8,241 (algo melhor do que a solução final do AG M-OLAP), em 1,952 seg., cerca de 4 vezes menor do que o tempo de execução para atingir a solução final, mas, ainda assim, bastante superior aos 377 seg. necessários ao AG M-OLAP, mas já não tão favoráveis para este último. Também extrapolando esta análise, se pode concluir que o Co-Evol-AG mostra ainda uma melhor escalabilidade do que o *greedy*.

Tabela 5.4. Custo total, gerações e tempo de execução relativo aos algoritmos greedy, AG e Co-Evol-AG quando aplicados a arquitecturas M-OLAP com 3 e 10 NSOs + nó base.

64 subcubos		3 NSOs + Nó base	10 NSOs + Nó base
Algoritmo Greedy	Qualidade da Solução Final	7,790	12,461
	Tempo de Execução	72,965	517,243
AG M-OLAP	Qualidade da Solução Final	7,152	8,430
	Gerações Necessárias	353	270
	Tempo de Execução	237,739	377,853
Co-Evol-AG M-OLAP	Qualidade da Solução Final	5,501	6,811
	Gerações Necessárias	187	670
	Tempo de Execução	323,906	8,036,837

Repetiu-se o último teste usando uma população de 200 indivíduos: os resultados confirmaram as conclusões anteriores, mas agora a qualidade das soluções obtidas pelos AGs foi ainda melhor: um custo de 7,124 s para o AG M-OLAP, a que correspondeu um tempo de execução de 518,616 ms e um custo de 4,234 s, para um tempo de execução de 16,409,446 ms para o Co-Evol-AG M-OLAP. E a qualidade ainda melhorou quando mais gerações (e tempo de execução) foram permitidas: 6,524/2,240 e 3,971/20,294 (custo/tempo de execução e para os mesmos algoritmos). Ainda que tempos de execução da ordem de 20,000 segundos sejam claramente excessivos, recorde-se que foi usado um *laptop* com um processador AMD Athlon XP-M 2500 para os obter. Se um computador mais poderoso fosse utilizado, os tempos de execução seriam substancialmente menores e dentro dos limites do utilizável. Por outro lado, uma versão paralela *peer-to-peer* poderia ser utilizada, já que de fácil concepção e implementação, considerando as características paralelas do AG co-evolucionário. Ambas as soluções evitarão os muito longos tempos de execução referidos atrás (e mais, se utilizadas em conjunto), justificando plenamente o consumo extra de capacidade de processamento, considerando os ganhos de desempenho obtidos.

Este mesmo teste pode também ser utilizado para obter algumas indicações acerca do *scale-up* da arquitectura M-OLAP. O custo total estimado foi quase o mesmo quando foi utilizada a arquitectura M-OLAP de 3 NSOs+nó base ou 10 NSOs+nó base. Isto significa que a arquitectura M-OLAP revela um fácil *scale-up*: um incremento na carga de interrogações foi acompanhado por um incremento proporcional de capacidade (processamento e armazenamento) que mantém quase o mesmo custo total (o mesmo tempo de resposta a interrogações e tamanho de janela de manutenção).

Ainda relativamente aos testes do tempo de execução, é importante também estender a análise preliminar de complexidade efectuada quando se discutiram os resultados da Tabela 5.2. Avalia-se agora o impacto do número de indivíduos da população no tempo de execução dos algoritmos

genéticos. Foram utilizados todos os parâmetros e condições de teste utilizados para gerar os valores mostrados na Figura 5.8. Assim, executou-se uma sequência de testes com populações de 20, 50, 100, 200 e 500 indivíduos. Os resultados obtidos apresentam-se no gráfico da Figura 5.9.

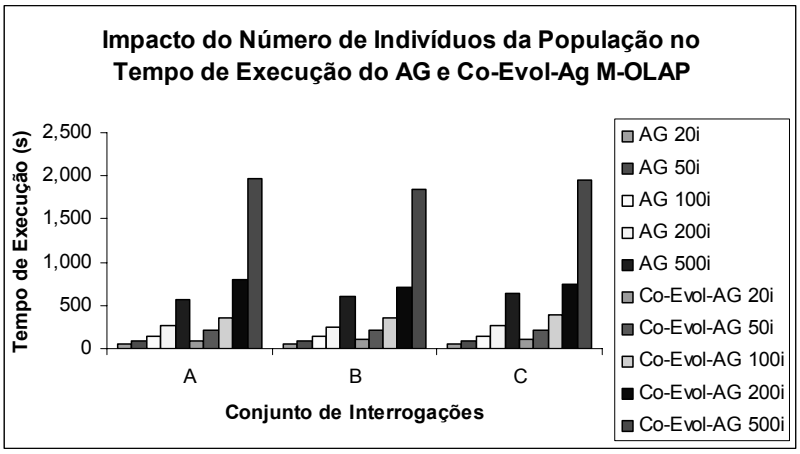


Figura 5.9. Impacto do número de indivíduos da população na qualidade das soluções conseguidas pelos AG M-OLAP e Co-Evol-AG M-OLAP.

É obvia a independência do tempo de execução relativamente ao conjunto de interrogações utilizado e é também clara a relação directa com o número de indivíduos da população, confirmando a análise de [Lin & Kuo, 2004]. Além disso, é claro que a versão co-evolucionária é substancialmente mais lenta do que a versão normal. A razão para este comportamento é simples. Todas as operações genéticas e avaliação da aptidão são directamente proporcionais ao número de indivíduos da população. A versão co-evolucionária, por seu lado, implica um número maior de avaliações, proporcional ao número de nós (igual ao número de espécies).

O próximo teste vai tentar avaliar o impacto do limite do espaço de materialização por nó na qualidade das soluções propostas pelos três algoritmos. Este teste pode ser muito valioso, já que proporcionará algumas indicações acerca do espaço de materialização "mais lucrativo", aquele que confira a melhor relação espaço X custos. Para isso, os algoritmos foram executados para espaços de materialização de 1, 2, 3, 5, 10, 20 e 30% do espaço de materialização total do cubo, usando uma arquitectura M-OLAP 3+1, o cubo A (64 subcubos) e os parâmetros genéticos do último teste. Os resultados obtidos mostram-se na Figura 5.10.

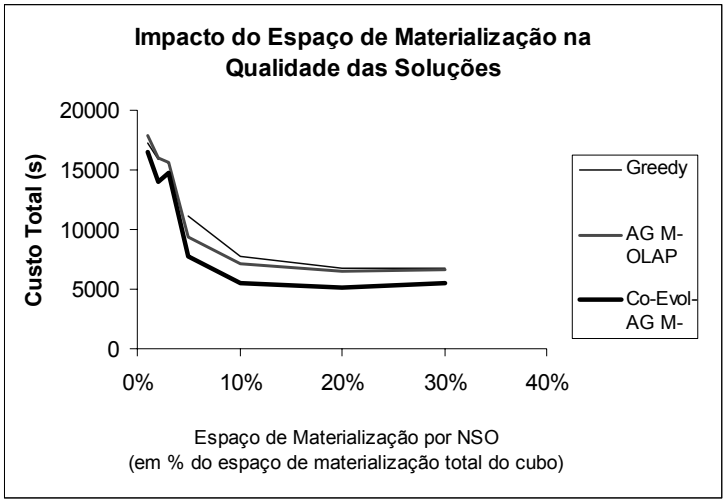


Figura 5.10. Impacto do limite de espaço para materialização por NSO na qualidade das soluções conseguidas pelos algoritmos Greedy, AG e Co-Evol-AG M-OLAP.

Confirma-se a melhor qualidade das soluções propostas pelo algoritmo Co-Evol-AG M-OLAP, sendo consistentemente superior aos outros. Também é patente que o espaço de materialização é lucrativo, com um retorno inicial elevado que, progressivamente, decresce. Se, inicialmente, a relação diminuição de custos de interrogação X custos de manutenção pende decididamente para o primeiro prato da balança (já que qualquer subcubo materializado pode ser muito útil para responder a interrogações), a balança vai, progressivamente, tendendo para o equilíbrio (com a progressiva degradação da utilidade de cada novo subcubo materializado) e algures entre 20% e 30% atinge-se o ponto de equilíbrio. Para lá deste valor, qualquer novo subcubo a materializar importa num custo de manutenção que não é justificado pelos benefícios havidos em termos de custo de interrogação, provavelmente por serem subcubos redundantes ou quase-redundantes. Este comportamento vem justificar os 10% de espaço de materialização que foi utilizado na maioria dos testes anteriores.

Este conjunto de testes experimentais não estariam finalizados sem que fosse avaliado o impacto de variações em alguns dos parâmetros genéticos no desempenho global dos algoritmos. De entre os parâmetros e combinações possíveis, seleccionaram-se para investigação: 1) a forma de lidar com os genomas inválidos (quando a solução produzida viola o constrangimento especial); 2) o tipo de selecção; e 3) para a selecção tipo competição (a utilizada até aqui), analisar o impacto da probabilidade de selecção do indivíduo mais apto (*Ps*), de entre os dois aleatoriamente buscados à população. Para estes testes, usou-se uma arquitectura M-OLAP 3+1, uma população de 100

indivíduos, uma probabilidade de mutação de 2% e uma probabilidade de cruzamento inter-específico de 2%.

Para o primeiro teste deste conjunto, implementou-se um outro método de lidar com genomas inválidos, simplesmente actuando no processo de evolução de forma a invalidar os genomas não conformes, permitindo somente a geração de genomas cuja solução de materialização esteja dentro do constrangimento especificado. Este tipo de actuação (evitou-se a nomeação reparação, já que esta não existe realmente) vai adiante ser nomeado como do tipo V (só permitidos genomas válidos). Este novo tipo adiciona-se aos dois já implementados: tipo A (reparação aleatória, utilizada até aqui, mostrada em Algoritmo 4.5) e tipo D (reparação por perda mínima de densidade de aptidão – relação (incremento de custo pela remoção do subcubo / tamanho do subcubo removido) – Algoritmo 4.6).

Num algoritmo genético, um genoma é gerado na fase inicial (ao ser criada a população) e sofre a acção dos operadores genéticos (*crossover* e mutação, principalmente). Em termos temporais, ter-se-á: a fase de geração, *crossover* e mutação. Em qualquer destas fases, torna-se necessário lidar com os genomas inválidos. Para cada uma dessas reparações, um dos métodos atrás referidos pode ser utilizado e, então, a reparação do genoma pode ser denotada com três letras, correspondendo à forma de lidar com o genoma inválido em cada uma das três fases. Por exemplo, AAA significará que será utilizada a reparação aleatória em cada uma das três fases: geração, cruzamentos e mutação. Como o número de combinações possíveis dos três tipos de reparação é muito elevado, optou-se por testar apenas três delas: AAA, DAA e AAV. A Tabela 5.5 mostra o resultado da execução dos algoritmos, utilizando o conjunto de interrogações A e cada um dos três métodos (combinações dos tipos indicados) de actuação. Os valores são a média de quatro a seis execuções de cada algoritmo.

Uma inspecção da tabela permite concluir, de imediato, que nenhum dos métodos utilizados constitui uma solução notável, já que nenhum se salienta decididamente na obtenção de soluções de qualidade destacada. Uma análise detalhada mostra que o método de reparação *greedy* inversa (usado na fase de geração, fazendo parte da combinação DAA) não é lucrativo (relativamente ao AAA), já que simplesmente não melhora a qualidade das soluções (ou fá-lo apenas marginalmente), implicando um aumento elevado do tempo de execução. Ao contrário, o tipo de reparação V aplicado à mutação é interessante, especialmente para a versão co-evolucionária. Este resultado salienta o interesse de uma investigação futura acerca de outros esquemas de reparação deste tipo que possam ser incluídos no operador cruzamento.

Tabela 5.5**.** Qualidade das soluções produzidas e tempo de execução dos algoritmos AG e Co-Evol-AG M-OLAP.

	AG M-OLAP		Co-Evol-AG M-OLAP	
	Custo Total	Tempo Exec.	Custo Total	Tempo Exec.
Lidar com M Inválido: AAA	7,182	155,036	6,037	406,037
Lidar com M Inválido: DAA	7,439	1,236,431	5,967	1,400,108
Lidar com M Inválido: AAV	6,967	173,969	5,569	287,854

Para o segundo teste, implementou-se um outro mecanismo de selecção: a selecção proporcional, já que é aquele que foi originalmente proposto por Holland [Holland, 1992], de entre outros esquemas típicos [Blickle & Thiele, 1996], também descritos em 4.4.4. Aqui, ele foi implementado na sua forma "roleta", um esquema descrito atrás em 4.4.4. Resumidamente, poder-se-á dizer que cada indivíduo da população ocupará uma "*slot*" de tamanho proporcional à sua aptidão. Quando a bola roda na roleta, a probabilidade de um indivíduo mais apto ser seleccionado é maior, pois que a sua *slot* é correspondentemente mais extensa.

Nas condições indicadas e nas 300 gerações permitidas, a versão normal do AG, por vezes, não convergiu, e a versão co-evolucionária, nunca. O problema pode ser explicado pela baixa pressão de selecção, devido à propriedade indesejável que resulta do facto da selecção proporcional não ser "translação invariante" (ver por exemplo [de la Maza & Tidor, 1993]). O facto pode ser explicado da seguinte forma: suponha-se uma população de 10 indivíduos onde o mais apto tem um custo de 1, o segundo melhor 2, e assim sucessivamente até ao menos apto com um custo de 10. A probabilidade de selecção será de 18.1% para o mais apto e de 1.8% para o menos apto, claramente diversas. Mas adicione-se 100 aos custos: ter-se-á, neste caso, uma probabilidade de 10.4 para o mais apto e de 9.5 para o menos apto, quase iguais. O problema, aqui tratado na sua versão multi-nó, cai nesta situação: se para OLAP centralizado, como se verá na secção 5.7, o método parece ser aplicável (ainda que não o melhor), aqui simplesmente não é utilizável, havendo que recorrer a outros métodos que não enfermem do mesmo problema, ou adoptar esquemas de correcção (por exemplo escala logarítmica [Grefenstette & Baker, 1989] ou sobresselecção [Koza, 1992]). Os autores em [Blickle & Thiele, 1996] concluem que a selecção proporcional é um esquema desaconselhado. Estes resultados confirmaram o uso da selecção competição nos testes ora reportados, até porque a existência do parâmetro probabilidade de selecção (cujo impacto se irá avaliar no último teste deste grupo) neste esquema, permite seleccionar a pressão selectiva de forma simples, o que não existe na selecção proporcional.

Todavia, a grande diversidade de métodos de selecção (e as muitas variantes), propostos na literatura e descritos em 4.4.4, levaram a pensar no interesse de uma investigação mais extensiva do impacto do esquema de selecção no desempenho dos algoritmos genéticos, quando aplicados ao problema da selecção de cubos. Métodos como selecção *rank* e selecção por aptidão uniforme parecem merecer maior investigação. Assim, empreendeu-se esta tarefa, cujos resultados serão apresentados e discutidos na secção 5.7 deste capítulo.

Finalmente, para o terceiro teste deste conjunto, usaram-se os mesmos parâmetros genéticos do primeiro, com reparação de genoma inválido tipo AAA. Interessa avaliar o impacto da alteração da pressão de selecção, variando *Ps* - probabilidade de selecção do melhor de entre dois, desde 75 a 100%, em incrementos de 5%, utilizando o conjunto de interrogações A. O resultado do teste mostra-se na Figura 5.11.

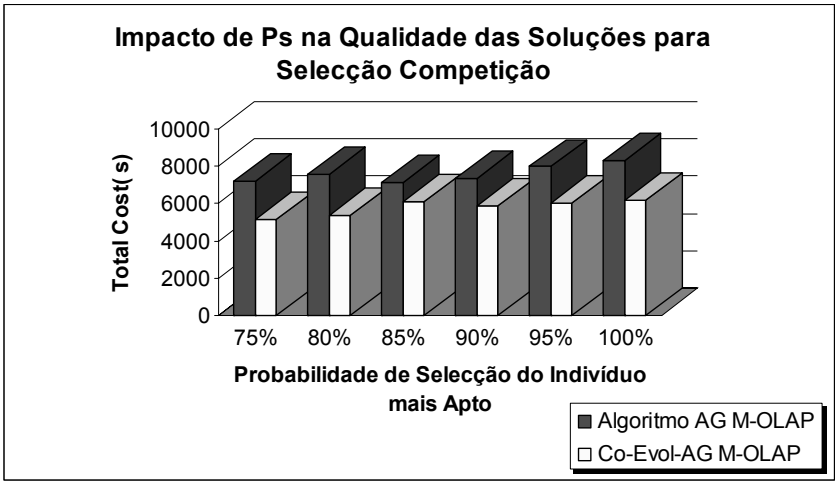


Figura 5.11. Impacto da probabilidade de selecção do indivíduo mais apto, quando utilizado o método de selecção competição nos algoritmos AG e Co-Evol-AG M-OLAP.

Uma análise breve ao gráfico permite concluir que o valor desta probabilidade não tem um impacto marcado na qualidade das soluções obtidas, mas, ainda assim, é mais visível para o algoritmo AG M-OLAP. Valores de 85-90% serão uma boa escolha, já que não favorecem especialmente qualquer dos dois algoritmos, como seria o caso de valores 75-80%, que seriam ideais para o Co-Evol-AG M-OLAP, mas lesivos do AG-M-OLAP. Observou-se também que, para valores <= 70%, os algoritmos apresentavam dificuldades de convergência (sintoma de baixa pressão selectiva). Estas observações justificaram a escolha dos valores de 85% e 90% utilizados na maioria dos testes realizados.

5.6.4 Teste da Abordagem Optimização Discreta por Enxame de Partículas

O ambiente de teste é o mesmo utilizado em 5.6.2, pelo que não serão aqui repetidos os pormenores. Os três algoritmos distintos vão estar em análise, pelo que se optou pela sua avaliação comparativa, evitando a profusão de séries nos gráficos. Além disso, a comparação do desempenho relativamente aos esquemas de optimização analisados na secção anterior é suficiente para avaliação externa.

Tal como na avaliação experimental descrita na secção anterior, foram executados alguns testes preliminares, para permitir afinar alguns parâmetros dos algoritmos ODiEP, tais como Vmax, w, c1 e c2. Usaram-se, como ponto de partida, os valores referência obtidos pela análise preliminar realizada na secção 4.10.2 e informação mencionada em outros trabalhos. Depois de alguns testes, encontraram-se, como mais adequados, os seguintes valores: Vmax=10, w=[0.99,1.00] (variado linearmente com o número de iterações) e c1=c2=1.74. Gerou-se o vector velocidade de modo aleatório e a posição inicial com a regra 4.2. Sempre que a posição de uma qualquer partícula é inválida, ela é reposicionada utilizando um esquema aleatório de comutação de vector, que desloca a partícula (dimensão a dimensão) de posições 1 para posições 0, até que entre numa zona do espaço permitida. Dada a natureza estocástica dos algoritmos, todos os valores apresentados constituem a média de 10 execuções. É importante referir que, em todos os testes, a convergência dos algoritmos foi verificada, mostrando a sua capacidade de encontrar soluções para o problema da selecção e alocação de cubos.

O primeiro teste destinou-se a avaliar o impacto do número de partículas na qualidade da solução, usando-se, neste caso, o algoritmo ODiEP M-OLAP. Os resultados mostram-se na Figura 5.12 (gráfico da esquerda).

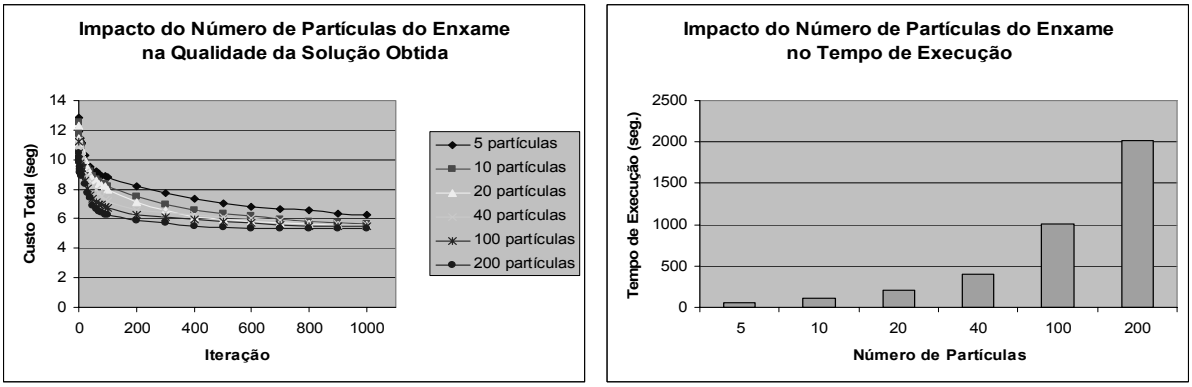


Figura 5.12. Impacto do número de partículas do enxame na qualidade das soluções obtidas e no tempo de execução do algoritmo ODiEP M-OLAP.

Como se pode ver, um enxame com um número elevado de partículas consegue boas soluções após um número reduzido de iterações; mas, se um número de iterações razoável for permitido, a diferença de qualidade nas soluções obtidas por um enxame pequeno ou grande reduz-se para valores pequenos.

Na Figura 5.12 (gráfico da direita) mostra-se o tempo de execução do ODiEP M-OLAP para 1000 iterações, variando o número de partículas. Como se pode observar, o tempo de execução cresce linearmente com o número de partículas, com um valor de dez segundos.partícula^{-1}. Daqui se deduz o interesse de utilizar enxames pequenos, já que o gráfico da esquerda mostra que enxames pequenos conseguem soluções de qualidade quase idêntica à obtida por enxames grandes, permitindo manter o tempo de execução dentro de intervalos de valores admissíveis.

Uma análise mais aprofundada da questão aventada na discussão dos resultados da Figura 5.12 materializa-se na análise qualidade vs. tempo de execução mostrada na Tabela 5.6. Observando a tabela, pode concluir-se que, por exemplo, um enxame de 200 partículas conseguiu uma solução de 5,914 após 200 iterações, gastando 402 segundos. Um enxame de 10 partículas conseguiu uma solução idêntica, após 700 iterações, em apenas 71 segundos e um enxame de 20 partículas, após 500 iterações, em 101 segundos. Para enxames de 40 e 100 partículas, os resultados foram piores: são necessárias 600 e 450 iterações gastando 237 e 443 segundos, respectivamente. Esta análise levou à selecção de enxames de 20 partículas para todos os testes subsequentes, um bom valor, pesando a relação qualidade da solução final e tempo de execução.

Tabela 5.6. Qualidade das soluções e tempo necessário à sua obtenção para o algoritmo ODiEP M-OLAP.

	às 200 iterações		às 400 iterações		às 500 iterações		às 600 iterações		às 700 iterações		às 800 iterações		às 1000 iterações	
Partículas	Custo	Tempo	Custo	Tempo	Custo	Tempo	Custo	Tempo	Custo	Tempo	Custo	Tempo	Custo	Tempo
10	7475	22	6558	41	6312	50.5	6152	60.5	5924	71	5800	82.5	5655	103.5
20	7080	44	6137	80.5	5922	101	5834	126.5	5743	136	5659	158	5573	200.5
40	6555	84.5	6263	167.5	6118	199.5	6025	237.5	5840	277.5	5707	252.5	5537	393.5
100	6264	196	5958	394.5	5810	500	5698	595	5566	721.5	5506	803	5469	1008.5
200	5914	402.5	5467	812	5406	1008.5	5365	1110.5	5349	1422	5311	1621.5	5304	2022

O próximo teste avalia comparativamente os três algoritmos de enxame desenvolvidos, em termos da qualidade das soluções e rapidez de obtenção. Como se pode verificar no gráfico da esquerda da Figura 5.13, as versões cooperativa e multi-fase chegam a boas soluções num número menor de iterações: como o intervalo entre avaliações de aptidão é mais pequeno, qualquer melhoria faz efeito imediato (é publicada para poder ser utilizada por outros). Contudo, se for permitido um número de

iterações elevado, as versões normal e cooperativa chegam a soluções de qualidade quase idêntica, comportando-se a versão multi-fase algo pior. Uma análise qualidade vs. tempo de execução mostra que, mesmo para um número baixo de iterações (por exemplo 100), os algoritmos ODiEP-C M-OLAP e ODiEP-MF M-OLAP mostram um melhor desempenho: pode observar-se que ODiEP-C M-OLAP necessita de 89 segundos para chegar a uma solução de 6507, enquanto, para uma solução similar, ODiEP M-OLAP usa 63 segundos para uma solução de 6479. Para o caso ODiEP-MF M-OLAP x ODiEP M-OLAP, o prato pende mais ainda para o segundo: 8,875 em 175 seg. vs. 6,893 em 43 seg.

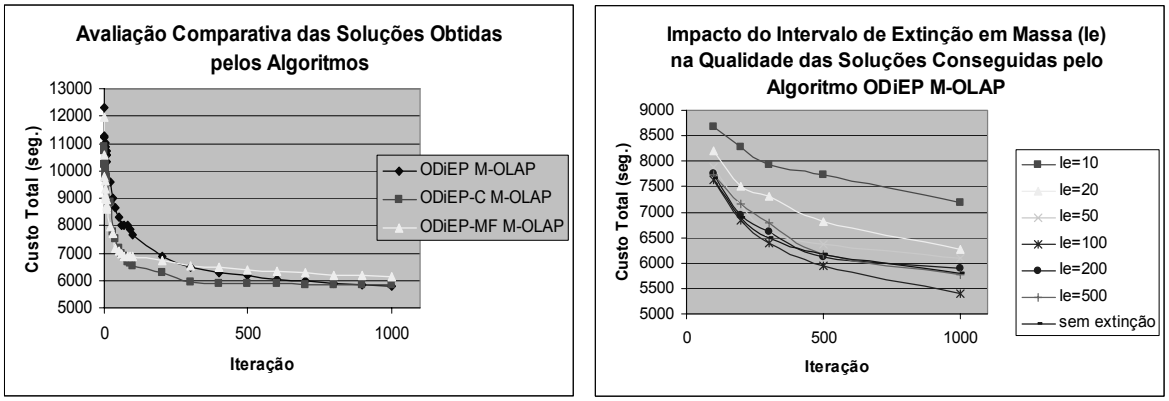


Figura 5.13. Qualidade obtida pelos algoritmos ODiEP, ODiEP-C e ODiEP-MF M-OLAP e impacto da extinção em massa na qualidade das soluções obtidas pelo algoritmo ODiEP M-OLAP.

O próximo teste procura avaliar se a extinção em massa será também benéfica quando a ODiEP é aplicada a este tipo de problema. Assim, executou-se o algoritmo ODiEP M-OLAP, usando para Ie os valores {10, 20, 50, 100, 500}. O gráfico da direita da Figura 5.13 mostra os resultados obtidos. Como pode verificar-se, apenas para Ie=100 o esquema parece ser benéfico.

Para completar este conjunto de testes, relativo ao desempenho dos algoritmos, estudou-se o comportamento das versões híbridas genéticas dos algoritmos ODiEP e ODiEP-C M-OLAP. Na generalidade, observou-se que os operadores genéticos têm um impacto reduzido no desempenho destes algoritmos. A análise específica ao impacto do operador cruzamento mostra-se na Figura 5.14. No gráfico da esquerda, ainda que o número de séries seja grande, pode observar-se que, só para o ODiEP M-OLAP e para 20% de cruzamentos, o operador parece ser limitadamente benéfico e só para valores muito elevados de iterações. Todos os outros valores utilizados prejudicam a qualidade. O detalhe mostrado na mesma figura, à direita, confirma a observação. Como pode ver-se, os cruzamentos de 20% são marginalmente benéficos para a versão normal, mas não para a

cooperativa. Neste último algoritmo, cada enxame corresponde a um nó, pelo que os cruzamentos só trocam informação relativa a cada um dos nós, parecendo que nenhum ganho é obtido dessa forma. Algo diverso sucede para a versão normal, onde o cruzamento de informação relativa aos diversos nós parece ser benéfica, mas só até um determinado nível, pois que valores muito elevados decerto perturbam o enxame, prejudicando a qualidade das soluções obtidas.

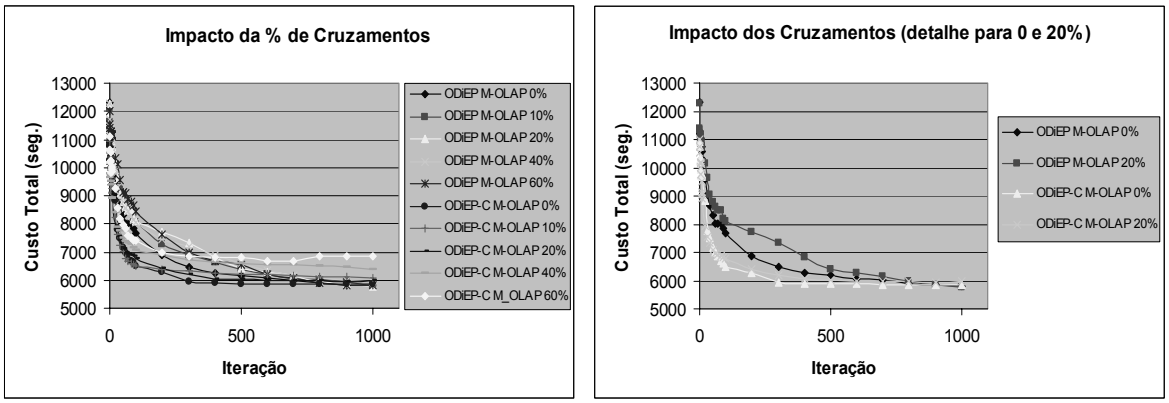


Figura 5.14. Impacto do cruzamento genético nos algoritmos ODiEP e ODiEP-C M-OLAP na qualidade das soluções obtidas.

O último teste avalia a escalabilidade, no que toca ao número de nós da arquitectura M-OLAP. Foi utilizado o algoritmo ODiEP M-OLAP, já que os outros dois, semelhantes em termos de complexidade, terão um comportamento semelhante (mas devendo exibir valores absolutos bastante mais elevados, uma vez que utilizam um maior número de avaliações de aptidão). Foram utilizadas duas arquitecturas: a até agora utilizada, com 3+1 nós e outra, com 10+1 nós. Gerou-se outro perfil aleatório para os 10 nós, também devidamente normalizado. O gráfico da Figura 5.15 mostra o resultado do teste.

Como pode ver-se, um aumento de 10/3 no número de nós implica um factor no aumento de tempo de execução de 3.8, revelando uma quase linearidade, provando um suporte fácil dos algoritmos para arquitecturas M-OLAP com muitos nós. Trata-se de uma característica interessante, já que a arquitectura M-OLAP evidenciou um bom *Scale-up* nos testes reportados em [Loureiro & Belo, 2006g] e reflectidos neste trabalho na secção 3.2.4.

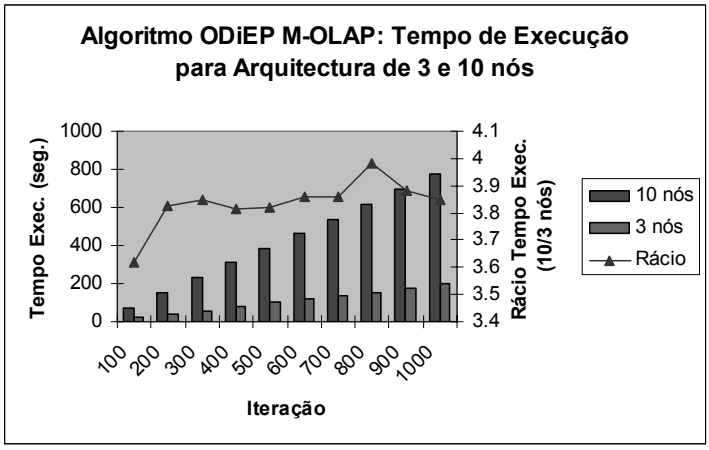


Figura 5.15. Tempo de execução do algoritmo ODiEP M-OLAP numa arquitectura M-OLAP 3+base e 10+base e rácio correspondente.

5.7 Estudo Comparativo dos Esquemas de Selecção no Desempenho dos Algoritmos Genéticos Quando Aplicados ao Problema da Selecção de Cubos

Para este estudo, utilizou-se a arquitectura OLAP centralizada e a distribuída, ficando assim os testes divididos em duas fases, sendo a última utilizada para verificação dos melhores esquemas encontrados na primeira fase. Na primeira, utilizou-se o cubo A e as condições típicas utilizadas nos testes ao Algoritmo 4.7 (SCO-AGP) apresentado na secção 4.8 e utilizado nos testes descritos na secção 4.10.3. Na segunda fase, utilizou-se também o cubo A e as condições típicas utilizadas nos testes descritos na secção 5.6.3, onde foi aplicado o algoritmo genético padrão (AG M-OLAP). Foram avaliados experimentalmente os seguintes esquemas de selecção: proporcional, truncamento, *ranking* (linear e exponencial), torneio, competição e aptidão uniforme, sendo utilizada selecção geracional.

Gerou-se uma distribuição aleatória de interrogações devidamente normalizada. Os testes foram executados para diversos espaços de materialização: 1, 2, 3, 4, 5, 6, 8, 10, 15 e 20% do espaço de materialização total do cubo. Devido à natureza estocástica dos AGs, cada teste foi executado entre 6 e 10 vezes para cada constrangimento espacial, perfazendo, assim, na maioria das vezes, um total de 60-100 execuções, sendo os valores exibidos nos gráficos e tabela, a média dos valores obtidos em cada execução dos algoritmos.

5.7.1 Fase 1 – Utilização de Arquitectura OLAP Centralizada

Alguns esquemas de selecção têm um parâmetro que permite controlar a pressão de selecção: 1) na selecção torneio, *t*, o tamanho do torneio; 2) para a selecção competição, a probabilidade P de selecção do melhor indivíduo (de entre o par seleccionado aleatoriamente); 3) para o tipo de selecção truncamento, o número de melhores indivíduos de onde é seleccionado um de forma uniforme; e finalmente, para 4) a selecção *ranking* exponencial, o valor de *r*, a base do expoente. Desta forma, foram realizados alguns testes preliminares para providenciar ao seu ajuste, utilizando um espaço de materialização de 5% (o valor onde ocorre o maior decréscimo de custos) e permitindo apenas 100 gerações. Para a selecção torneio, os resultados mostraram que o tamanho do subconjunto selecção tem um impacto limitado na qualidade da solução conseguida, mas observou-se que, quanto mais elevado, mais rápida era a convergência, o que prejudicava a qualidade das soluções. Assim, escolheu-se o valor 4 (duplo do torneio binário). Já relativamente à selecção competição, executou-se um teste, onde P foi variado desde 50 até 99%. Para valores de P abaixo de 60%, o algoritmo não convergia. Para valores 60%<P<75%, a convergência era fraca e, acima de 75%, o algoritmo convergia decididamente, só havendo lugar a fenómenos de convergência prematura para valores acima de 98%. Desta forma, seleccionou-se P=85%, um valor intermédio que parece ser uma boa escolha entre a pressão de selecção e necessidade de diversidade, aliás já profusamente utilizado nos testes efectuados. Para a selecção truncamento, variou-se *T* desde 80 a 99. Os resultados mostram-se na Figura 5.16, no gráfico da esquerda. Como se pode verificar, valores no intervalo [90, 95] parecem constituir uma boa escolha, mas, para T=95, as 100 gerações parecem ser curtas para a convergência: seleccionou-se o valor 94.

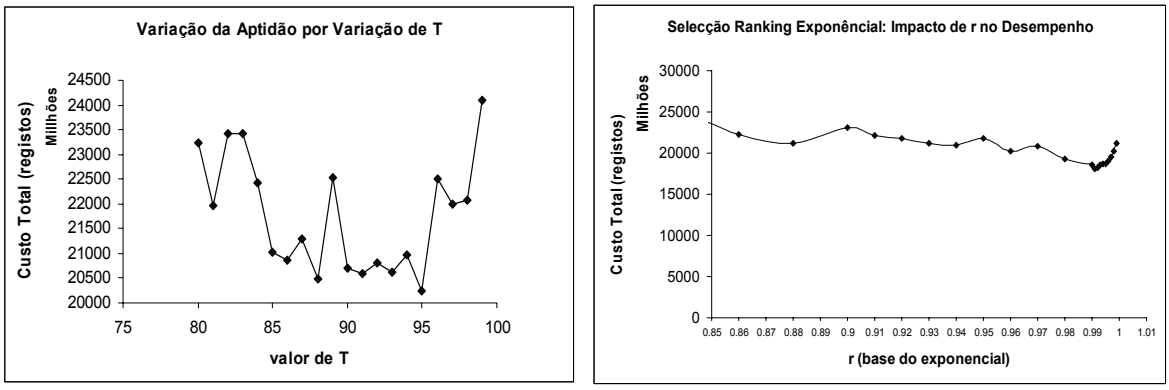


Figura 5.16. Impacto de *T* e *r* no desempenho do AG.

Para o *ranking* exponencial, o desempenho máximo é atingido para valores cerca de 0.99. O valor r=0.991 (obtido da análise dos dados) foi o utilizado (Figura 5.16, gráfico da direita).

O primeiro teste efectua uma análise comparativa entre os diversos tipos de esquemas de selecção, sendo utilizado o cubo B para estes testes. Os resultados apresentam-se na Figura 5.17. Como se pode observar no gráfico da esquerda, é patente uma diferença algo relevante (cerca de 9.5 % entre o melhor e o pior valor). O *ranking* exponencial é o esquema de selecção que confere ao AG as melhores soluções; a selecção por aptidão uniforme, roleta e truncamento também são boas candidatas. O desempenho da selecção por *ranking* exponencial era expectável, já que o processo de afinação do parâmetro *r* é particularmente simples e possivelmente o mais correcto, dada a clareza e consistência do seu comportamento. Também a selecção por aptidão uniforme parece ser uma boa escolha, já que é rápida (Figura 5.17, à direita) e mostra um comportamento interessante: há uma concentração do processo de busca em áreas promissoras, mas preservando a diversidade genética. Contudo, o desempenho superior do *ranking* exponencial é pago pelo aumento do tempo de execução: um aumento de 80%, para uma diferença de desempenho de 2%. É algo previsível, dada a sua maior complexidade. Na verdade a implementação poderia ser melhorada utilizando programação dinâmica, baixando a complexidade em n. Por outro lado, a sobrecarga imposta pela ordenação (implementada como um método padrão "*quick sort*") parece não ter uma influência relevante, visto que, sendo utilizada em comum pelos métodos *ranking* e truncamento, este último mostra-se mesmo como o mais rápido: há que considerar que é efectuada apenas uma vez por geração. No caso de se estar a utilizar um AG de estado estacionário, já haveria decerto lugar a um comportamento diverso.

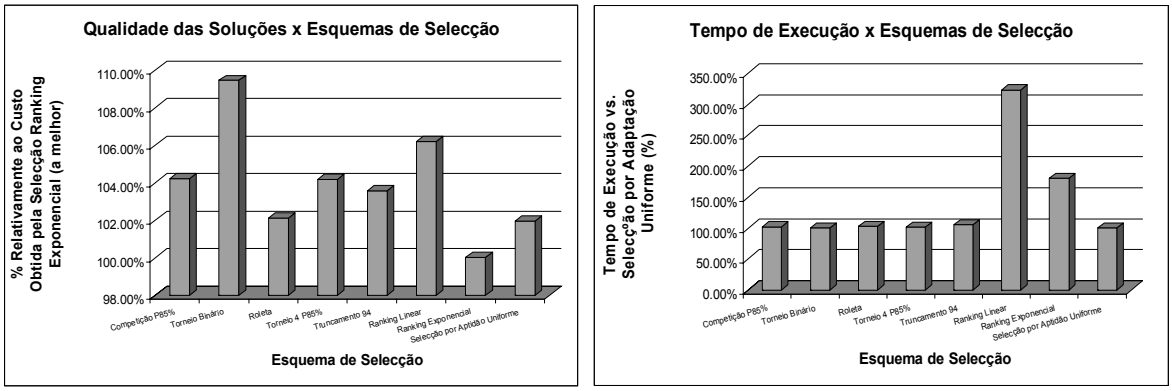


Figura 5.17. Qualidade e tempo de execução conforme o esquema de selecção utilizado num AG.

5.7.2 Fase 2 - Utilização de uma Arquitectura M-OLAP

Este teste foi executado com uma arquitectura simulada de 3 NSOs mais o nó base. Usaram-se os mesmos parâmetros genéticos empregues na fase 1 do teste, diferindo apenas no limite de 200 gerações e no facto de se usar o cubo A. Restringiu-se o teste executado aos esquemas de selecção Competição com P85%, Roleta, Truncamento 94, *Ranking* Exponencial e Selecção por Aptidão Uniforme. A Figura 5.18 mostra os resultados obtidos.

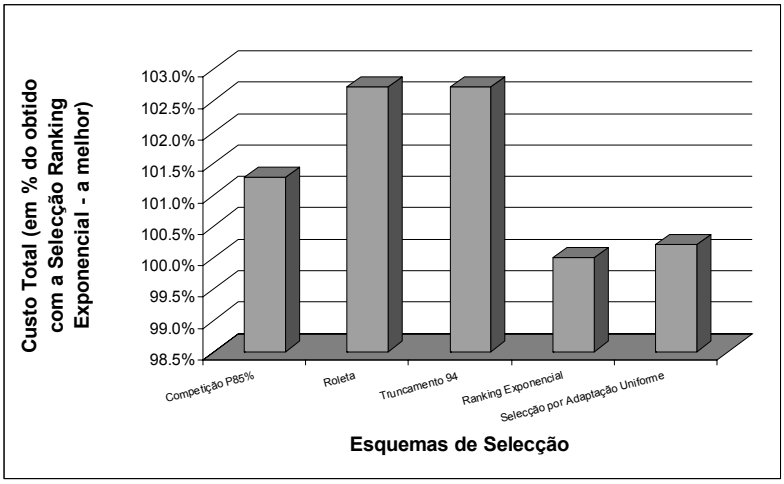


Figura 5.18. Qualidade para diversos esquemas de selecção quando um AG é aplicado à optimização de uma arquitectura M-OLAP.

Também, neste caso, se nota um impacto reduzido do tipo de selecção utilizado na qualidade das soluções obtidas, apresentando um comportamento semelhante ao observado na fase 1, excepto para o caso dos esquemas competição e roleta. Na verdade, o primeiro é agora mais competitivo e o segundo piorou decididamente. Este esquema mostra dificuldade em lidar com o aumento de complexidade do problema introduzida pela existência de vários nós: é o problema da "translação invariante", discutido no final da secção 5.6.3.

5.8 Avaliação dos Algoritmos Propostos

Os algoritmos propostos permitem, de uma forma simples, gerir a distribuição de estruturas multidimensionais, capitalizando as vantagens da computação e armazenamento distribuído, com baixos custos de administração. Como se viu, tendo como base o esquema do cubo, o perfil de

interrogações e respectivos nós de acesso, a arquitectura M-OLAP e conexões de rede, os algoritmos *greedy*, genéticos e de optimização por enxame de partículas foram capazes de propor uma distribuição de estruturas OLAP que minimizaram os custos de interrogação e manutenção. Isto melhora as propostas existentes de quatro maneiras distintas:

1. lida com valores de parâmetros reais relativamente a nós, rede de comunicação e à medida utilizada – tempo, claramente próxima da forma de aquilatar da satisfação dos utilizadores e do tamanho da janela de manutenção;
2. introduz o custo de manutenção na equação de custos a minimizar, relativamente a uma arquitectura OLAP distribuída;
3. para esta arquitectura, introduz propostas genéticas e de optimização por enxame de partículas para solucionar o problema da selecção e alocação de cubos, propondo diversas variantes e efectuando uma análise comparativa de desempenho;
4. usa, como função de aptidão para AGs e ODiEP e para cálculo do ganho (no algoritmo *greedy*), algoritmos de estimativa de custos (de interrogação e manutenção), que simulam a execução paralela das tarefas (utilizando o inerente paralelismo da arquitectura M-OLAP).

Os resultados experimentais parecem mostrar a superioridade da abordagem genética e ODiEP face à *greedy*, com especial relevo para a versão co-evolucionária do AG, na forma do algoritmo Co-Evol-AG M-OLAP. Se os algoritmos ODiEP revelaram ser competitivos face à abordagem genética padrão e consequentemente ao *Greedy* M-OLAP, a variante co-evolucionária do AG parece tirar partido da existência das múltiplas espécies, coisa que o ODiEP não consegue fazer com os enxames cooperativos. A variante Multi-Fase (ODiEP-MF M-OLAP) revelou um desempenho algo decepcionante, já que era expectável um benefício elevado, atendendo aos resultados dos testes reportados em problemas genéricos de carácter combinatório. Já a extinção em massa para a ODiEP parece ser interessante, desde que o intervalo de extinção seja convenientemente escolhido. Relativamente ao uso de operadores genéticos, os resultados dos testes parecem aconselhar apenas o uso do operador cruzamento para o ODiEP M-OLAP com uma percentagem de cruzamentos baixa (no caso, 20%).

Uma análise de complexidade, outra característica relevante de qualquer algoritmo de optimização, mostrou que os testes de aptidão aos indivíduos ou partículas impõem uma sobrecarga elevada de processamento, fazendo com que as variantes monopopulação sejam, efectivamente, mais rápidas. De qualquer forma, a análise de complexidade experimental evidenciou a fácil escalabilidade do AG e ODiEP em ambas as direcções: complexidade do cubo e número de nós da arquitectura M-OLAP. De facto, ambas as abordagens têm um tempo de execução a variar linearmente com a população, o

que, verificada a quase independência da qualidade das soluções com o número de partículas, faz pender a balança para a ODiEP. Mesmo com um número pequeno de partículas (e correspondente tempo de execução), a ODiEP consegue boas soluções. Mas, ainda assim, é preciso não esquecer que se trata, em ambos os casos, de uma dependência linear e não exponencial. Já no que toca à complexidade do esquema dimensional (e consequentemente, do número de subcubos possíveis), foi evidenciada uma dependência directa para o AG M-OLAP (e um aumento de tempo de execução um pouco abaixo do incremento do grau de complexidade), relação esta mais desfavorável para o Co-Evol-AG M-OLAP, mas, ainda assim, distante de uma dependência quadrática: apenas a meio caminho. Já quanto ao número de nós da arquitectura, verificou-se, para o AG M-OLAP, um aumento de tempo de execução abaixo do incremento do número de nós, sendo, no entanto, muito mais desfavorável para o Co-Evol-AG M-OLAP, que evidencia uma dependência acima de quadrática. Ainda relativamente à escalabilidade, quanto ao número de nós da arquitectura M-OLAP, há a acrescentar que as abordagens ODiEP mostram uma variação linear com o número de nós que, sabida a melhor qualidade das soluções conseguidas pelo ODiEP M-OLAP face ao AG M-OLAP, faria escolher o primeiro, para casos onde a complexidade do problema é muito elevada.

Para terminar, poder-se-á dizer que o algoritmo Co-Evol-M-OLAP deve ser o escolhido para situações onde se pretendam soluções de elevada qualidade, para casos de complexidade pequena ou média, ou mesmo elevada, se houver muito tempo disponível, ou empregando recursos computacionais poderosos, especialmente podendo usar processamento paralelo. Para casos complexos, ou quando o tempo de execução esteja a prémio, o algoritmo ODiEP M-OLAP deverá ser o eleito, já que mostrou uma complexidade semelhante ao AG M-OLAP, necessitando de um menor número de elementos na população, conseguindo, mesmo assim, melhores soluções.

As duas abordagens, a genética e por enxame de partículas, como foi referido, encerram virtudes, embora diversas, o que faz adivinhar a possível conveniência da sua fusão. Não na forma híbrida intentada na hibridação genética da ODiEP (com vantagens, é certo, mas de natureza limitada), mas onde pudessem utilizar, na sua individualidade, as suas especiais potencialidades. O processo deve ser dotado do arbítrio necessário à livre utilização de um qualquer método ou de uma sequência deles. De mais interesse seria, se outras heurísticas pudessem ser adicionadas, o que permitiria que o esquema usasse a combinação das técnicas mais adequadas e mesmo utilizá-las todas, em fases diferentes do processo de procura da solução. O processo ora antevisto seria, à primeira vista, simultaneamente elegante e eficiente, já que combinaria uma adaptabilidade ao problema e uma capacidade de utilização da heurística, que, a cada passo, se revelasse mais adequada. Desta forma, passaria a dispor-se de um conjunto de algoritmos devidamente estudados, cada um com especiais

virtualidades e limitações, cada um com aptidões específicas a um leque de casos, permitindo a sua utilização de forma isolada ou em combinações.

Se usados de forma isolada, a selecção do algoritmo pode ser efectuada pelo utilizador (administrador do DW), que, guiado por uma heurística capaz de indicar o mais apropriado à natureza do problema a tratar, deduzida e concebida através do estudo do comportamento de cada um, poderá empreender, de igual forma, um conjunto de testes preliminares para avaliar da sua real efectividade. Ter-se-ia uma situação como a apresentada na Figura 5.19, podendo ser adicionada com novas heurísticas de optimização, sendo assegurado, através da existência de um conjunto alargado de parâmetros e de serviços, o controlo total por parte do utilizador do estado e funcionamento dos diversos algoritmos, bem como a especificação do ambiente.

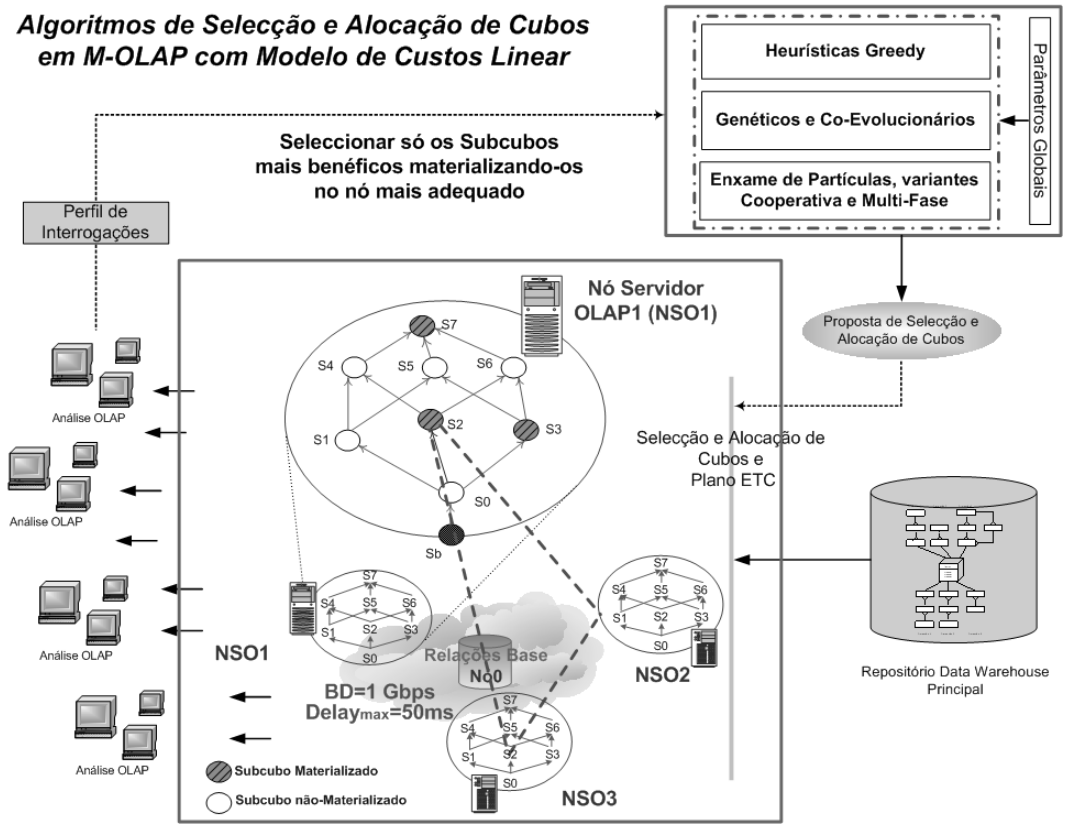


Figura 5.19. Utilização das heurísticas de selecção e alocação de subcubos em M-OLAP.

Na segunda situação, a utilização dos algoritmos em combinações, sobe-se a um nível de maior abstracção: não há indivíduos, partículas, *hill climbers*, ou outras quaisquer entidades capazes de procurar de uma forma cooperativa ou concorrencial a solução para um problema, mas há uma população de entidades de busca que podem ser instanciadas como indivíduos, partículas, ou outros. Estas empreenderiam a procura da solução, vestindo o "hábito" mais apropriado, assumindo a forma de uma partícula (integrando um enxame), de um indivíduo (de uma população genética) ou uma outra qualquer forma, consoante fosse mais adequado. A entidade de busca saberia quando mudar de forma e podê-lo-ia fazer livremente, empreendendo a sua metamorfose. O sistema inclui assim um esquema adaptativo que permite a utilização dos algoritmos em casos bastante díspares.

Numa e noutra situação o âmbito de utilização dos algoritmos poderia ser algo estendido e especializado. Sabida que é a limitação do modelo de custos até aqui utilizado, haveria toda a conveniência na utilização do modelo não linear distribuído e algoritmos de estimativa de custos respectivos (ver secção 3.3). Este novo modelo, ao permitir o suporte simultâneo de manutenção integral e incremental, poderá, eventualmente, permitir que o algoritmo tente empreender a afinação progressiva das estruturas materializadas, utilizando algum tempo disponível na janela de manutenção, para, além da manutenção incremental, empreender a recriação limitada de algumas estruturas, com eventual eliminação de outras: entrar-se-ia numa abordagem dinâmica, já que seria efectuada a adaptação progressiva das estruturas à mudança do perfil de interrogações.

Cada um dos algoritmos participantes no esquema metamorfose poderia ser ainda melhorado. É o caso do AG, onde a investigação dos mecanismos de reparação tipo V a incluir nos cruzamentos e o uso de *crossover* multi-ponto e multi-pais seriam características a adicionar; também a utilização da denominada procura local genética, uma heurística híbrida que combina as vantagens da procura baseada em população com a optimização local, poderá ser uma mais valia e merece ser alvo de investigação.

Capítulo 6

Selecção e Alocação de Cubos em Arquitectura M-OLAP com Modelo de Custos não Linear

6.1 Optimização por *Hill Climbers*

Um *hill climber* (HC) sugere a imagem de um montanhista, procurando o caminho para o topo da montanha. O paradigma é algo realista: na verdade, o processo de optimização pode assumir a forma de uma pesquisa, por jornadas sucessivas, do máximo ou mínimo da função objectivo. Um HC, na sua versão mais básica, empreende uma pesquisa tipo aleatória, onde, em cada jornada, é efectuada uma qualquer deslocação, sem aplicação de qualquer outra heurística, excepto a eventual aplicação de constrangimentos, que levarão à não admissão de soluções inválidas. Dito por outras palavras, nem o resultado da jornada actual (em termos de função objectivo), nem a deslocação empreendida terão qualquer intervenção na eleição do movimento a efectuar na próxima jornada, o mesmo se passando com qualquer heurística onde intervenha a localização futura. Esta forma de pesquisa deverá ser, certamente, bastante ineficiente, podendo ser melhorada, bastando para tal, por exemplo, aceitar apenas movimentos do *hill climber* que melhorem a sua posição, avaliada em termos da função objectivo. Com base nesta heurística pode construir-se um algoritmo denominado pesquisa local ou melhoramento iterativo; mas outras heurísticas mais eficazes podem ser concebidas, adoptando uma qualquer das denominadas metaheurísticas.

6.1.1 Algoritmo de Melhoramento Iterativo e Metaheurísticas

No algoritmo de melhoramento iterativo, é procurada, a cada passo, na vizinhança da posição (solução) actual, uma que melhore a qualidade da solução. Se uma tal posição for encontrada, torna-se a solução actual e a pesquisa local continua. Estes passos são repetidos até que nenhuma solução na vizinhança da solução corrente seja melhor, dizendo-se que o algoritmo terminou num mínimo local. A descrição formal do algoritmo por melhoramento iterativo é mostrada em Algoritmo 6.1. Como se pode verificar, trata-se de um algoritmo bastante simples. A única complexidade poderá advir da própria função *MelhorViz(s)*, que devolve uma solução melhor, na vizinhança da solução actual, se uma tal solução existir, ou a solução actual, de contrário. Quando esta última situação ocorre, dá-se a finalização do algoritmo.

A escolha de uma estrutura de vizinhança apropriada é crucial para o desempenho do algoritmo e depende especificamente do problema em consideração. A estrutura da vizinhança define o conjunto de soluções que podem ser alcançadas a partir de s, num único passo do algoritmo de pesquisa local.

Contudo, esta pesquisa apresenta dois problemas. Em primeiro lugar, o resultado depende fortemente da solução inicial. Por outro lado, e com maior relevância, o algoritmo pode ficar "preso" em mínimos locais. Se supusermos que a função objectivo deverá ser minimizada, poder-se-á obter, como solução, um mínimo local, de onde o *hill climber* não poderá sair, pois que só são admitidos movimentos que minimizem o valor da função objectivo.

```
Heurística  Melhoramento Iterativo

Escolher uma solução inicial s
begin
    s' ← gerar solução na vizinhança de s
    enquanto s' ≠ s fazer:
            s ← s'
            s' ← MelhorViz(s)
    fim enquanto
    return s
end
```

Algoritmo 6.1. Algoritmo simplificado da heurística por melhoramento iterativo.

Com Glover [Glover, 1986], surgem as metaheurísticas, técnicas que admitem degradar temporariamente a qualidade da solução, permitindo explorar o espaço das soluções mais extensivamente, evitando que o *hill climber* fique "preso" em óptimos locais.

As metaheurísticas mostraram ser métodos eficazes e capazes de resolver uma vasta gama de problemas reais. Possuem em comum as seguintes características:

- Incluem um procedimento para criar uma solução inicial, que serve de ponto de partida;
- Aplicam um método de pesquisa local assente num conceito de vizinhança para melhorar a solução inicial;
- Memorizam soluções ou características das soluções geradas durante o processo de pesquisa.

Segundo [Taillard et al., 1998], todas as metaheurísticas podem ser agrupadas sob uma denominação comum: "*Adaptive Memory Programming – AMP*". Este termo também é utilizado por [Glover & Laguna, 1997], uma vez que exploram a memória para construir uma nova solução e a melhoram, actualizando de seguida a memória com o conhecimento trazido pela nova solução. De entre a enorme quantidade de propostas nesta área, salientam-se: *simulated annealing* [Kirkpatrick et al., 1983], aceitação de limiar (*threshold accepting*) [Dueck & Scheuer, 1990], pesquisa tabu (*tabu search*) [Glover & Laguna, 1997], arrefecimento quântico (*quantum annealing*) [Finnila et al., 1994], [Das & Chakrabarti, 2005], uma variante do *simulated annealing*, método atravessar-entropia (*cross-enthopy method*) [de Boer et al., 2005], optimização por colónia de formigas e algoritmos genéticos, já atrás referenciados ou descritos. No contexto actual deste trabalho, vai descrever-se a primeira, pois que vai ser utilizada para optimização do problema da selecção e alocação de cubos.

6.1.2 *Simulated Annealing*

O *simulated annealing* foi proposto por Kirkpatrick, Gelat e Vecci em [Kirkpatrick et al., 1983] explorando uma analogia com a termodinâmica, ao simular o arrefecimento de um conjunto de átomos aquecidos - o processo real de *annealing* - que pode ser visto como o arranjo organizado das partículas de um sólido quando este é arrefecido de forma cuidadosa, desde a sua forma inicial fundida, até à forma final sólida. Na prática, o que se pretende é que o estado final seja o de um reticulado altamente estruturado, correspondendo a um estado de energia mínima.

O algoritmo *simulated annealing* inclui um mecanismo que vai diminuindo o grau de liberdade do processo de pesquisa, havendo assim o paralelo com o processo de *annealing* térmico. A pesquisa assume uma forma de um processo iterativo, onde cada *hill climber* se move de uma posição para outra, evitando estagnar em mínimos locais ao permitir-lhe movimentos que piorem a solução. Uma versão simples do algoritmo é apresentada em Algoritmo 6.2.

```
Heurística Arrefecimento Simulado

Escolher uma solução inicial s e uma temperatura inicial T0>0
Begin
    T ← T0
    s_best ← 0
    enquanto critério de paragem não for alcançado
        s' ← gerar solução na vizinhança de s
        Δ ← f(s') − f(s)
        se Δ <= 0 ou e^(−Δ/T) > rnd[0,1) então s ← s'
        se f(s) > f(s_best) então s_best ← s
        T ← T * α
    fim enquanto
    return s_best
End
```

Algoritmo 6.2.Algoritmo simplificado da heurística por *simulated annealing*.

A admissão de movimentos que degradam a solução depende de uma probabilidade que está relacionada com um parâmetro *T*, designado como *temperatura*. Em cada iteração, um vizinho s' da solução corrente s é gerado aleatoriamente e testada a variação da função objectivo, isto é $\Delta = f(s') - f(s)$. Se $\Delta < 0$ (o movimento conduziu a uma melhor solução, Mov^+), o algoritmo aceita a solução e s' passa a ser a solução corrente. Até agora nada de novo: descreveu-se a heurística de melhoramento iterativo. Mas, caso $\Delta > 0$ (está-se perante um movimento que degrada a qualidade da solução, Mov^-), a solução s' também poderá ser aceite, mas neste caso, com uma probabilidade $e^{-\Delta/T}$, permitindo ao *hill climber* escapar de mínimos locais.

Esta probabilidade assume duas consequências imediatas:

- Movimentos que impliquem valores baixos de delta serão admitidos mais frequentemente do que o contrário, ou seja, saltos que degradem muito a qualidade da solução terão uma probabilidade pequena de acontecer.
- A temperatura *T* será um factor que permite variar esta probabilidade, permitindo especificar o grau de exploração e focalização (secção 4.1) pretendido: para o mesmo delta, se *T* for elevado a probabilidade será maior (serão aceites movimentos que permitam uma maior degradação da qualidade da solução) e inversamente.

Pensando em termos do algoritmo, percebe-se o interesse de favorecer a exploração na fase inicial da pesquisa, ao permitir movimentos Mov^- com elevada degradação da qualidade da solução (e assim permitir a um *hill climber* escapar de fortes mínimos locais), conseguida pela especificação de um valor inicial T_0 elevado. No entanto, se este comportamento do *hill climber* persistir, o algoritmo continuará a exploração, sem nunca se deter verdadeiramente para uma inspecção mais demorada de cada região visitada, o que seria conseguido se a probabilidade de aceitar movimentos Mov^- fortes fosse baixa, implicando um valor baixo de *T*. Está-se perante dois objectivos conflituosos que importa, como sempre, harmonizar. Basta para tal especificar um valor inicial elevado para *T*, que depois vai sendo reduzido, até um valor final pequeno. Com as sucessivas iterações, à medida que *T* se aproxima de zero, reduz-se a probabilidade de aceitar movimentos que degradem a solução: $T \to 0 \Rightarrow e^{-\Delta/T} \to 0$. Desta forma, favorece-se a exploração do espaço de soluções na fase inicial da execução do algoritmo, situação que vai gradualmente sendo modificada a favor da focalização, à medida que a pesquisa se aproxima do final. Em resumo, a temperatura *T* assume um valor inicial T_0 elevado. Após um número fixo de iterações (o qual representa o número de iterações necessárias ao sistema para atingir o equilíbrio térmico a uma dada temperatura), a temperatura é gradualmente reduzida de α, tal que $T_n \leftarrow \alpha * T_{n-1}$, sendo $0 < \alpha < 1$. Este é um possível mecanismo de agendamento de arrefecimento, mas outros são possíveis, como p. ex. o simples decréscimo de *T* de um valor constante, ou um decréscimo sucessivamente mais pequeno, à medida que se aproxima o final do processo de pesquisa. O procedimento pára quando *T* chega a um valor próximo de zero e nenhuma solução que piore o valor da função objectivo é mais aceite, dizendo-se que o sistema está congelado: atingiu-se a estabilidade.

6.2 Redefinição do Problema da Selecção de Cubos

O modelo arquitectural é o mesmo que o abordado no capítulo anterior, mas foi aplicado um novo constrangimento: a duração da janela disponível para refrescamento das estruturas. O problema da selecção dos cubos apresentado na definição 5.1 da secção 5.1 tem de ser adaptado à nova realidade, obrigando à redefinição do problema da selecção de M em OLAP distribuído. Por outro lado, já que agora é imposto um novo constrangimento que era anteriormente parte da função objectivo, esta pode ser alterada, permitindo-se agora duas situações: a função objectivo incluir ou não os custos de manutenção, adicionalmente aos custos de interrogação. A nova definição (Definição 6.1), deverá assim considerar estas duas funções objectivo.

Definição 6.1: Selecção de M em OLAP distribuído com constrangimento espacial e temporal

Seja $I = \{I_{11}, I_{12}, ..., I_{Nn}\}$ o conjunto de interrogações com frequências de acesso $F = \{F_{11}, F_{12}, ..., F_{Nn}\}$ e extensões de interrogação $Ei = \{Ei_{11}, Ei_{12}, ..., Ei_{Nn}\}$; seja a frequência de actualização e extensão de actualização Fm e $Em = \{Em_1, Em_2, ..., Em_n\}$ respectivamente e seja S_{N_i} o espaço disponível para materialização de subcubos em cada nó OLAP *i* e J_r a duração máxima da janela de refrescamento. A solução para o problema de selecção do cubo a materializar é um conjunto $M = \{su_{11}, su_{21}, ..., su_{nN}\}$ com constrangimentos $\sum_j |su_{jN_i}| \leq S_{N_i}$ e $Cm(M) \leq J_r$ e tal que: a) o custo de resposta às interrogações I, $Ci(I,M)$ ou b) o custo de resposta às interrogações I e de manutenção de M, $Ci(I,M) + Cm(M)$, seja mínimo.

Para a estimativa de $Ci(I,M)$ e $Cm(M)$ vai utilizar-se o modelo não-linear distribuído descrito em 3.3.1, e os algoritmos de estimativa de custos descritos em 3.3.2 e 3.3.3, discutidas que foram as vantagens deste modelo, especialmente no que respeita ao suporte da estimativa dos custos de manutenção (ver conclusões em 3.3.7). Os custos estimados por estes algoritmos serão utilizados para cálculo da função de aptidão de cada *hill climber*.

6.3 Algoritmo Hill-Climbing com *Simulated Annealing* em Arquitectura M-OLAP (HC-SA M-OLAP)

6.3.1 Aplicação do Algoritmo ao Problema da Selecção e Alocação de Cubos em Arquitectura M-OLAP

Observando o Algoritmo 6.2, percebe-se que para o aplicar a um qualquer problema será necessário:

1. encontrar uma forma de codificar o problema;
2. gerar a solução inicial *s*;
3. definir a vizinhança *s´* de uma qualquer solução *s*;
4. definir o esquema de movimentação do HC (como alcançar uma solução *s´* na vizinhança de *s*);
5. como avaliar cada solução, o que permite calcular o delta;
6. especificar um valor T_0 e α.

A solução para a primeira das questões colocadas é simples: assumindo o paradigma espacial onde cada HC se movimenta, bastará supor um espaço n-dimensional discreto de localizações {0,1}. Assim, o HC poderá ocupar, numa qualquer dimensão, a posição 0 ou 1. Este espaço multidimensional pode ser mapeado para o espaço de soluções do problema da selecção e alocação de cubos: numa dada dimensão, se o HC está na posição 0, isso significa que o subcubo correspondente à dimensão não está materializado, e inversamente. O HC movimentar-se-á num espaço de dimensão N=nSCubos_do_lattice*nNós_OLAP. A posição do HC será codificada como uma cadeia binária de N bits. O esquema representa-se na Figura 6.1 (uma representação funcional do algoritmo) no canto inferior esquerdo. Só é representado o mapeamento dos subcubos materializados num NSO, bastando replicar cada cadeia binária para mapear cada um dos outros NSOs. O mapeamento espacial completo, como se percebe desta descrição, é em tudo idêntico ao concebido para o algoritmo M-OLAP ODiEP (Figura 5.4), mas agora referente à posição de cada HC. Todavia a semelhança acaba aqui. A entidade em busca e o seu comportamento são diversos.

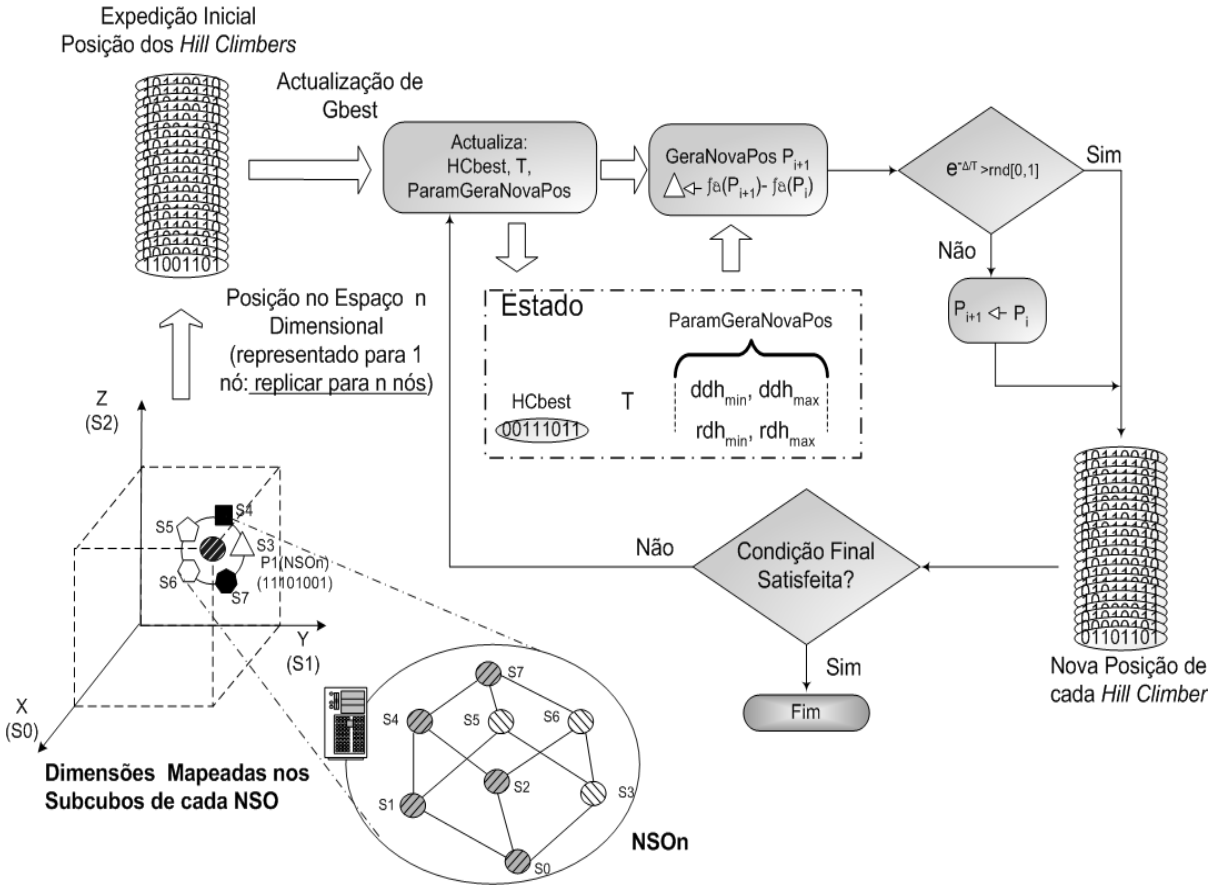


Figura 6.1. Esquema funcional do HC-SA M-OLAP.

Quanto à segunda questão, adopta-se a solução já utilizada para OEP e AG: gera-se a posição aleatoriamente. Em vez de um HC único, vai utilizar-se uma expedição (mostrada no canto superior esquerdo da Figura 6.1), onde cada HC irá pesquisar, de forma individual, o espaço de soluções, empreendendo-se uma pesquisa paralela. Na verdade, o algoritmo pode funcionar com um único HC, já que não há qualquer esquema de competição, partilha ou cooperação. No entanto, esta decisão foi motivada por duas razões: 1) adoptou-se uma forma idêntica à já implementada para a ODiEP, podendo, ao especificar-se uma população de um, obedecer ainda à forma "pura" do algoritmo; 2) a inclusão deste algoritmo no algoritmo metamorfose: é necessária a possibilidade de haver uma população E (expedição) de HC*s* para que o processo de metamorfose seja simples, já que, havendo um enxame de partículas e uma população de indivíduos, deverá haver uma contrapartida em termos de grupo relativa a outras entidades de busca.

Já relativamente à terceira das questões formuladas, a forma de definir a vizinhança de uma solução, percebe-se que, sendo $s = M$ (um conjunto de subcubos materializados), qualquer outro $s' = M' \neq M$ será uma solução na vizinhança de s. Poder-se-á é definir uma distância de Hamming máxima de vizinhança, *dhm*. Na prática, e visto este esquema sob o paradigma espacial, isto obrigará a que cada jornada do HC esteja limitada a determinado raio de acção máximo. Em termos do problema em mãos, isto implica que o número de subcubos a materializar e a desmaterializar em cada iteração é limitado, podendo, no entanto, ser variado pela especificação de um valor diferente.

A quarta questão, a movimentação do HC (mostrada no segundo rectângulo da Figura 6.1), resulta de forma imediata da definição de vizinhança descrita anteriormente: em cada iteração vai permitir-se a desmaterialização de *dsc* subcubos, $ddh_{\min} \leq dsc \leq ddh_{\max}$, e a rematerialização de *rsc* subcubos $rdh_{\min} \leq rsc \leq rdh_{\max}$, seleccionando ambos de forma aleatoriamente no intervalo especificado. Mais uma vez, o balanceamento da relação exploração x focalização vai ser alterado ao longo do processo de pesquisa (a cada *faidr* iterações – frequência de alteração dos intervalos de desmaterialização e rematerialização), diminuindo o intervalo de variação possível de *dsc* e de *rsc*. Na prática, selecciona-se aleatoriamente um nó e subcubo a desmaterializar, mudando o bit correspondente da cadeia binária que representa a posição do HC de 1 para 0, repetindo o processo *ddhm* vezes, desde que haja subcubos possíveis a desmaterializar. Segue-se o processo de rematerialização realizado de forma idêntica. Todos estes parâmetros são especificados pelo utilizador, podendo assim efectuar uma sintonização do funcionamento do algoritmo.

A avaliação de cada solução (a função fa da figura), a quinta questão a solucionar para utilizar o algoritmo, deriva, de forma imediata, da função objectivo definida. Por conseguinte, vai adoptar-se uma abordagem dual: 1) considerar um objectivo idêntico ao assumido nos capítulos 4 e 5, a minimização da soma dos custos de interrogação e manutenção; 2) só a minimização dos custos de interrogação (algo mais próximo do interesse imediato dos utilizadores), uma vez que vai assumir-se, em qualquer das funções objectivo, um constrangimento temporal de manutenção. Para a avaliação de uma e de outra função objectivo vão usar-se os algoritmos de estimativa de custos descritos em 3.3.2 (sendo utilizada a sua implementação com execução paralela simulada de tarefas) e 3.3.3, adoptando o modelo de custos não linear distribuído descrito em 3.2.1.

Quanto à última das questões colocadas, que se prende com o balanceamento da exploração e focalização, através da especificação do valor inicial de T e α, vai permitir-se a sua indicação pelo utilizador, bem como a indicação da frequência de actualização (número de iterações que implicam a

diminuição de *7*). Para facilitar, e dada a já enorme profusão de parâmetros de controlo de funcionamento, cujo valor importa afinar para optimizar o processo de pesquisa do algoritmo, permite-se também que o utilizador deixe que o próprio algoritmo calcule o valor de T_0, através da eq. 6.1, apresentada abaixo:

$$T_0 = \frac{f(M : M = \varnothing) - \frac{\sum_{hc \in E_0} f(M_{hc})}{\# E_0}}{\frac{\sum_{hc \in E_0} \# M_{hc}}{\# E_0}} \quad \text{(eq. 6.1)},$$

situação em que α e fuT (a frequência de actualização de T) são calculados de forma a que, próximo do final do processo de pesquisa, seja assegurada a congelação.

6.3.2 Descrição Formal do Algoritmo HC-SA M-OLAP

O algoritmo *hill climbing* com *simulated annealing* aplicado à selecção e alocação de cubos numa arquitectura M-OLAP (HC-SA M-OLAP), é mostrado em Algoritmo 6.3. Uma análise breve permite verificar que são empregues as soluções descritas na secção anterior. A sombreado mais escuro mostram-se os blocos que correspondem:

1. ao reposicionamento dos HCs gerados em locais, cuja solução não viole o constrangimento espacial colocado;
2. ao mecanismo de arrefecimento (variação de T) e à actualização dos parâmetros de controlo do esquema de movimentação dos HCs;
3. o próprio processo de movimentação de cada HC;
4. a aceitação ou rejeição do novo movimento dos HCs.

```
Input: L  // lattice com todas as combinações de granularidade, dependências,
          // tamanhos, dimensões e hierarquias
       I={I1, ..., In}, Ei={Ei1, ..., Ein}, Em={Em1, ..., Emn}, Fm, E=(E1... En.)
          // perfil de interrogações e actualizações: frequência e extensão
       Pas // Parâmetros do algoritmo de arrefecimento simulado: NumHC, NumIterac,
          // DefMovim, T0, α, tpgerac
       Pb // parâmetros base: S={S1, ..., Sn} (Espaço disponível para materialização),
          // Tr (tipo de reparação), nNSO (número de NSOs), tamJanelaMax
       P=(P1... Pn.); X=(X1... Xn.), E=(E1... En.)  // Capacidade de Processamento e
          // espaço por nó e parâmetros de conexões
Output: M  // conjunto de subcubos a materializar e sua localização
Begin
 1. Inicialização:
  Inicializar parâmetros e serviços do sistema; // carregar parâmetros Pas,
    // perfil de carga, lattice e esquema dimensional
  E←{ }; d←NSCubos(L)*nNSO  // expedição de hill climbers vazia-; d=num. dim. Do
    // espaço de pesquisa (= número de subcubos*número de NSOs)
  D←{d1, ..., dd}; frozen←false; // espaço d dimensional; no início o sistema não
    // está congelado: aceita movimentos que degradam a qualidade da solução
 2. Geração, reparação, avaliação e mostra do estado da expedição inicial de
  hill climbers:
  Enquanto (CostManut(hc))>tamJanelaMax Fazer: // enquanto posição do hill
    // climber gerado não satisfizer o constrangim. temporal de manutenção
    Repetir NumHC Vezes: // vai gerar a expedição com NumHc de hill climbers e
      // posicioná-los numa zona que não viole qq. constrangimentos
      hc←GeraHC(d) // gera cada hill climber numa posição aleatória
      ParaCada nó n Em M-OLAP, Fazer:
        pos_nó=pos(hc, n); // pos(hc,n) devolve as coordenadas do hill climber
          // correspondente às dimensões mapeadas no nó n da arq. M-OLAP
        Se (Tamanho(M (pos_no))>Sn) Então ReposicionaHC (pos_no, Tr); // se hc
          // gerado em posição cuja solução de materialização viola constrangimento
          // espacial, vai repará-lo, aplicando Alg. 4.8 ou corr. hc de Alg. 4.6
      Próximo n
      E←E∪hc; // adiciona o hill climber gerado à expedição
      Fitness(hc)=f (hc, I, Ei, Fi, Em, Fm, X); // aptidão do hill climber
        // (f calculada aplicando Algoritmo 3.9, Algoritmo 3.10 ou Algoritmo 3.11)
      Se Fitness(hc)>hcBest Então hcBest ← Fitness(hc); // calcula hcBest
        // relativo à expedição inicial hcBest não é realmente necessário ao
        // funcionamento do algoritmo, mas é utilizado para mostrar visualmente
        // a melhor posição atingida
    FimRepetir
  FimEnquanto
  MostrarEstado (E); // mostra estado instantâneo da expedição e hcBest: quantos
    // hc's propõem a materialização de um determinado subcubo em cada nó da arq.
    // M-OLAP e hcBbest, correspondendo à melhor posição de um qq. hill climber
 3. Movimentação da expedição, avaliação e mostra do estado:
  Iter←0; // contador do número de iterações
  Enquanto ( Iter<NumIter)  Fazer:  // condição de finalização é o
      // número de iterações
    T ← update(T); // faz variar T de T0 até T→0, a cada fuT iterações,
      // sendo o sistema congelado ao fim de frozenI iterações
    Se iter>frozenI OU T<frozenT Então frozen←true; // ao fim de frozenI
      // iterações ou quando T inferior a patamar frozenT, o sistema fica congelado
```

Algoritmo 6.3. Algoritmo HC-SA M-OLAP.

Algoritmo 6.3., HC-SA M-OLAP, continuado da página anterior.

```
    interv_Max_Desmat ← update(interv_Max_Desmat); // act. interv. de distância
     // de hamming máxima de desmaterialização a cada fIh iterações
    interv_Max_Remat←update(interv_Max_Remat); // act. interv. de distância de
     // hamming máxima de rematerialização a cada fIh iterações
    wU←rnd(interv_Max_Desmat); uR←rnd(interv_Max_Remat); // gera distância de
     // hamming para desmaterialização (wU) e para rematerialização(wR)
     // vai movimentar cada hill climber, aplicando o esquema de movimentação definido
    ParaCada hc Em E, Fazer:
     Enquanto (CostManut(hc))>tamJanelaMax Fazer: // enquanto nova posição do
        // hill climber não satisfizer o constrangim. temporal
       // comutar wU bits de 1 para 0 (movimentar o hill climber nas correspondentes
       // dimensões de posição 1 para 0, desmaterializar os subcubos)
       Repetir wU Vezes:
        d←rnd(D); // selecciona aleatoriamente nó e subcubo a desmaterializar:
           // subcubo seleccionado deve estar materializado
        hc(d) ←0; // comuta o bit da string que reresenta a posição do hill climber
           // de 1 para 0
       FimRepetir
       // comutar wR bits de 0 para 1 (movimentar o hill climber nas correspondentes
          // dimensões de posição 0 para 1, rematerializar os subcubos)
       Repetir wR Vezes:
        d←selec(D);// selecciona nó e subcubo a rematerializar com probabilidade
           // proporcional ao espaço disponível em cada nó ou à relação espaço
           // disponível/espaço total em cada nó; o subcubo a rematerializar não pode
           // implicar solução que viole o constrangimento espacial
        hc(d) ←1; // comuta o bit da string que representa a posição do hill climber
           // de 0 para 1
       FimRepetir
     FimEnquanto
     // aceitar ou rejeitar movimento de hill climber
     delta ← f (hc(iter-1), I, Ei, Fi, Em, Fm, X) - f (hc(iter), I, Ei, Fi,
      Em, Fm, X) // calcula o delta correspondente ao movimento do hill climber
     Se delta > 0 E frozen=true OU delta > 0 E rnd([0,1])>e^(-Δ/T) Então
      hc(iter) ← hc(iter-1); // repõe posição anterior, não aceitando a nova
         // posição do hill climber
     Fitness(hc)= f (hc, I, Ei, Fi, Em, Fm, X);
     Se Fitness(hc)>hcBest Então hcBest ← Fitness(hc); // actualiza hcBest
    FimPara
    MostrarEstado (E); // mostra estado instantâneo da expedição e hcBest:
       // quantos hc's propõem a materialização de um determinado subcubo em cada
       // nó da arq. M-OLAP e hcBbest, correspondendo à melhor posição de um qq.
       // hill climber
    iter←iter+1; // incrementa o número de iterações
   FimEnquanto
  4. Devolver resultado:
   Return M pos(hcBest); // retorna o M correspondente à melhor posição atingida por
      // um qualquer hill climber da expedição
End
```

6.3.3 Teste Experimental Simulado

O teste experimental foi demorado, pois que, mais uma vez, e dada a natureza estocástica da heurística, cada teste foi repetido dez vezes, sendo depois calculado o valor médio dos valores obtidos. O algoritmo desenvolvido foi integrado como um componente no algoritmo metamorfose, a descrever na próxima secção. Na verdade, como se verá adiante, cada heurística integrante do algoritmo metamorfose pode ser aplicada isoladamente, bastando especificar um valor adequado para o parâmetro que controla o desencadear das metamorfoses.

Para o teste foi utilizado o cubo A, uma arquitectura M-OLAP de 3 nós e de 6 nós (mais o nó base), diversos conjuntos de interrogações geradas aleatoriamente com um número variável de interrogações, considerando-se custos de manutenção incremental, para o cálculo da aptidão das soluções propostas. Com isto pretende-se testar o impacto dos parâmetros seguintes no desempenho do algoritmo: 1) número de iterações, 2) valor de T, 3) número de HCs, 4) número de interrogações colocadas, e 5) numero de nós da arquitectura M-OLAP.

Foram realizados vários testes preliminares para afinação de vários outros parâmetros de funcionamento, cujos valores seleccionados se apresentam na Tabela 6.1.

Tabela 6.1. Valores especificados para diversos parâmetros do HC-SA M-OLAP.

ddh_{min}	ddh_{max}	Rdh_{min}	rdh_{max}	DecrementoD	DecrementoR	tpGeracPosInic
4	9	3	8	1	1	Aleatória

Outros parâmetros como a frequência de actualização de intervalos de desmaterialização e rematerialização (*faidr*), a frequência de actualização de T (*fuT*) e número da iteração a partir da qual o processo de *simulated annealing* se considerava congelado, são alterados de acordo com os valores escolhidos, em cada teste, para os parâmetros associados.

No primeiro teste procurou-se avaliar o impacto do número de iterações na qualidade das soluções obtidas e no tempo de execução do algoritmo. Utilizaram-se 20 HCs e T_0=30. Na Figura 6.2 revela-se a evolução dos resultados obtidos. Como se pode ver no gráfico da esquerda, a qualidade das soluções tem, no início, uma melhoria rápida, até cerca das 100 iterações, seguida de uma evolução mais lenta, sendo muito pequena a partir das 300 iterações. Aliás, um outro teste em que se permitiram 1000 iterações, revelou uma melhoria pouco significativa na qualidade das soluções (3435 para 500 iterações x 3411 para 1000 iterações). Por outro lado, uma vez que há uma relação linear

entre o número de iterações e o tempo de execução (Figura 6.2, à direita), o aumento de tempo de execução não se revela compensador.

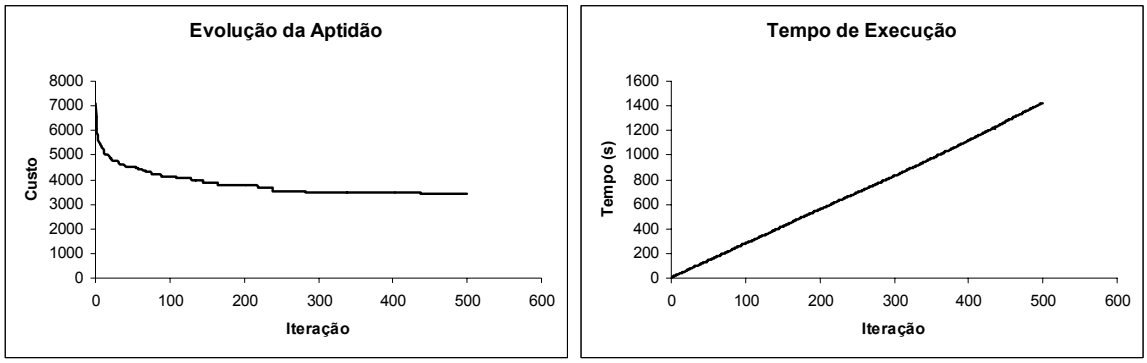


Figura 6.2. Impacto do número de iterações na qualidade e tempo de execução do HC-SA M-OLAP.

No segundo teste procurou-se avaliar o impacto do valor de T_0 na qualidade das soluções obtidas, sendo limitado o número de iterações a 300 (de acordo com análise custo x benefício obtida da análise da Figura 6.2). O resultado do teste mostra-se na Figura 6.3.

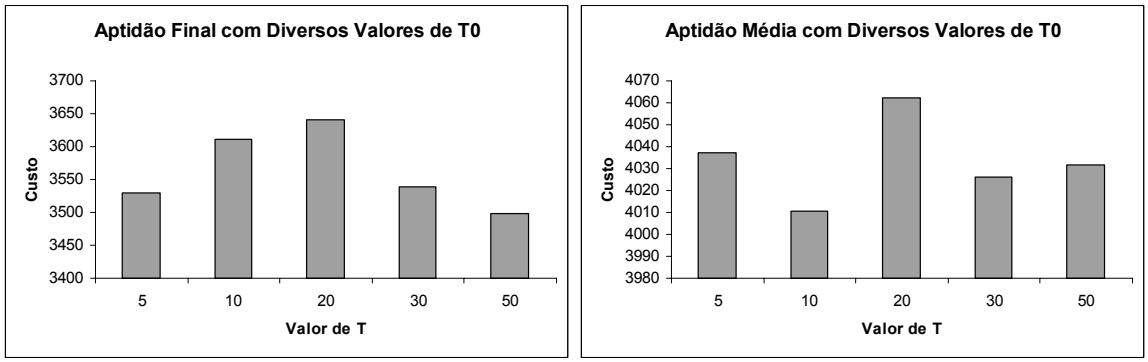


Figura 6.3. Impacto de T_0 no desempenho do HC-SA M-OLAP.

Uma conclusão que se pode tirar é que o desempenho do algoritmo parece ser bastante independente do valor de T_0 (1.2% para a aptidão média e 4% para a aptidão final). Por outro lado, os gráficos não permitem deduzir um padrão de comportamento, já que exibem uma variação não monotónica. Observando um e outro gráfico, o valor de T_0=30 parece ser aquele que assegura um melhor compromisso entre a qualidade da solução final e a média, pelo que foi o utilizado nos restantes testes.

Outra característica que importa avaliar, relativamente ao comportamento do algoritmo, é o impacto do número de HCs na qualidade das soluções obtidas e no tempo de execução. Os resultados obtidos mostram-se na Figura 6.4. Como se pode verificar no gráfico da esquerda, o impacto do número de HCs na qualidade das soluções é reduzido (apenas 2.7%), o que se compreende, porquanto não existe qualquer mecanismo de competição ou cooperação. Cada HC actua independentemente de qualquer outro e não há qualquer processo de aprendizagem. Por outro lado, analisando o gráfico da direita, percebe-se que há um crescimento quase linear do tempo de execução com o número de HCs. Estas duas constatações permitem concluir que é de interesse que o número de HCs seja pequeno (soluções boas, obtidas em tempos bastante inferiores).

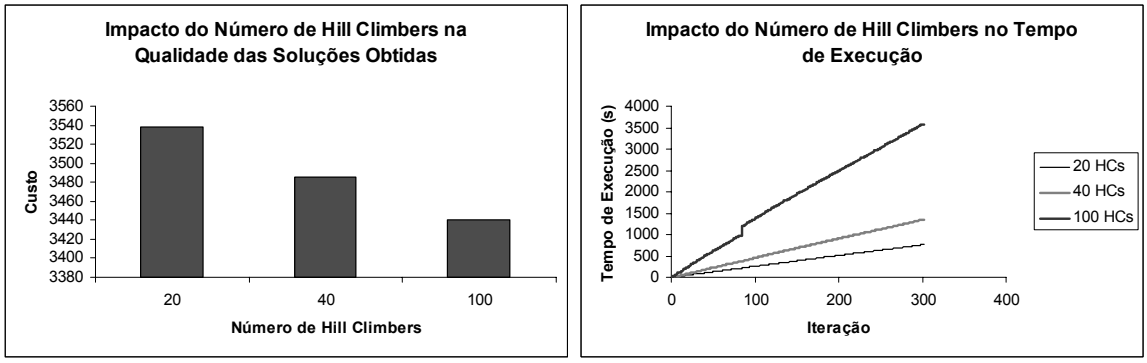


Figura 6.4. Impacto do número de *hill climbers* no desempenho do HC-SA M-OLAP.

Finalmente, impõe-se avaliar a escalabilidade do algoritmo HC-SA M-OLAP, no que concerne ao número de interrogações aplicadas e, especialmente, já que estamos a utilizar o algoritmo numa arquitectura M-OLAP, avaliar o impacto do número de nós da arquitectura no tempo de execução. Assim, para o primeiro caso, utilizaram-se conjuntos de 30, 60 e 90 interrogações. Os tempos de execução observados mostram-se na Figura 6.5 (à esquerda). O gráfico mostra uma quase independência do tempo de execução face ao número de interrogações colocadas. O gráfico da direita mostra o impacto do número de nós da arquitectura M-OLAP. Um aumento para o dobro no número de nós, motivou um aumento reduzido no tempo de execução: para o tempo de execução das 300 iterações houve apenas um aumento de 10.6%. Uma e outra observações mostram que o algoritmo HC-SA M-OLAP é escalável, podendo lidar com muitas interrogações e arquitecturas de muitos nós. Adicionado o facto já demonstrado da fácil escalabilidade do algoritmo *hill climbing* com *simulated annealing* em termos de dimensionalidade do cubo [Kalnis et al., 2002b], e estamos em presença de um algoritmo capaz de escalabilidade em todos os eixos.

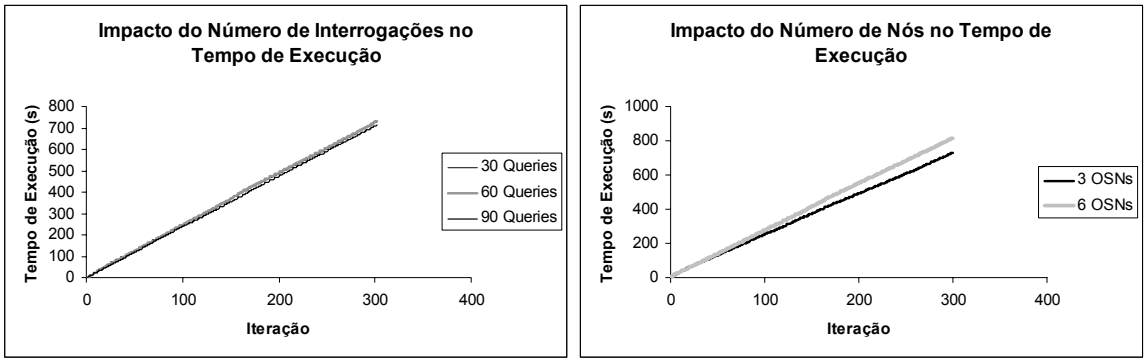


Figura 6.5. Escalabilidade do algoritmo HC-SA M-OLAP.

6.4 Algoritmos Híbridos, Miméticos e Multi-Forma

Os algoritmos híbridos, miméticos e multi-forma, embora diferentes, têm em comum o facto de serem combinações ou misturas de alguns outros. Os dois primeiros são algo similares, utilizando uma combinação de duas estratégias de busca, sendo muitas vezes denominados de algoritmos genéticos de pesquisa local ou mesmo algoritmos evolucionários híbridos, sem ser feita, explicitamente, a sua distinção de outros métodos de hibridação. O terceiro tipo de algoritmos utiliza três (ou mais) estratégias de busca. Mas a diferença não se restringe apenas ao número de algoritmos utilizados, mas à própria forma como eles operam. Os primeiros utilizam ambas as estratégias de pesquisa simultaneamente (dentro da mesma iteração principal): todas as entidades de busca (EB) usam uma estratégia e depois outra, sequência que é repetida sucessivamente. O último algoritmo tem uma granularidade operativa a um nível mais fino: cada algoritmo componente opera não sobre toda a população de cada vez, mas ao nível do EB. Desta forma, cada AB pode actuar como um qualquer dos algoritmos participantes, mudando a sua forma e comportamento, podendo prosseguir a busca da solução sob um de vários "hábitos", operando-se uma transmutação de uma para outra forma de EB através de uma operação denominada metamorfose, sob controlo da própria entidade (esta decide quando irá ocorrer).

6.4.1 Algoritmos Híbridos e Miméticos

Embora exista alguma indefinição quanto à distinção entre estes algoritmos e os genéticos, parece que a principal característica diferenciadora será uma deslocação do próprio paradigma da estratégia de optimização: o primeiro centra-se na evolução biológica, assistindo-se, no segundo, a uma deslocação para a evolução cultural. O biólogo Richard Dawkins [Dawkins, 1976] indica que, por analogia à transmissão genética, a transmissão cultural é o fluxo de informação num processo cultural evolucionário. A contrapartida dos genes serão os "memes", na evolução cultural. Estas unidades de transmissão cultural estão sujeitas a selecção: p. ex. ideias opostas que competem, pois que o cérebro humano tem uma capacidade limitada (em número) de suportar ideias. Durante a transmissão, ocorrem variações (mutações) nos memes (muitas vezes de grande extensão): as ideias são modificadas, substituídas ou postas noutra perspectiva. Embora um algoritmo mimético [Moscato, 1999] seja similar a um algoritmo genético híbrido, parece haver uma distinção clara, principalmente quanto ao grau de elaboração do esquema de hibridação que: a) deve permitir um aumento de eficiência, b) deve mudar a forma de ver a optimização – já não apenas orientada a objectivos, mas onde os indivíduos sejam vistos como entidades de busca conscientes, sujeitas a cooperação e competição e c) devem utilizar mecanismos de diversificação que permitam a introdução de ideias inovadoras. Num algoritmo mimético, tal como proposto em [Merz & Freisleben, 1999], [Merz & Freisleben, 2000]:

1. todos os indivíduos da população representam óptimos locais, o que é assegurado pela aplicação de pesquisa local depois da aplicação dos operadores evolucionários;
2. os operadores cruzamento e mutação são aplicados independentemente um do outro;
3. as heurísticas construtivas *greedy* podem ser utilizadas para geração das soluções iniciais;
4. um esquema de diversificação é utilizado, algo que actua como uma meta mutação (se a população convergir, todos os indivíduos sofrem mutações, excepto o mais apto), sendo aplicada uma heurística de pesquisa local para assegurar que todos os indivíduos sejam óptimos locais;
5. a selecção para a reprodução é efectuada de uma forma puramente aleatória, mas a nova população não substitui automaticamente a anterior, mas é adicionada à população actual, formando uma população temporária de onde são seleccionados os μ melhores indivíduos, sendo eliminadas cópias idênticas.

6.4.2 Ciclo de Vida e Metamorfoses

Trata-se de uma abordagem heurística em que o algoritmo tenta reproduzir o ciclo de vida dos indivíduos desde o nascimento, até à maturidade e reprodução. Todas estas morfologias e comportamentos do fenótipo se manifestam dentro do mesmo genoma. Muitas vezes, este processo gera transformações drásticas do indivíduo, uma manifestação multi-forma sob o mesmo genótipo, tentando adaptar-se a um ambiente ou objectivo particular. Alguns ciclos de vida são eventos não repetíveis, determinados por uma fase particular da vida em termos de objectivo, como é o caso da reprodução. Mas há outras alterações que são recorrentes, tal como estações de reprodução, ou podem ser desencadeados por factores ambientais.

Esta mesma abordagem pode ser transposta para os algoritmos de optimização, gerando uma heurística de busca auto-adaptativa, na qual cada EB (contendo a solução candidata) pode decidir se, e quando prefere assumir formas integrantes da população de algoritmos diferentes. Em [Krink & Løvbjer, 2002], os autores propõem um ciclo de vida com três algoritmos (Algoritmo 6.4).

Uma inspecção rápida a esse algoritmo permite ver que cada EB pode ser um indivíduo de uma população de um AG, uma partícula de um enxame num algoritmo OEP, ou um HC de um algoritmo de *simulated annealing*. As vantagens deste "*modus operandi*" serão de vária ordem:

- a auto-adaptação permitirá ao algoritmo a aplicação a um espectro mais alargado de casos, aproveitando as especificidades de cada um dos algoritmos integrantes;
- vão possibilitar a emergência de qualidades novas não existentes em qualquer dos participantes;
- vão permitir a inclusão de outros tipos de entidades de busca, bastando, para tal, acrescentar as estruturas de dados de suporte à nova população e o módulo gerador do comportamento da nova EB além do esquema encarregado das necessárias metamorfoses;
- finalmente, será possível dotar o algoritmo de uma grande versatilidade, já que se poderá especificar a forma inicial das EB, qual a sequência de metamorfoses e quando estas ocorrem, que formas incluir, e mesmo utilizar apenas uma das formas (bastando para tal indicá-la como forma inicial e especificar que a metamorfose será desencadeada por um evento não realizável, por exemplo, ao fim de um número de iterações superior ao número total de iterações admitidas).

```
Algoritmo Multi-Forma
1. Inicialização:
   Gerar um enxame de partículas // as entidades de busca nascem como
            // partículas
2. Iteração Principal CicloDeVida:
 Enquanto critério de paragem não verificado, Fazer:
    Para Cada entidade_busca Em população, Fazer:
     Calcular aptidão(entidade_busca);
     Mudar Estado_Ciclo_Vida se não houver melhoria recente;
    Para (OEP entidades_busca) Fazer:
     Movimentação de partículas: aplicar regras e equações da dinâmica
       das partículas
    Para (AG entidades_busca) Fazer:
     Seleccionar nova população;
     Fazer cruzamentos;
     Aplicar mutações na população;
    Para (HillClimbers entidades_busca) Fazer:
     Procurar possíveis soluções na vizinhança, movimentando o hill
       climber;
     Calcular aptidão da nova solução (posição);
     Aceitar a nova solução (posição) com probabilidade p;
 Fim Enquanto
3. Retornar o resultado: a melhor solução de uma qq. entidade_busca
```

Algoritmo 6.4. Algoritmo Multi-Forma, adaptado de [Krink & Løvbjer, 2002].

Em resumo, pode dizer-se que o algoritmo multi-forma revela uma mudança de modelo e foco de operação. Por um lado, tem-se uma população de soluções que são sujeitas à actuação dos algoritmos participantes um de cada vez, mas todos em simultâneo. Por outro lado, o foco é agora a EB (não a totalidade da população), não sendo cada algoritmo participante a tomar decisões quanto a cada EB; pelo contrário, este decide quando sofrer a metamorfose e assumir cada nova forma. Desta maneira, e num dado instante, pode ter-se uma população de EBs composta por um número variável de EBs em cada uma das possíveis formas. Se a EB continuar a melhorar a sua aptidão, pode manter a sua forma actual. Mas, se decorrido um número de iterações, não conseguir melhorar a sua solução, pode assumir outra forma, tentando uma melhor adaptação ao ambiente actual (tipo de problema, estado da sua resolução e solução actual).

6.5 Selecção e Alocação de Cubos em M-OLAP com Algoritmo Metamorfose

A denominação algoritmo metamorfose deriva de forma directa da operação que confere a especialidade ao algoritmo: a metamorfose que permite a transmutação da entidade de busca, procurando a sua adaptação. Um esquema do algoritmo é mostrado na Figura 6.6.

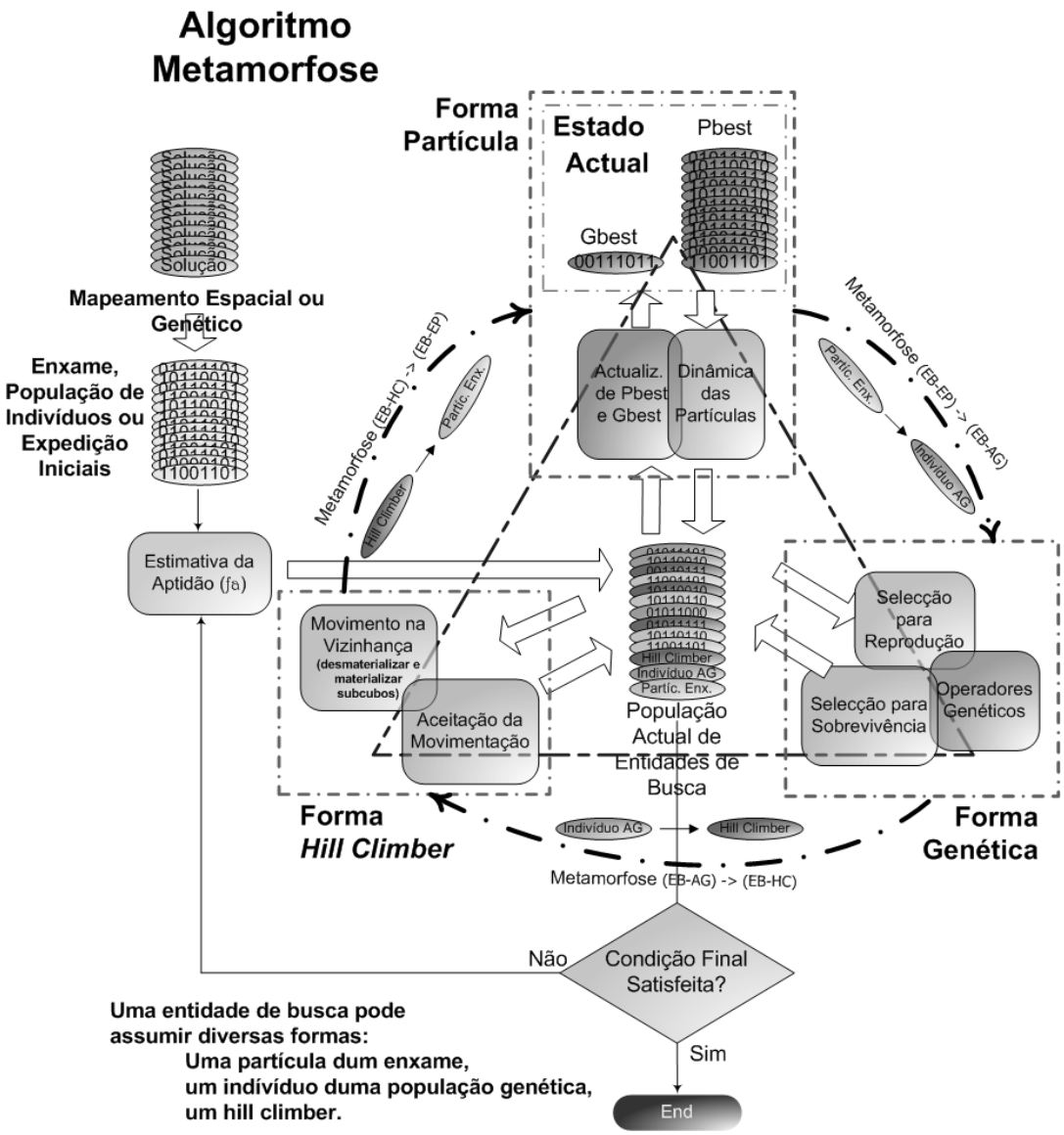


Figura 6.6. Algoritmo metamorfose: a entidade de busca muda a sua aparência e comportamento.

Na Figura 6.6, a seta circular simboliza as sucessivas metamorfoses que são desencadeadas por uma condição de impasse. Quando a forma corrente da EB, durante um número especificado de iterações, não consegue qualquer melhoria da solução, deve sofrer uma metamorfose, passando a assumir uma nova forma. Desta forma, a população corrente de EBs num dado momento é uma mistura dos possíveis tipos: uma partícula dum enxame (EB-EP), um indivíduo dum AG (EB-AG) ou um HC (EB-HC).

6.5.1 Arquitectura do Algoritmo MetaMorf M-OLAP

Na Figura 6.7 é mostrado um diagrama da estrutura estática da arquitectura do algoritmo MetaMorf M-OLAP. Como classes principais teremos:

1) O conjunto das classes correspondentes às populações de cada forma da EB e os métodos que vão gerar o respectivo comportamento:
    a. Enxame_de_Partículas, relativa ao enxame de partículas;
    b. Indivíduos, correspondente à população de indivíduos do algoritmo genético padrão;
    c. Indivíduos_Coop, relativa às espécies dos indivíduos do algoritmo genético co-evolucionário;
    d. Expedição, correspondente à população de hill climbers.
2) A classe Entidade_de_Busca, uma super-classe, relativa às entidades de busca, responsável pelo estado global e consequentemente pelo controlo do desencadear do processo de metamorfose e verificação de consistência.
3) A classe M_OLAP_Estado, que permite mostrar o estado actual e o melhor até ao momento relativo às entidades de busca.
4) A classe M utilizada para avaliar uma solução de distribuição de subcubos proposta pelas entidades de busca.

Muitas outras classes foram agrupadas em subsistemas como parâmetros base dos diversos algoritmos e relativos ao ambiente, estrutura do lattice e serviços de saída de dados, de entre outras, de forma a não tornar o diagrama demasiado complexo. Vamos explicar seguidamente as opções de concepção tomadas e a respectiva justificação. Comecemos pelas classes correspondentes às diversas formas da EB.

Como se pode ver no esquema apresentado na Figura 6.6, o algoritmo metamorfose permite três formas diferentes para a entidade de busca: partícula, individuo e HC. Inclui, assim, na prática, os algoritmos de OEP, genético e *hill climbing* com *simulated annealing*. Em termos de concepção, cada algoritmo é um componente bastante isolado, o que permite incluir facilmente outras heurísticas no futuro.

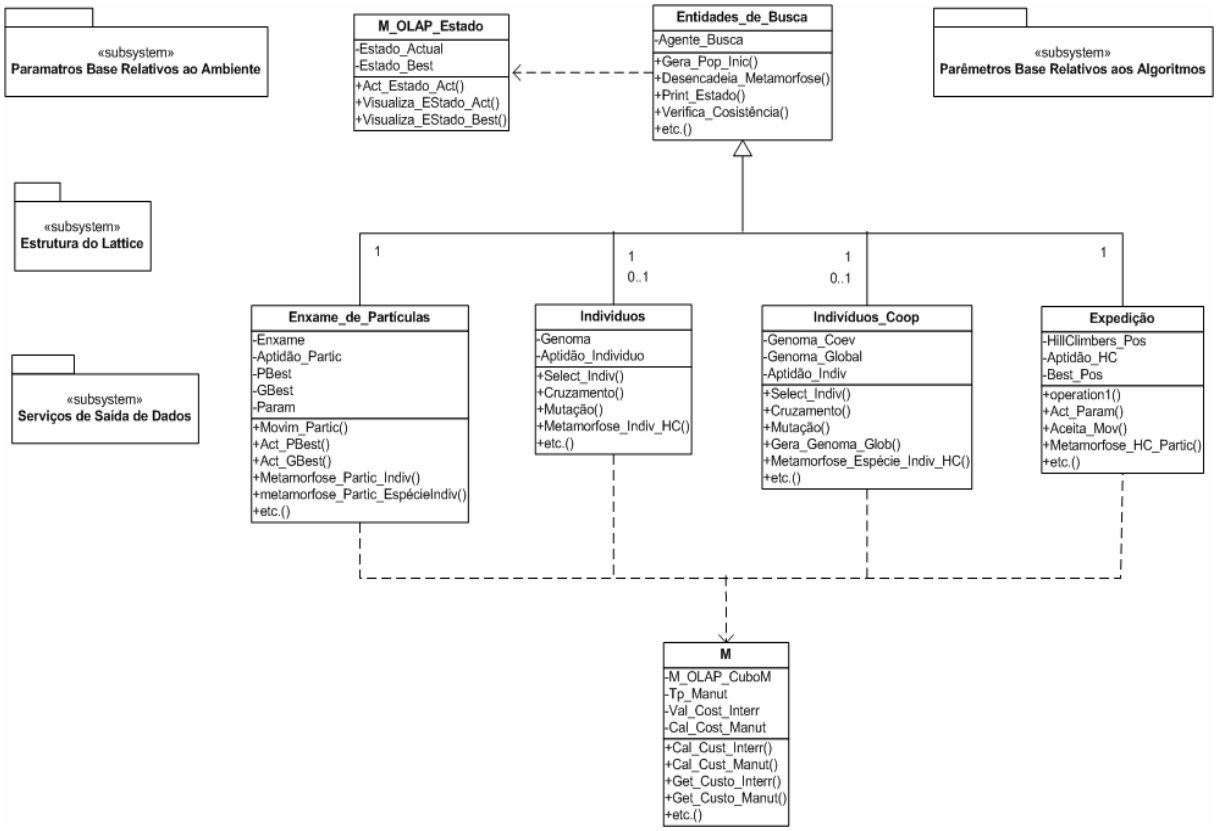


Figura 6.7. Arquitectura do Algoritmo MetaMorf M-OLAP.

Para os dois primeiros algoritmos (ODiEP e AG), e dadas as variantes possíveis já abordadas anteriormente, colocaram-se várias opções de selecção, nomeadamente:

- a versão padrão do algoritmo de optimização por enxame de partículas (ODiEP M-OLAP), descrita na secção 5.5.2, a classe Enxame_de_Partículas da Figura 6.7;
- as versões cooperativa e multi-fase do algoritmo de optimização por enxame de partículas (ODiEP-C M-OLAP e ODiEP-MF M-OLAP), descritas também na secção 5.5.2;
- uma versão padrão do algoritmo genético (AG M-OLAP), a classe Indivíduos da Figura 6.7;
- uma versão co-evolucionária (Co-Evol-AG M-OLAP) do algoritmo genético (ver secção 5.4.2), a classe Indivíduos_Coop da Figura 6.7;
- o algoritmo *hill climbing* com *simulated annealing* (HC-SA M-OLAP), a classe Expedição da Figura 6.7.

A versão padrão do algoritmo de enxame de partículas, dada a análise efectuada nas secções 5.6.4 e 5.8, foi a escolhida. Se a versão cooperativa do ODiEP (ODiEP-C M-OLAP) mostrou gerar soluções um pouco melhores do que a versão padrão, a relação tempo de execução X qualidade das soluções pende, decididamente, para a segunda. Por outro lado, recordando o desempenho desanimador do ODiEP-MF M-OLAP, este foi, simplesmente, preterido.

Quanto à inclusão da dupla roupagem genética, pesaram na decisão três ordens de razões:

1. a maior complexidade e consequente maior tempo de execução do Co-Evol-AG M-OLAP, faria pender a escolha para a versão standard;
2. a possível utilização de cada um dos algoritmos individualmente, e dada a superioridade manifestada pela versão co-evolucionária, implica a sua inclusão;
3. a existência das múltiplas espécies e a necessidade da existência do genoma global poderiam implicar a degradação da diversidade, além de tornar o processo de metamorfose muito mais complexo, mas, apesar disso, importava verificar a efectividade da versão co-evolucionária.

Finalmente, quanto ao *hill climbing* com temperamento simulado, é utilizada a versão descrita em 6.3.

Em resumo, quanto aos algoritmos integrantes, são utilizados aqueles que já foram concebidos e implementados anteriormente, apesar das alterações a incluir para acomodar o novo constrangimento, pelo que não se irá repetir a sua descrição nesta secção. Vai-se, sim, descrever a arquitectura do próprio algoritmo, pois que necessariamente complexa, dada a inclusão de quatro algoritmos e respectivas estruturas de controlo. Também o processo das metamorfoses, bastante elaborado, obrigou a muitas decisões de concepção.

A classe Entidades_de_Busca surge como uma super-classe, permitindo a múltipla forma da EB. Mas o estado da EB poderia incluir atributos capazes de acomodar as especificidades de cada uma das suas formas possíveis. Discutamos então, o porquê desta opção.

Relembremos o paradigma subjacente à codificação da solução e as respectivas estruturas de suporte ao estado da EB em cada uma das possíveis formas. Se numa partícula ou HC o mapeamento da solução assenta num paradigma espacial, sendo portanto bastante similares, já o mesmo não se passa com um indivíduo genético. Contudo, vistas as coisas à luz das próprias estruturas de dados, a semelhança é bastante maior: tem-se sempre uma cadeia binária, independentemente de simbolizar uma posição no espaço n-dimensional ou um genoma. Há outras estruturas de dados auxiliares, por exemplo, no caso do enxame de partículas, *pbest* e *gbest*, genomas globais no Co-Evol-AG, mas o

cerne, o código da própria solução, é o mesmo. Desta forma, e dada a semelhança, poder-se-ia optar por:

- ter um estado referente às entidades de busca, com a totalidade dos possíveis dados relativos ao estados de todos as formas possíveis da EB e dados relativos ao controlo do algoritmo;
- manter estados individualizados relativos a cada população de cada uma das formas da EB, acrescentando um novo estado, que suportaria a informação relativa ao mapeamento da EB para cada um dos elementos dos estados individualizados (não mais do que um apontador) e as estruturas de controlo do algoritmo metamorfose.

Pesados os prós e contras de cada uma das opções, achou-se a segunda opção a mais adequada, especialmente pelas seguintes razões:

1. a existência dos algoritmos integrantes já concebidos e desenvolvidos permitiria, com relativa facilidade, a sua reutilização integrando o novo algoritmo;
2. seria assegurado o isolamento de cada um dos algoritmos integrantes, já que o estado seria autónomo (relativo a cada um dos algoritmos);
3. permitir-se-ia, com relativa facilidade, a inclusão de novas heurísticas no algoritmo, ou seja, poder-se-ia ter novas formas da EB;
4. o novo estado a conceber seria relativamente simples e, mais uma vez, representaria o estado da EB na sua parcela global a qualquer forma assumida, ficando cada um dos outros estados individualizados como sendo uma especialização da EB;
5. a desfavor da opção estaria um aumento do tamanho das estruturas de dados (mas realmente limitado) e a necessidade de alguma movimentação de dados que acompanharia o processo de metamorfose, que é real, mas mais oneroso em termos do código necessário do que em custos de processamento associado.

Tendo em conta estes considerandos, descreve-se apenas a classe relativa ao estado das EBs, onde foi incluído suporte para informação relativa:

a) à forma actual de cada EB;
b) ao apontador da EB para o correspondente elemento de uma das populações (enxame, indivíduos ou expedição);
c) à melhor aptidão conseguida por cada uma das EBs;
d) ao número de iterações decorridas sem melhoria da aptidão de cada EB;

e) ao número total de cada um dos tipos de EB;
f) a estatísticas várias acerca das EBs.

Quanto às metamorfoses, e olhando mais uma vez a Figura 6.6, pode ver-se que serão necessárias três: 1) partícula → indivíduo, 2) indivíduo → HC e 3) HC → partícula; como há duas versões do AG, duas destas metamorfoses serão duplicadas, perfazendo um total de cinco. Destas, as mais complexas serão, à partida, as que envolvem a forma genética.

Em [Krink & Løvbjer, 2002] as metamorfoses são do tipo 1 ↔ 1, ou seja uma EB partícula transforma-se numa EB genética e esta num HC. A esta relação e no que concerne ao número de EBs que resultam de uma metamorfose denominaremos multiplicidade. Neste caso, estar-se-á perante uma relação de multiplicidade de grau igual a um. Mas, como já se viu anteriormente (secções 4.10.3 e 5.6.3, considerações tecidas à volta da Figura 4.16 e Figura 5.8, respectivamente), é necessária uma população bastante maior de indivíduos, relativamente, por exemplo, a partículas, já que a qualidade das soluções conseguidas é baixa para populações com um pequeno número de indivíduos. Aliás, em [Krink & Løvbjer, 2002], é já indicado, como trabalho futuro, a utilização de uma metamorfose 1 ↔ n (multiplicidade n) nas metamorfoses onde as EB genéticas participam. Esta realidade foi já considerada: permitiu-se a existência de uma população maior de indivíduos, o que motivou a já aludida maior complexidade das metamorfoses 1 e 2. Assim, e para o caso da versão standard do AG, concebeu-se o seguinte esquema:

- Para a metamorfose 1 (o método metamorfose_partic_indiv() - Figura 6.7), o primeiro indivíduo gerado fica com o genoma correspondente à posição actual da partícula metamorfoseada; o segundo indivíduo fica com um genoma correspondente à posição de PBest da partícula metamorfoseada; o terceiro indivíduo é gerado através do cruzamento dos genomas correspondentes à posição da partícula e de GBest; os restantes indivíduos vão ser gerados da forma utilizada ao gerar-se a população inicial de indivíduos.
- Já a metamorfose 2 (o método metamorfose_indiv_hc - Figura 6.7) tem de lidar com o processo inverso. A expansão populacional ocorrida na metamorfose 1 tem agora de transformar-se na redução correspondente e cada n indivíduos vão gerar um HC. O HC só pode ser gerado quando o enésimo indivíduo tiver desencadeado a sua metamorfose. Assim, concebeu-se uma área de quarentena, para onde os sucessivos indivíduos em metamorfose são transferidos. Esta área, quando atinge uma população n, irá gerar o HC, sendo então esvaziada. Como haverá n indivíduos que se transformarão num único HC, concebeu-se um mecanismo probabilístico para seleccionar aquele que se metamorfoseará efectivamente:

com uma probabilidade P, é seleccionado o mais apto, e com uma probabilidade (1 - P) é seleccionado um, aleatoriamente.

- Quanto à metamorfose 3 (o método metamorfose_hc_partic - Figura 6.7) é simples, pois que a posição do HC torna-se a localização da partícula e PBest, sendo a velocidade gerada de forma aleatória em cada dimensão, mas só a aceitando se ela puder originar a posição recebida do HC metamorfoseado, aplicando a regra 4.2 (secção 4.5.2).

Para a versão co-evolucionária do AG, as metamorfoses 1 e 2 são algo mais complexas, já que, além da gestão da multiplicidade, acresce que cada indivíduo contém apenas parte da solução (recorde-se que há múltiplas espécies), ou seja, cada partícula irá gerar n*d indivíduos, onde d é o número de dimensões. Acresce a complexidade adicional da necessidade da existência do genoma global utilizado para avaliar o indivíduo de cada espécie em termos da solução global. Dadas estas condicionantes, conceberam-se os esquemas a seguir apresentados:

- Para a metamorfose 1 (o método metamorfose_partic_espécieindiv - Figura 6.7), o número de dimensões da posição da partícula correspondente aos possíveis subcubos a materializar em cada NSO tornam-se no genoma do indivíduo de cada espécie, ou seja, por cada partícula a metamorfosear-se é gerado um indivíduo de cada espécie. A forma de gerar cada um dos n indivíduos em todas as espécies é idêntico ao utilizado para a versão standard do AG, mas deve atender-se agora à existência de espécies.
- Quanto à metamorfose 2 (o método metamorfose_espécieindiv_hc - Figura 6.7), vai ser desencadeada quando nenhum dos indivíduos na mesma posição, numa qualquer das espécies, apresentar melhoria da sua aptidão. Quando tal suceder, selecciona-se com um esquema probabilístico a espécie da qual será metamorfoseado o indivíduo. Para o restante genoma, é utilizado o genoma global. Assim, é transferido para a área de quarentena o genoma global, sendo substituído pelo genoma do indivíduo da espécie seleccionada. Quando a área de quarentena está cheia, gera-se o HC utilizando o mesmo esquema probabilístico descrito para a versão padrão do AG.

6.5.2 Teste Experimental Simulado

Este conjunto de testes utilizou o mesmo ambiente utilizado para realizar os testes descritos na secção 6.3.3, pelo que não será aqui descrito. Esta situação explica-se, pois que na realidade, os

testes realizados, então utilizaram já esta implementação, sendo por isso restrita a utilização dos algoritmos componentes ao HC-SA. Para os parâmetros de controlo do comportamento da EB, em cada uma das suas formas, foram seleccionados os valores encontrados como mais adequados nos testes efectuados aos respectivos algoritmos integrantes (secções 5.6.3, 5.6.4 e 6.3.3). Os restantes parâmetros vão ser especificados teste a teste, já que são aqueles cujo impacto no comportamento do algoritmo se pretende avaliar. São eles:

1. a forma inicial que assumirá a população de EBs;
2. o número de iterações que decorreram sem melhoria de aptidão que vai disparar a metamorfose (D);
3. a multiplicidade de conversão de número de EBs quando uma das formas intervenientes é uma EB-AG;
4. o tipo de selecção quando há conversões relacionadas com multiplicidade e respectiva probabilidade (se selecção tipo P);
5. o tipo de AG (padrão ou co-evolucionário).

O conjunto de testes concebido pretende avaliar, em termos de desempenho do algoritmo, o seguinte:

1. qual o tipo de EB inicial que é mais benéfico;
2. qual o impacto do número de entidades de busca na qualidade das soluções conseguidas;
3. qual o impacto da relação de multiplicidade nas metamorfoses onde EBs-AG intervêm;
4. qual o impacto de D;
5. a respectiva escalabilidade, no que respeita ao número de interrogações e ao número de nós da arquitectura M-OLAP.

Também aqui, e dada a natureza estocástica do algoritmo, cada teste é repetido 10 vezes, tomando-se a média para o valor observado nas execuções do teste. Um valor mais elevado seria estatisticamente mais significativo, mas, dada a morosidade da execução (devida à complexidade do algoritmo e à exiguidade dos recursos computacionais utilizados) e como o tempo disponível para a sua execução é escasso, optou-se por este valor como um compromisso entre a precisão obtida e tempo possível.

Para o primeiro teste utilizou-se um conjunto de 90 interrogações geradas aleatoriamente e a versão padrão do algoritmo genético. O algoritmo foi executado, sendo gerada a população inicial de EBs da primeira vez como EB-EP, da segunda como EB-AG e da terceira como EB-HC. Usou-se uma relação

de multiplicidade igual a 5 e permitiram-se 500 iterações. O resultado obtido, relativo à qualidade das soluções obtidas e respectivos tempos de execução mostram-se na Figura 6.8.

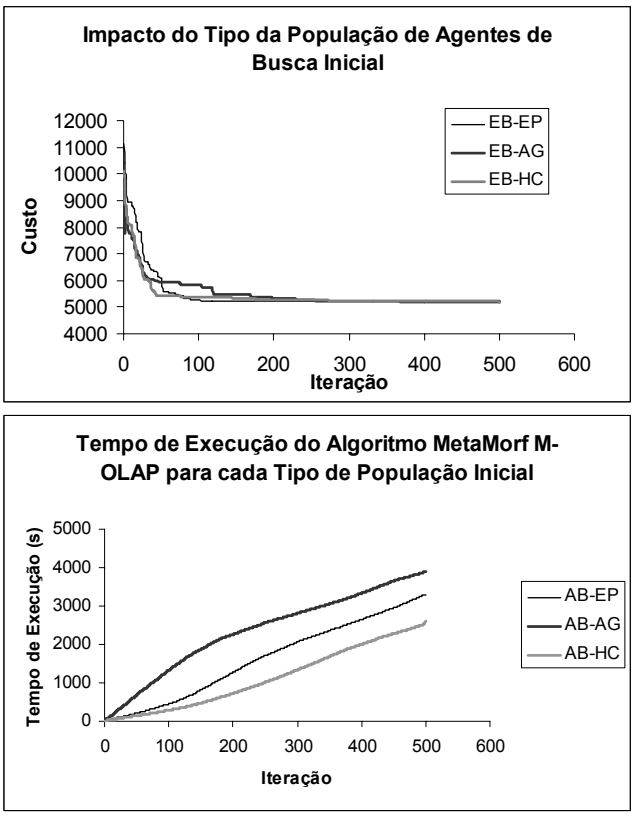


Figura 6.8. Impacto da forma inicial das entidades de busca no desempenho do MetaMorf M-OLAP.

Como se pode ver no primeiro dos gráficos, o tipo de população inicial tem um impacto reduzido na qualidade das soluções obtidas. No entanto, uma análise mais detalhada parece mostrar que a forma inicial mais benéfica para as entidades de busca é a *hill climber*, já que permite uma obtenção mais rápida de boas soluções. A adopção da forma inicial EB-HC é reforçada pelos tempos de execução evidenciados pelo MetaMorf M-OLAP mostrados na Figura 6.8 (segundo gráfico), claramente inferiores para populações iniciais EB-HC. Também é evidente, para o problema em questão, que 200 iterações serão suficientes para os testes, pois que as iterações ulteriores não trouxeram benefícios minimamente significativos.

No segundo teste pretendeu-se avaliar o impacto do número de entidades de busca no desempenho do algoritmo. Vai usar-se uma relação de multiplicidade de grau 5. São utilizadas populações

máximas de 50, 100 e 200 EB. O que significa que teremos, no máximo, um número igual de EBs-AG e um número cinco vezes inferior de EBs-EP e EBs-HC. Por exemplo, para 100 EBs, poder-se-á ter, numa população homogénea, (20 EBs-EP + 0 EB-AG + 0 EB-HC) ou (0 EB-EP + 100 EBs-AG + 0 EB-HC) ou (0 EBs-EP + 0 EB-AG + 20 EB-HC) ou ainda, para uma população heterogénea, (5 EBs-EP + 50 EBs-AG + 5 EBs-HC) ou (10 EBs-EP + 25 EBs-AG + 5 EBs-HC) ou (2 EBs-EP + 70 EBs-AG + 4 EBs-HC). Na prática, uma população de 100 EBs implica que a população inicial seja de 20 EBs-HC. A Figura 6.9 mostra os resultados obtidos no teste.

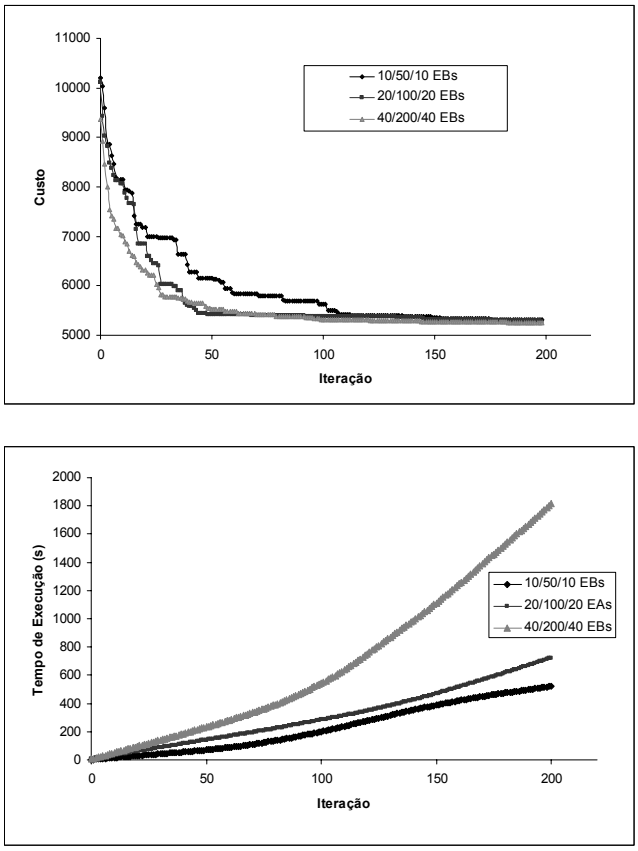


Figura 6.9. Impacto do número de entidades de busca no desempenho do MetaMorf M-OLAP.

Observando o primeiro gráfico, verifica-se que o número de entidades de busca tem um impacto relativamente reduzido na qualidade das soluções obtidas, embora uma maior população permita obter mais rapidamente soluções de qualidade. Este comportamento explica-se provavelmente devido ao impacto positivo que terão, neste particular, as formas EB-EP e EB-HC que, já se viu, são relativamente independentes, no que respeita à qualidade das soluções conseguidas, do tamanho da população. O segundo gráfico mostra que o tempo de execução cresce com a população, de forma

moderada (inferior à relação das populações) para populações moderadas (entre 10/50/10 e 20/100/20) e de forma acentuada para a população de 40/200/40. Este comportamento deve-se, provavelmente, ao facto de, para populações elevadas, os EBs-AG se revelarem mais eficazes e assim, predominarem na população, implicando tempos de execução mais elevados, pois que o algoritmo genético, além de ser de processamento mais complexo, utiliza, neste caso, um maior número de EBs. De qualquer forma, a relação qualidade x população parece pender para populações pequenas, visto que o aumento de tempo de execução não é pago numa melhoria sensível da qualidade da soluções obtidas. Populações de 20/100/20 parecem ser um bom compromisso a utilizar nos testes restantes.

O terceiro teste pretendeu avaliar o impacto do grau de multiplicidade e de D (o número de iterações sem melhoria de desempenho do EB que vai desencadear a metamorfose) no desempenho do algoritmo metamorfose M-OLAP. Vão utilizar-se os parâmetros usados no algoritmo anterior e uma população máxima inicial de 20 EBs-HC. O grau de multiplicidade vai ser de 1, 2, 5 e para D vão ser utilizados os valores 15, 30 e 60. Os testes efectuados deram origem aos gráficos apresentados na Figura 6.10.

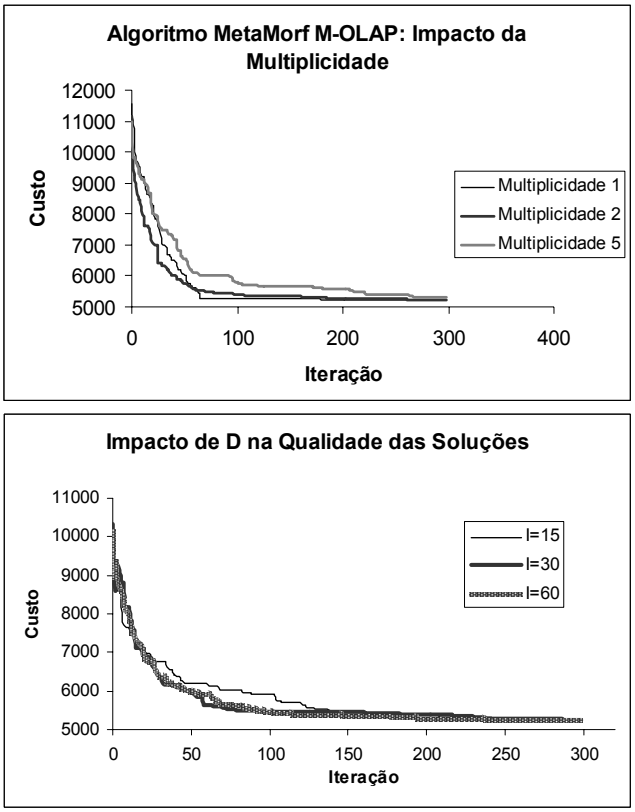


Figura 6.10. Impacto do grau de multiplicidade e de D no desempenho do MetaMorf M-OLAP.

Como se pode verificar, o algoritmo parece depender em pequena escala do grau de multiplicidade e valor do disparo da metamorfose. No entanto, uma análise mais pormenorizada parece indicar que um valor de multiplicidade = 2 e de D=30 ou 60 serão os mais adequados.

No último teste executado pretendeu-se perceber a escalabilidade do algoritmo MetaMorf M-OLAP, relativamente ao número de interrogações colocadas e ao número de nós da arquitectura M-OLAP. Para tal utilizaram-se três conjuntos de interrogações (com 30, 60 e 90 interrogações, respectivamente) e duas arquitecturas M-OLAP: a que tem vindo a ser utilizada e uma outra, com 6 NSOs (mais nó base), tendo sido aplicado, a ambas, um conjunto de interrogações com o mesmo número de interrogações. A análise do gráfico da Figura 6.11 permite verificar que o tempo de execução do algoritmo MetaMorf M-OLAP tem uma dependência pequena do número de interrogações colocadas (observou-se um crescimento de 16% quando o número de interrogações aumentou 3 vezes).

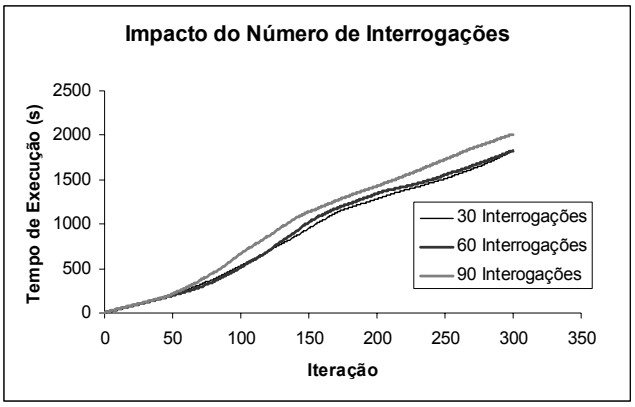


Figura 6.11. Escalabilidade do algoritmo relativamente ao número de interrogações.

Quanto ao impacto do número de nós da arquitectura M-OLAP no tempo de execução do algoritmo, o gráfico da Figura 6.12 mostra que ele é também bastante reduzido. Um crescimento de 2 vezes no número de nós da arquitectura implicou um aumento do tempo de execução de 20%. As duas evidências mostram que o algoritmo MetaMorf M-OLAP é bastante escalável quanto ao número de interrogações colocadas e número de nós da arquitectura M-OLAP.

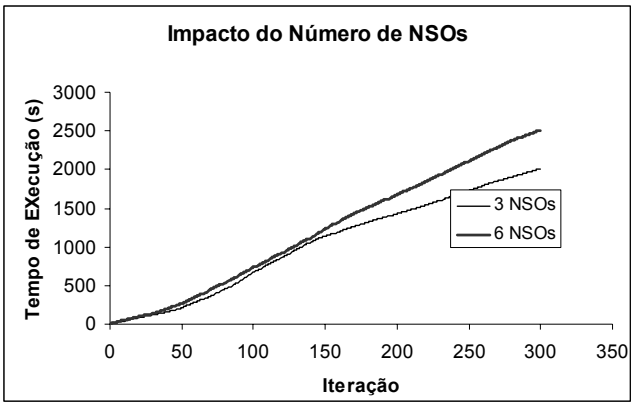


Figura 6.12. Escalabilidade do algoritmo MetaMorf M-OLAP.

6.6 A Restruturação do Cubo: Algoritmo Hill-Climbing com *Simulated Annealing*

O algoritmo apresentado na secção 6.3 pode ser utilizado para empreender a restruturação do cubo, numa perspectiva de ajuste dinâmico. Os custos de manutenção, então considerados, podiam ser incrementais ou integrais, mas seria natural considerá-los incrementalmente. Supunha-se então que, ao empreender-se a restruturação do cubo, seriam considerados os custos de manutenção incremental (em termos de optimização do cubo), já que aqueles a incorrer no dia-a-dia de operação dos sistemas de processamento analítico, sendo os custos de materialização do novo cubo (ou da sua reconfiguração) considerados exteriores ao processo normal de refrescamento das estruturas multidimensionais (até porque a decorrer a intervalos bastante mais alargados). Agora vai incluir-se a reconfiguração do cubo (com a geração das necessárias estruturas multidimensionais) no próprio processo de refrescamento do cubo. Esta reconfiguração será de natureza limitada, mas poderá decorrer a intervalos mais curtos.

A definição anterior de vizinhança (definida na secção 6.3.1) é mantida. Mas o esquema de movimentação é alterado, sendo realizado em duas fases. Na primeira fase, cada hill climber vai mover-se para uma nova posição que corresponde à desmaterialização de subcubos anteriormente materializados (na distribuição *M* relativa à solução corrente, a manter incrementalmente). Este movimento é seguido de um conjunto de outros movimentos (repetidos até que uma condição seja satisfeita) que correspondem a rematerializações de subcubos a manter integralmente (já que têm de

ser recriados). Estes movimentos constituem a segunda fase do novo esquema de movimentação dos *hill climbers*. Vejamos a possível aplicação.

Suponha-se uma distribuição actual *M* de subcubos, optimizada para um perfil de interrogações $I = \{i_i, ..., i_n\}$ que, depois de actualizada, ficou disponível para resposta a interrogações. Decorrido um período *tp*, é gerado um novo perfil de interrogações $I' = \{i_i, ..., i_n\} \neq I$, em resultado das interrogações colocadas. Eventualmente, alguns dos subcubos existentes em *M* serão agora de pouca valia (tornaram-se obsoletos), enquanto que outros (inexistentes em *M*), seriam benéficos. Assim, importa refrescar *M* (num período máximo *tm*), mas tentar, simultaneamente, a sua afinação, em resultado da alteração do perfil de interrogações agregadas colocadas. Esta afinação assumirá a forma da eventual eliminação de alguns subcubos e a geração de outros (aqueles que se revelarem mais adequados em termos da minimização do tempo de resposta ao novo perfil de interrogações), gerando-se uma nova distribuição M', que ficará disponível para consultas e actualizações, até que se proceda a nova afinação. Em termos formais, o problema pode definir-se assim:

Dado $M = \{s_1, s_2, ..., s_n\}$ optimizado para resposta às interrogações $I = \{i_i, ..., i_n\}$, importa gerar $M' = \{s_1, s_2, ..., s_n\} \,||\, M' = M \setminus M_d + M_r$, onde $M_d = \{s_{d1}, s_{d2}, ..., s_{dn}\}$ é o conjunto de subcubos a desmaterializar e $M_r = \{s_{r1}, s_{r2}, ..., s_{rn}\}$ o conjunto de subcubos a rematerializar, de forma a minimizar o custo de resposta às interrogações $I' = \{i_i, ..., i_n\} \neq I$, atendendo a um constrangimento espacial e temporal de manutenção, sendo $M \setminus M_d$ objecto de manutenção incremental e M_r gerado de novo.

Desta forma, em intervalos tão curtos quanto se pretenda, pode-se empreender uma afinação dos subcubos materializados nos vários NSOs da arquitectura M-OLAP de forma a acompanhar, mais de perto, a alteração do perfil de utilização. Esta actuação pode entender-se como uma restruturação dinâmica das estruturas multidimensionais, daí a classificação transversal estática e dinâmica da Figura 2.13 (secção 2.8).

Como se percebeu do esquema de movimentação atrás descrito, na prática, o esquema de procura de soluções é empreendido a dois níveis: no primeiro é realizada a eliminação de alguns subcubos, seguindo-se um processo iterativo de rematerialização de outros, procurando-se durante um número de iterações a melhor solução. Nesta abordagem, a actuação é efectuada ao nível de um qualquer NSO e subcubo. Tem-se uma arquitectura M-OLAP com n NSOs e vão eliminar-se subcubos em

quaisquer NSOs e gerar subcubos também em quaisquer NSOs. Neste caso, está a supor-se que os NSOs estão situados num mesmo nível hierárquico arquitectural.

Outra solução é possível, aliás, decorrente da actuação intra-nível enunciada no 2.º parágrafo da secção 2.9, implementando a diferenciação dos NSOs da arquitectura M-OLAP. Passarão a ter-se NSOs de características predominantemente estáticas (onde residem as estruturas multidimensionais de maiores dimensões) e outros, bastante mais pequenos, objecto de manutenção integral, cujo conteúdo procurará seguir de perto a evolução das necessidades. Estes podem residir em localizações remotas, procurando assim minimizar custos de comunicação quando interrogações são colocadas em escritórios localizados em outros locais, que, dada a globalização das organizações e consequente mobilidade dos decisores e tomada de decisões locais, se tornam cada vez mais comuns, interrogações essas que incluem considerações de ordem global. Para o processo de selecção e alocação de cubos poderá utilizar um dos algoritmos descritos anteriormente, bastando que o processo de pesquisa de soluções seja limitado aos nós dinâmicos.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Avaliação Crítica

Este trabalho foca a sua acção na optimização das estruturas multidimensionais cuja materialização é condição de desempenho em sistemas de processamento analítico, tendente à satisfação dos decisores, seus clientes. Aliás, o acrónimo da sua designação, OLAP, tal como foi proposto por E. F. Codd, inclui o epíteto "*on line*". Muitas outras características são apanágio destes sistemas, derivadas, de forma imediata, das necessidades dos seus utilizadores e da forma nativa como vêem o negócio. Tais necessidades acabaram por determinar a forma conceptual (e física) dos repositórios de dados, soluções arquitecturais e, no que respeita ao desempenho, impuseram um enorme stresse sobre as plataformas de processamento analítico. Mas a própria natureza das interrogações colocadas e o seu perfil de frequência permitiram perseguir uma solução, sob a forma da materialização das respostas às próprias consultas, na sua forma agregada, constituindo as denominadas vistas materializadas ou cubos. Porém, a complexidade do modelo dimensional da maioria dos DW determina a impossibilidade da materialização global do cubo, obrigando à eficaz selecção das agregações mais benéficas. Este problema, conhecido como selecção de vistas a materializar, constitui um dos focos primários da investigação no domínio dos sistemas de processamento analítico. Propostas de modelos de custos e algoritmos de estimativa de custos e a utilização de várias heurísticas de optimização foram e são partes da sua solução. Mas outros vectores de investigação se têm perfilado como opções a utilizar para a solução do mesmo problema, criando uma rede de interacções tendente à prossecução do objectivo em causa, a saber: a distribuição das próprias agregações e a temporalidade da sua recalibração, em resultado da variabilidade do perfil das interrogações. Esta triplicidade dos vectores-

solução relativos ao desempenho dos sistemas de processamento analítico foi resumida na Figura 1.1, sendo muitos deles materializados em propostas de solução mostradas na figura do anexo 2, enquadradas nos trabalhos existentes. Se foram utilizadas então como directoras deste projecto de investigação, vão agora ser usadas para mostrar em que medida os propósitos foram cumpridos, transpondo os trabalhos desenvolvidos para o esquema da Figura 1.1, originando a Figura 7.1.

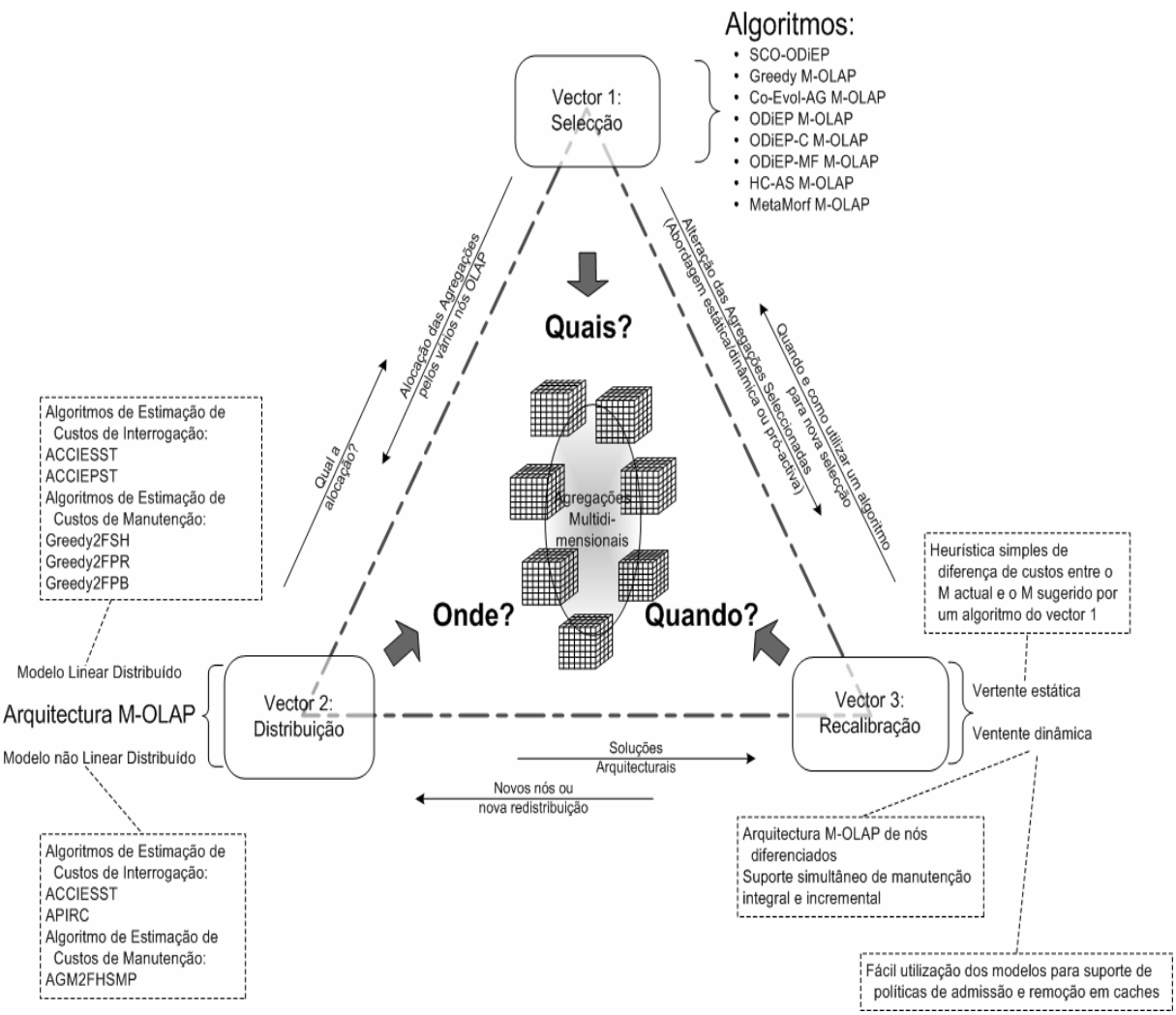


Figura 7.1. Enquadramento do trabalho desenvolvido nos três vectores de investigação definidos na secção 1.2.

Começando pelo vector 2, já que os modelos e algoritmos de estimativa de custos vão ser utilizados como função de aptidão ou de cálculo de benefício pelos algoritmos desenvolvidos no âmbito do vector 1, foi proposta e definida uma arquitectura OLAP distribuída, a arquitectura M-OLAP, para a qual foram concebidos dois modelos de custos: sendo um linear e outro não linear. O primeiro baseou-se no modelo de custos linear [Harinarayan et al., 1996] e *lattice* de agregação distribuído [Bauer & Lehner, 2003], tendo sido introduzido um suporte explícito para a capacidade de processamento de cada nó e parâmetros relativos ao cálculo de custos de comunicação. Este modelo permite assim a utilização de uma rede de comunicação heterogénea e a existência de nós diferenciados e o suporte ao cálculo de custos de interrogação e manutenção (ainda que com algumas limitações), sendo o custo calculado em unidades temporais. Esta característica é bastante interessante, já que é uma medida que tem reflexo imediato: a) na satisfação dos utilizadores (relacionada com o tempo de resposta às interrogações) e, não menos importante, b) na verificação das possíveis restrições temporais de manutenção.

Baseado neste modelo foram desenvolvidos dois algoritmos de estimativa de custos de interrogação e três de estimativa de custos de manutenção, que geravam também um plano de manutenção. A arquitectura M-OLAP, inerentemente paralela, pode ser usada de forma a que o processamento possa decorrer em paralelo nos vários NSOs. Assim, para a estimativa de custos de interrogação, foi proposto um algoritmo de simulação sequencial das tarefas (ACCIESST - algoritmo de cálculo de custos de interrogação com execução sequencial simulada de tarefas) e outro que utiliza o paralelismo (ACCIEPST – algoritmo de cálculo de custos de interrogação com execução paralela simulada de tarefas). A razão para o primeiro algoritmo, de simulação sequencial de tarefas (o que é um aparente contra-senso, já que se dispõe de uma arquitectura inerentemente paralela), é apresentada na secção seguinte, pelo que não será aqui repetida. Já os algoritmos de cálculo de custos de manutenção adoptam todos o processamento simulado paralelo de tarefas, tendo no entanto abordagens diferenciadas na heurística que vai determinar a ordem de actualização dos subcubos. Surgiram assim os algoritmos Greedy2FSH, Greedy2FPR e Greedy2FPB [Loureiro & Belo, 2006g]. Os testes efectuados mostraram que os dois primeiros são os mais adequados, por serem mais rápidos, com ligeira vantagem para o primeiro. Também o plano de manutenção gerado pelos dois primeiros mostrou uma clara vantagem face ao plano gerado pelo último, o que constituiu uma surpresa em face da heurística muito mais elaborada empregue no Greedy2FPB.

O segundo modelo de custos, o não linear, procurou responder à limitação detectada no anterior, respeitante ao suporte ao cálculo de custos de manutenção incremental. Este modelo constitui um refinamento do anterior, introduzindo não linearidades e suporte explícito de custos de manutenção

incremental. O *lattice* vai incluir pesos relativos a custos não só nos nós, mas também nas ligações, o que permite a diferenciação de custos dependendo do "caminho" de cálculo (as dependências intra-nó percorridas entre o subcubo fonte e o subcubo a gerar). Com base no modelo, foram concebidos dois algoritmos de estimativa de custos de interrogação, um sequencial e outro paralelo, o ACCIESST e o APIRC (algoritmo de execução paralela de interrogações em pipelining com alocação de nós por reordenação em janela constrangida) [Loureiro & Belo, 2006d], e um algoritmo de estimativa de custos de manutenção, o AGM2FHSMP (algoritmo *greedy* de estimativa de custos de manutenção em duas fases em sequência hierárquica com simulação de processamento paralelo de tarefas em *multipipelining*) [Loureiro & Belo, 2006h]. A simulação de paralelismo foi aqui derivada das técnicas de *pipelining*, usadas amiúde para descrever e compreender a operação dos processadores [Hennessy & Patterson, 2002]. Este algoritmo de estimativa de custos de manutenção continua a gerar o plano de manutenção e, mais importante, permite o suporte diferenciado e simultâneo de custos de manutenção integral e incremental, possibilitando assim que seja especificado o tipo de manutenção a efectuar ao nível do subcubo. Este modelo de custos é bastante mais complexo do que o correspondente modelo linear, podendo assim responder melhor a factores de heterogeneidade, típicos da arquitectura M-OLAP. Esta complexidade estende-se aos algoritmos APIRC e AGM2FHSMP, mais ainda pela sobrecarga advinda da gestão do lançamento das interrogações ou tarefas de manutenção em cada *pipeline*. De qualquer forma, a utilização de técnicas de programação dinâmica permitiram algum alívio para este problema, e os algoritmos mostraram um tempo de execução maior do que os correspondentes do modelo linear, mas, ainda assim, perfeitamente utilizáveis, mesmo em situações de uso intensivo.

Uma última nota quanto aos modelos e algoritmos: ainda que fundamentados em modelos propostos na literatura, poderiam ter beneficiado de uma avaliação experimental. Esta foi, aliás, equacionada, mas o conjunto de componentes requeridos para implementar o sistema de testes que seria necessário desenvolver, implicaria um tempo que inviabilizaria o trabalho de investigação correspondente ao vector 1, ao qual foi decidido dar uma maior relevância, já que a afinação dos modelos poderia ser feita, de forma simples, *a posteriori*, porque implicará, possivelmente, apenas a alteração das próprias fórmulas de custo. Esta avaliação é, no entanto, considerada muito relevante, especialmente na implementação prática de um sistema de *data warehousing* com inclusão de M-OLAP, pelo que é proposta para trabalho futuro.

No vector 1, foram propostas duas novas heurísticas: optimização discreta por enxame de partículas (ODiEP) e algoritmo metamorfose (MetaMorf). Variantes de várias heurísticas de optimização foram

também incluídas: *hill climbing* com *simulated annealing*, algoritmo genético coevolucionário, ODiEP cooperativo, ODiEP multi-fase com extinção em massa e ODiEP com hibridações genéticas.

A primeira heurística revelou um melhor desempenho quando aplicada ao problema da selecção de cubos em OLAP centralizado quando comparada com heurísticas *greedy* [Loureiro & Belo, 2006c], conseguindo também uma qualidade de soluções análoga à obtida pelos algoritmos genéticos padrão, mas conseguida em tempos de execução bastante inferiores [Loureiro & Belo, 2006a]. A ODiEP foi também aplicada à selecção de cubos para arquitecturas M-OLAP, tendo sido investigado o desempenho relativo a duas variantes e à inclusão de hibridação genética. Os resultados obtidos no teste experimental [Loureiro & Belo, 2006i] mostraram que a ODiEP é também uma boa solução de optimização para a selecção de cubos em arquitecturas M-OLAP. No entanto, a versão cooperativa e, especialmente, a versão multi-fase, não exibiram, pelo menos nas condições dos testes realizados, os ganhos de desempenho que os resultados experimentais reportados na literatura fariam esperar e que, de algum modo, os induziram como opção a considerar. Só a versão cooperativa mostrou conseguir melhores soluções do que a versão padrão da ODiEP. Todavia, além da melhoria obtida não ser substancial, implicou um crescimento muito elevado no tempo de execução.

Uma outra característica muito relevante, especialmente se o alvo for a arquitectura M-OLAP, é a escalabilidade, já que, além da complexidade do próprio cubo, há que lidar agora com um número de nós potencialmente elevado. Os testes realizados mostraram que o algoritmo ODiEP apresentou uma escalabilidade boa no que concerne à complexidade do próprio cubo, pois que para um crescimento do número de subcubos de 4 vezes, evidenciou um aumento de tempo de execução de cerca de 3 vezes. Comparativamente, e nas mesmas condições de teste, o algoritmo genético padrão revelou um crescimento do tempo de execução de, aproximadamente, 7 vezes. Quanto à capacidade de lidar com arquitecturas M-OLAP complexas, o algoritmo ODiEP M-OLAP exibiu um crescimento quase linear do tempo de processamento com o número de NSOs. Em resumo, a escalabilidade do algoritmo ODiEP pode considerar-se assegurada.

A segunda heurística, o algoritmo metamorfose [Loureiro & Belo, 2006-rb], constituiu uma oportunidade de introduzir uma nova proposta para a resolução do problema da selecção e alocação de cubos em M-OLAP, incluindo agora uma característica desejável da auto-adaptabilidade relativamente às heurísticas mais apropriadas ao problema particular em resolução e, simultaneamente, a realização de um componente de optimização mais vasto, que incluiu a maioria das propostas algorítmicas concebidas e desenvolvidas. A forma como foi concebido permite até a fácil inclusão de novas heurísticas que podem ser utilizadas depois de forma isolada, ou mesmo

inseridas no próprio algoritmo metamorfose. Aliás, uma nova heurística foi desenvolvida para integrar de imediato o algoritmo: *hill climbing* com *simulated annealing* (HC-SA), que se passa a avaliar.

Se o algoritmo já tinha sido proposto em ambientes DW centralizados [Kalnis et al., 2002b], onde evidenciou conseguir boas soluções e uma muito boa escalabilidade, aqui foi alargada a sua abrangência espacial, migrando para uma arquitectura M-OLAP. Mas o próprio mecanismo de procura de soluções de vizinhança foi também modificado, incluindo vários esquemas adaptativos, que procuraram optimizar o processo de pesquisa de soluções para problemas mais complexos. Os testes levados a cabo mostraram que o algoritmo consegue boas soluções, mesmo com um número pequeno de HCs (compreensível, já que não há qualquer mecanismo de interacção entre HCs), assegurando um tempo de execução reduzido (à semelhança também do comportamento da ODiEP). A escalabilidade é mesmo a sua característica digna de nota, pois que o HC-SA M-OLAP mostrou uma quase independência do tempo de execução relativamente ao número de interrogações colocadas, e quanto ao número de NSOs da arquitectura M-OLAP, revelou também um comportamento de relevo (é mesmo a melhor de todas as heurísticas testadas, no que respeita a esta característica), já que um crescimento para o dobro do número de NSOs implicou um aumento de tempo de execução de apenas 10%. Em resumo, o HC-SA M-OLAP mostra-se um bom candidato, especialmente em situações de elevada complexidade dimensional e número de NSOs elevado.

Já o algoritmo metamorfose (MetaMorf M-OLAP) também se revelou um bom candidato. Ainda que carecendo de uma investigação mais aprofundada, especialmente no que respeita aos mecanismos de implementação das metamorfoses e comunicação inter-populações, já mostra um bom desempenho, ainda que não atingindo o referido em [Krink & Løvbjer, 2002]. Se mostrou conseguir soluções de qualidade superior ao HC-SA M-OLAP e AG M-OLAP, não suplantando apenas o ODiEP M-OLAP, conseguiu, contudo, soluções aproximadas. No seu actual estado, a sua maior virtude parece ser a inerente adaptabilidade. No entanto, algumas evidências quanto à inclusão de esquemas de diversificação na metamorfose parecem mostrar que o algoritmo encerra ainda uma grande capacidade de evolução.

Para terminar a análise a este vector de investigação, falta apenas tecer algumas considerações acerca da variante co-evolucionária do AG que, aplicada ao problema da selecção e alocação de cubos para arquitecturas M-OLAP, fez conceber e implementar o algoritmo Co-Evol-AG M-OLAP. Neste algoritmo não se tem uma população de indivíduos, mas várias, cada uma denominada espécie. O epíteto co-evolucionário advém do facto das espécies cooperarem entre si na obtenção da solução global, já que cada indivíduo contém em si apenas a solução para uma parte do problema, neste caso,

os subcubos a materializar em cada NSO. Esta situação torna necessária a existência de um genoma global (correspondente a um *M* global) para permitir a avaliação de cada indivíduo. O algoritmo original foi alterado na parte relativa à geração do genoma global, introduzindo-se um mecanismo probabilístico na selecção dos indivíduos de cada espécie cujo genoma formará o genoma global. Na prática, introduz alguma variabilidade, não em cada população, mas na própria matriz utilizada para avaliação das soluções. Se a variante cooperativa do algoritmo ODiEP mostrou ser apenas marginalmente vantajosa, esta, relativa ao algoritmo genético, revelou-se bastante interessante, já que nos testes realizados conseguiu soluções finais de qualidade 33% e 44% superiores ao algoritmo genético padrão e ao algoritmo *greedy*, conforme os valores obtidos nos testes experimentais descritos na secção 5.6.3, e que se podem observar pela análise da Tabela 7.1.

Tabela 7.1. Custo das soluções propostas pelos algoritmos *greedy* e genéticos

	Greedy M-OLAP	AG M-OLAP	Co-Evol-AG M-OLAP
Conjunto Interr. A	7790.0	7152.0	5614.6
Conjunto Interr. B	13832.0	12221.0	9229.7
Conjunto Interr. C	8105.0	8030.0	5757.7
Média	9909.0	9134.3	6867.3
Ganho Co-Evol-AG X Greedy			44%
Ganho Co-Evol-AG X AG padrão			33%

No entanto, esta superioridade clara na qualidade das soluções propostas é paga em tempo de execução, pois que a heurística implica a necessidade de um maior número de avaliações da aptidão, num valor proporcional ao número de espécies. Recorde-se ainda que o algoritmo genético padrão é já de execução mais demorada do que o correspondente algoritmo ODiEP e percebe-se o custo elevado em tempo de execução que é pago pela melhoria conseguida pelo Co-Evol-AG M-OLAP. A escalabilidade do algoritmo é talvez o seu calcanhar de Aquiles: para um aumento de complexidade de 4 vezes do cubo utilizado (cubo A para cubo B) revelou um crescimento no tempo de execução de 8,7 vezes (conforme pode observar-se na Tabela 5.2); já quanto ao número de nós da arquitectura M-OLAP, observou-se um comportamento ainda mais penalizador, pois que para um crescimento de 3,5 vezes no número de NSOs, registou-se um aumento no tempo de execução de cerca de 25 vezes (conforme valores da Tabela 5.4), um crescimento claramente exponencial. Acrescente-se, no entanto, que, em ambos os casos, a qualidade da solução conseguida pelo Co-Evol-AG foi superior à do AG padrão. Será também de salientar o facto de o algoritmo Co-Evol-AG M-OLAP parecer ter ainda uma enorme reserva de poder na busca da qualidade: na verdade, ao permitir-se um maior número

de iterações e utilizando populações maiores, o algoritmo conseguiu soluções de qualidade ainda melhores. Em resumo, o algoritmo parece ser de eleger sempre que for pretendida uma qualidade superior da solução de materialização, requerendo, no entanto, recursos computacionais importantes.

Quanto ao vector 3, há a dizer que a maioria do trabalho efectuado vai permitir a reestruturação das estruturas multidimensionais, mas empregando uma abordagem estática. No entanto, os modelos de custos podem ser utilizados para avaliar o benefício do armazenamento da resposta a uma dada interrogação, especialmente se a arquitectura M-OLAP for vista numa perspectiva espacial dispersa, onde cada NSO poderá ser visto como um servidor *proxy* ou mesmo residir no próprio cliente. Os modelos poderiam ser utilizados para implementar a política de admissão e remoção nas caches, implementando um serviço que poderia residir em cada um dos nós ou ser centralizado. Esta via de trabalho não foi realmente seguida, visto que, além de já estarem disponíveis diversas arquitecturas que a implementam, não deveria implicar um grande esforço de investigação, e faria sentido quando uma implementação prática do sistema de DW com M-OLAP estivesse em operação, que, como já foi discutido, implicava um tempo de desenvolvimento que extravasava o âmbito temporal deste trabalho.

Porventura em direcção à reestruturação dinâmica incluiu-se já o duplo suporte de manutenção integral e incremental, o que permitiu já descrever os algoritmos e condições arquitecturais que possibilitarão uma recalibração mais fina das estruturas multidimensionais. Também a arquitectura M-OLAP, ao permitir a existência de nós estáticos e dinâmicos e a estratégia de suporte à manutenção (incremental ou integral, diferenciada ao nível do subcubo) do algoritmo AGM2FHSMP, pode permitir a utilização de alguns dos algoritmos disponibilizados pelo vector 1. A implementação prática é proposta como trabalho futuro (secção 7.3), sendo incluída na sua fase 1. A sua viabilidade é assegurada também pelo suporte já incluído nos algoritmos do duplo constrangimento espacial e temporal.

7.2 As Contribuições da Dissertação

A optimização em problemas de carácter combinatório é baseada numa medida de avaliação comparativa de soluções, pois que a função objectivo consiste num custo (ou proveito) que importa minimizar (ou maximizar). Este trabalho, claramente focado na optimização de estruturas multidimensionais de dados, apresenta assim duas grandes vertentes:

1. os algoritmos de optimização propriamente ditos (apresentados, nesta dissertação, nos capítulos 4, 5 e 6) e
2. os modelos e algoritmos de estimativa de custos (de interrogação e manutenção), necessários à avaliação das soluções sucessivamente propostas (apresentados no capítulo 3).

Por outro lado, acrescente-se que o âmbito espacial de aplicação das soluções de optimização foi sendo sucessivamente estendido e também a abrangência dos próprios modelos (suporte a novas características) foi sendo igualmente alargada. Desta forma, as publicações foram reflectindo o trabalho realizado, acompanhando o seu desenvolvimento temporal. Esta dissertação, de algum modo, reflectiu também essa abordagem, já que, naturalmente, a organização temporal acompanhou a evolução na complexidade dos próprios modelos e algoritmos.

Assim, o trabalho iniciou-se com a aplicação da optimização discreta por enxame de partículas (ODiEP) ao problema de selecção de cubos em OLAP centralizado. Uma população de partículas voga num espaço n-dimensional mapeado num espaço de soluções. A cada posição no espaço corresponderá uma determinada elegância. Assim, uma partícula que ocupe essa posição constituirá uma solução com essa elegância. Como se pretendia a minimização dos custos de interrogação e manutenção, a função de elegância era então o somatório destes custos. Aplicava-se um constrangimento espacial ao processo de optimização. Assim, era necessário o cálculo dos custos de interrogação e manutenção, o que implicou o desenvolvimento dos algoritmos respectivos, adoptando o modelo de custos linear em ambiente OLAP centralizado. Se o primeiro cálculo não é mais do que a transposição directa da fórmula de cálculo de custos de interrogação, já o segundo é mais complexo, pois que é preciso considerar a possível utilização das agregações entretanto já refrescadas. Além disto, foi concebido um conjunto de serviços adicionais que possibilitavam a criação de um ambiente de simulação para permitir o teste experimental dos algoritmos, desenhado de forma a poder suportar facilmente testes de outros algoritmos neste domínio, útil em fases posteriores deste trabalho.

A descrição da aplicação da ODiEP ao problema da selecção de cubos em OLAP centralizado e os resultados da sua avaliação experimental simulada comparativamente com um algoritmo *greedy* foram publicados nas actas da *8th International Conference on Enterprise Information Systems (ICEIS2006)* [Loureiro & Belo, 2006c]. Uma aplicação de algoritmos baseados na vida (o mesmo algoritmo ODiEP e um algoritmo genético normal), na sua aplicação ao problema anterior, e resultados experimentais comparativos, foram publicados nas actas da *4th WSEAS International Conference on Information Security, Communications and Computers (ISCOCO 2006)* [Loureiro &

Belo, 2005], sendo uma versão alargada do mesmo artigo seleccionada para publicação na revista *WSEAS Transactions on Computers, Issue 1, Volume 5,* Janeiro de 2006 [Loureiro & Belo, 2006a]. Estes trabalhos são reflectidos, como se percebe da estrutura mostrada na secção anterior, nos capítulos 3 e 4 desta tese.

Posteriormente, alargou-se o âmbito da arquitectura do sistema OLAP a um modelo distribuído. Se a justificação para esta abordagem já foi referida atrás (secção 1.3) em termos das suas possíveis vantagens, as suas desvantagens constituíram, paradoxalmente, o mote para a continuação do trabalho: permitir a escalabilidade a custos mais baixos e limitar os estrangulamentos, mas conceber os modelos e criar as ferramentas necessárias ao controlo do aumento da complexidade e consequente crescimento do esforço de gestão (a outra face da moeda). Um novo modelo é introduzido, baseado no *lattice* de agregação distribuída [Bauer & Lehner, 2003], ainda usando um modelo linear de custos, mas estendido pela inclusão de suporte explícito de custos de comunicação (através da especificação de parâmetros da rede capaz da interligação dos diversos nós) e também pela inclusão do parâmetro adicional: a capacidade de processamento dos vários servidores OLAP constituintes. A introdução deste parâmetro na equação de custos permitiu o necessário suporte à diferenciação dos nós (já que o agregado OLAP será, em regra, heterogéneo) e, por outro lado, possibilitou a transposição da métrica de custos, passando da unidade registos (ou células) para unidades temporais. Na verdade, num sistema de processamento analítico, o tempo surge associado à satisfação do utilizador de uma forma directa e indirecta: por um lado, ela é directamente dependente do tempo de resposta a interrogações (um custo de interrogações medido em unidades temporais mostra um valor ligado ao mundo real); por outro lado, e de forma indirecta, o tempo, surgindo como grandeza mensurável relativa ao esforço de refrescamento das agregações materializadas, impõe limites à dimensão dessas estruturas, constituindo-se, muitas vezes, como um constrangimento no processo de optimização que implica, indirectamente, um limite no aumento de desempenho do sistema OLAP e consequente diminuição do seu tempo de resposta.

Mas a arquitectura M-OLAP é inerentemente paralela: o processamento de interrogações ou a propagação das alterações havidas nas relações base pode decorrer em paralelo. A estimativa de custos, assumindo uma abordagem tradicional, sequencial, embora suficiente em situações em que apenas se pretende uma valoração comparativa, supondo que o comportamento perante duas situações seja idêntico ao do processamento paralelo, não o será, se esta condição não for satisfeita, e, principalmente, em casos onde um constrangimento tenha de ser obedecido. Desta forma, o *lattice* distribuído estendido com modelo de custos linear e respectivos algoritmos de cálculo de custos de interrogação e manutenção, com simulação de execução paralela de tarefas por discretização

temporal em janelas, foi apresentado numa versão preliminar, e aplicado a um modelo de DW distribuído (perspectiva mais geral do que a arquitectura M-OLAP), mas de perspectivação idêntica, num artigo publicado nas actas da *IASTED International Conference on Databases and Applications (DBA 2006)* [Loureiro & Belo, 2006b]. Neste artigo foi apresentada e justificada a arquitectura DW distribuída, o *lattice* distribuído e respectivo modelo de custos, com apresentação de um algoritmo sequencial de estimativa de custos de interrogação, mas incluindo já um algoritmo de estimativa de custos de manutenção com simulação de execução de tarefas em paralelo.

O modelo foi depois refinado e focado na arquitectura M-OLAP no artigo publicado nas actas da *XI Conference on Software Engineering and Databases (JISBD 2006)* [Loureiro & Belo, 2006g] e ainda no artigo a ser incluído nas actas da *2006 International Database Engineering & Applications Symposium (IDEAS2006)* [Loureiro & Belo, 2006f]. O primeiro artigo apresenta a arquitectura M-OLAP e respectivo modelo de custos, focando depois a sua atenção na descrição e apresentação de três algoritmos de estimativa de custos de manutenção. Baseado ainda no modelo de custos linear, mas utilizando a unidade de custos temporal, introduz o conceito de "janela de execução" e transacção para discretização temporal e cálculo dos custos, supondo a execução paralela das tarefas ligadas à manutenção das estruturas OLAP. É efectuada também a avaliação comparativa dos algoritmos, cujas conclusões revelaram o domínio de aplicabilidade, mostrando assim regras para a sua utilização futura. Havendo já um algoritmo sequencial de estimativa de custos e três paralelos de estimativa de custos de manutenção, foi depois desenvolvido um algoritmo de estimativa de custos de interrogação paralelo, apresentado no segundo dos artigos referidos no início deste parágrafo, sendo aí utilizado, como se verá já a seguir.

Todos os trabalhos acabados de referir, posicionados na vertente 2 desta dissertação (e reflectidos no capítulo 3), foram integrados no ambiente de simulação concebido anteriormente, permitindo a sua aplicação em trabalho de investigação, já no domínio da vertente 1. Assim, em [Loureiro & Belo, 2006f] é mostrada a aplicação do algoritmo genético e algoritmo genético co-evolucionário ao problema de selecção de cubos em ambientes OLAP distribuídos, com avaliação experimental comparativa em conjunto com o algoritmo *distributed node set greedy* proposto em [Bauer & Lehner, 2003] e com um algoritmo genético padrão. As conclusões da análise efectuada mostram que o algoritmo genético co-evolucionário evidencia um desempenho claramente superior ao algoritmo genético padrão e ao algoritmo *greedy*, mostrando o interesse da sua aplicação a este problema.

A optimização discreta por enxame de partículas, o outro método de optimização introduzido e previamente reportado, foi depois utilizado para resolução do problema atrás tratado pelas soluções

evolucionárias. À imagem da utilização do algoritmo genético em [Loureiro & Belo, 2006f], onde foi utilizada também uma sua variante, para a ODiEP foram empregues, além do algoritmo ODiEP base, diversas outras variantes - a cooperativa e multi-fase - sendo também dotadas de hibridações genéticas. O relatório deste trabalho foi aceite para publicação nas actas da *2006 International Conference on Systems, Computing Sciences and Software Engineering (SCSS 06)* [Loureiro & Belo, 2006i], incluindo, além da descrição dos algoritmos e mapeamento do problema no espaço OEP discreto, a respectiva avaliação experimental comparativa. As conclusões mostraram um desempenho algo semelhante da ODiEP normal e variante cooperativa, sendo claramente a multi-fase de menor interesse para este problema (pelo menos na implementação testada).

Ainda em resultado dos trabalhos experimentais levados a efeito em [Loureiro & Belo, 2005], mas especialmente em [Loureiro & Belo, 2006f], surgiu a questão de conhecer o impacto do esquema de selecção no desempenho dos algoritmos genéticos. O porquê da questão estudada, a sua formulação em termos da aplicação dos algoritmos genéticos ao problema da selecção de cubos em ambiente OLAP tradicional e M-OLAP, a descrição dos vários esquemas de selecção a avaliar, os resultados do estudo experimental e conclusões com as regras de utilização foram publicados nas actas da *First International Conference for New Trends in Knowledge Management (KMO 2006)* [Loureiro & Belo, 2006e], estando reflectido nas aplicações dos algoritmos genéticos, no âmbito do presente trabalho. Este estudo é igualmente descrito no capítulo 5 desta dissertação.

Especialmente no decorrer do desenvolvimento dos algoritmos de estimativa de custos de manutenção, percebeu-se a fraqueza do modelo linear distribuído, no suporte à manutenção incremental. De facto, apenas a inclusão da extensão de actualização permitiu a sua adequação, mas de carácter bastante limitado: supunha-se que o custo de geração de cada delta [Mumick et al., 1997] seria afectado por um factor igual, independentemente da agregação gerada e da agregação utilizada (só dependente do respectivo tamanho), e, mais, afectaria por igual qualquer agregação, independentemente do impacto das alterações havidas nas relações base em cada uma das agregações. Assim, o modelo linear distribuído foi evoluído para o domínio da não linearidade, permitindo responder às limitações evidenciadas. As mesmas não linearidades permitiram igualmente refinar o cálculo dos custos de interrogação, permitindo acomodar as diferenças de custos (para além do tamanho intrínseco das agregações utilizadas) implicados pelas diferentes ordenações ou índices, diversos para cada agregação possível. Este modelo não linear distribuído com suporte simultâneo de manutenção integral e incremental que, conforme visto atrás, foi apresentado no capítulo 3, foi publicado nas actas da *18th International Conference on Advanced Information Systems Engineering Forum (CAISE 2006)* [Loureiro & Belo, 2006d]. Quanto aos algoritmos de estimativa de custos de

interrogação e manutenção, com simulação de execução paralela de tarefas numa abordagem *multi-pipelining* [Hennessy & Patterson, 2002], aproveitando o paralelismo inerente da arquitectura M-OLAP, baseados no modelo de custos distribuído não-linear, apresentados parcialmente no capítulo 3, foram tratados em artigo publicado nas actas da *2006 IEEE International Conference on Systems, Man and Cybernetics (SMC 2006)* [Loureiro & Belo, 2006h].

Este novo modelo e algoritmos de estimativa de custos foram aplicados como função de elegância nos algoritmos desenvolvidos até à data (*Greedy*, ODiEP e AG) e em dois outros algoritmos, entretanto concebidos e implementados.

O primeiro adoptou uma pesquisa aleatória, com *hill climbers* e *simulated annealing*, daí o seu acrónimo HC-SA. Se a sua utilização para OLAP centralizado foi já reportada [Kalnis et al., 2002b], ainda que sob uma implementação algo diversa e utilizando um modelo de custos linear, aqui foi proposta a sua aplicação a M-OLAP com modelo de custos não linear e algoritmos de estimativa de custos de simulação de execução paralela de tarefas. Desta forma, a vantagem conhecida deste tipo de algoritmo – a sua escalabilidade – vai ser aproveitada, estendendo-se a sua utilidade a ambientes OLAP distribuídos. Aqui, o objectivo do processo de optimização foi diverso: minimizar os custos de interrogação, observando o constrangimento temporal da janela de manutenção. Uma breve apresentação do modelo, algoritmos de estimativa de custos, a descrição do algoritmo HC-SA, os resultados dos testes experimentais e conclusões respectivas foi descrita em relatório técnico e deu origem a um artigo que será submetido a breve trecho, sendo reflectida no capítulo 6 (secções 6.1 a 6.3) desta dissertação.

O mesmo problema foi tratado por um outro algoritmo, algo como uma mistura de várias das abordagens de optimização tratadas neste trabalho. Considerar uma mistura de algoritmos é, na verdade, uma perspectiva bastante redutora do novo algoritmo. Com efeito, não se trata apenas de ter vários algoritmos de busca em execução paralela, agindo de forma isolada ou operando sob um qualquer esquema de cooperação ou competição, mas, mais do que isso, tem-se um conjunto de elementos em pesquisa de soluções, em que, cada um pode assumir, em sequência, uma de várias formas possíveis. No algoritmo proposto há interacções de dois tipos: aquela que assegura o processo de metamorfose, podendo considerar-se inter-população (já que transvaza as fronteiras de cada população, permitindo a migração das entidades de busca) e outra, inerente a cada uma das populações – intra-população – já que um elemento de busca genético cruza-se com outros e um elemento de busca de um enxame utiliza informação global relativa ao enxame. Este algoritmo constituiu-se como o corolário deste trabalho, porque, além de introduzir um novo conceito de

optimização aplicado ao problema da selecção e alocação de cubos em M-OLAP e mesmo em razão da sua oportunidade temporal, permitiu também a integração numa única plataforma da maioria dos algoritmos propostos. O algoritmo é inspirado no modelo do ciclo de vida descrito em [Krink & Løvbjer, 2002], tendo como característica mais saliente a capacidade da entidade de busca mudar o seu estado, através de metamorfose, assumindo diversas formas, comportando-se em cada uma como um elemento da população do algoritmo correspondente, sendo apropriadamente denominado algoritmo metamorfose. Na versão desenvolvida, cada entidade pode assumir-se como um individuo integrando uma população de um algoritmo genético (na sua versão padrão e, de acordo com as conclusões em [Loureiro & Belo, 2006f], também como um algoritmo genético co-evolucionário), como uma partícula que passa a integrar um enxame de partículas em espaço discreto e um *hill climber* com *simulated annealing*. Cada entidade de busca, além de ter autonomia em termos de procura de soluções, pode decidir também quando desencadear a sua metamorfose. Desta forma, num dado instante da execução do algoritmo, a população de entidades de busca é, em regra, heterogénea, sofrendo a sua composição, variações ao longo da execução do algoritmo. O algoritmo e resultados da avaliação experimental são descritos no capítulo 6 desta dissertação (secções 6.4 e 6.5) e constam de relatório técnico e artigo a ser submetido a conferência, a breve trecho.

Em suma, os modelos de custos e os algoritmos de estimação de custos nas suas diversas versões permitem o cálculo de custos de interrogação e manutenção, integrando parâmetros de custos reais e sua especificação diferenciada e, assim, o suporte a arquitecturas M-OLAP heterogéneas (em termos de facilidades de comunicações e nós). De salientar a simulação de execução paralela de tarefas nos algoritmos de estimação de custos concebidos e implementados, procurando uma maior aproximação à forma como um sistema real se comporta. Também inclusão de não linearidades e especificação, a nível de cada subcubo, do tipo de manutenção a efectuar, permitem aumentar a possível abrangência de aplicação do modelo. Este carece, no entanto, na sua fase actual, de uma avaliação experimental real, tendente, porventura, a desencadear algumas correcções, no que toca à fórmula de cálculo dos custos. De salientar também que os algoritmos desenvolvidos revelaram um bom comportamento quando submetidos a utilização intensiva (atente-se à sua inclusão para avaliação das soluções nos algoritmos de optimização). No entanto, a escalabilidade dos algoritmos de estimação de custos, relativa à complexidade do modelo dimensional OLAP, levanta alguns problemas. A inclusão de conceitos de programação dinâmica na sua génese, se permitiu tempos de execução baixos e assim, uma escalabilidade imediata, pode, para esquemas complexos, constituir uma limitação: o tamanho das estruturas de dados necessárias ao suporte dos valores pré-calculados pode tornar-se excessiva, impondo a utilização de capacidades de memória muito elevadas. Esta limitação manifesta-se

duplamente nos algoritmos de custos que utilizam o modelo de custos não linear. Além de necessitarem da informação relativa às dependências entre cada par de subcubos, precisam também dos valores dos pesos (w_i e w_m) relativos aos custos de interrogação e manutenção. Para este problema uma possível solução poderá ser a adopção de uma solução de compromisso: gerar as dependências e pesos quando requeridos, guardando-os para utilização futura numa *cache*, utilizando uma política de admissão e remoção de cache, a seleccionar, de entre as disponíveis.

Relativamente aos algoritmos de optimização propostos foi mostrada a sua aplicabilidade a sistemas de processamento analítico de repositório único ou múltiplo, sendo aplicadas ao problema da selecção e alocação dos cubos duas novas heurística e variantes de duas outras. Se a versão co-evolucionária do algoritmo genético se revelou de grande interesse em termos da qualidade das soluções obtidas, foi a optimização por enxame de partículas que revelou uma maior escalabilidade, fornecendo igualmente boas soluções. O algoritmo metamorfose permitiu, além do encapsulamento das várias heurísticas numa única solução, a capacidade de selecção adaptativa do algoritmo que, a cada passo, se revelar mais promissor. De salientar que a concepção do algoritmo permite a fácil inclusão de novas heurísticas, existentes ou a surgir. As limitações referidas acima, reflectem-se nestes algoritmos, já que utilizam os algoritmos de estimativa de custos para avaliar cada solução. Também a necessidade de especificação de vários parâmetros e consequente prévia afinação, pode revelar-se limitativa, pelo tempo necessário à sua efectivação e pelas competências que supõe (isto numa possível implementação comercial dos algoritmos). Uma solução possível será a investigação de regras que permitam a especificação automática dos parâmetros e a inclusão de mecanismos de auto-selecção. Uma e outras soluções são já correntes em ferramentas de mineração de dados.

7.3 Áreas de Trabalho Futuro

Como é sobejamente conhecido, um trabalho de investigação é um processo contínuo, feito de avanços e recuos, pesquisando e avaliando caminhos e soluções alternativas, resultando de interacções várias, nomeadamente sob a forma de novas ideias provenientes da comunidade científica da área e divulgação do trabalho próprio realizado. Percebeu-se a evolução havida durante o período em que decorreu esta investigação, mas, também, o quanto poderia ter sido feito (acarretando possíveis benefícios), especialmente quando múltiplos caminhos se abriam ou múltiplas opções de concepção se colocaram. Condicionalismos temporais e disponibilidade de recursos de vária ordem impuseram, muitas vezes, limitações. Em primeiro lugar, o próprio âmbito do trabalho foi orientado

para a investigação de questões julgadas mais relevantes, tendentes à solução dos problemas de administração de um sistema de processamento analítico. Mesmo o processo de concepção e desenvolvimento dos modelos e algoritmos foi algo coarctado, especialmente quando múltiplas opções de concepção se colocavam, cada uma implicando um conjunto de avaliações experimentais bastante demoradas. Assim, muitas possíveis soluções foram abandonadas; outras foram perseguidas, fundamentadas em resultados de testes efectuados num outro contexto, ou simplesmente baseadas em avaliações apriorísticas. Também outros algoritmos de optimização ou suas variantes poderiam ter sido tidos em conta, mas o esforço de concepção, desenvolvimento e teste adicional que implicariam, fizeram optar por aqueles que pareceram os mais adequados ao problema a solucionar. No entanto, todas as opções que se colocaram, que por razões várias foram preteridas ou simplesmente adiadas, foram sendo anotadas. Estas notas constituíram uma boa base de trabalho, cuja análise possibilitou a materialização de um conjunto de propostas para o desenvolvimento da investigação futura. O trabalho a empreender é extenso, sendo assim dividido em duas fases. A primeira, mais imediatista e não implicando grandes recursos adicionais, constitui, na prática, um aditamento ao trabalho empreendido na fase final e descrito no capítulo 6. Já a segunda fase, implicando recursos mais extensos, deverá envolver uma equipa de maiores dimensões, para obtenção de resultados em tempo útil.

Assim, na primeira fase, importa completar alguns trabalhos em execução e avaliar algumas outras opções, especialmente:

- empreender um teste mais exaustivo, focado nos algoritmos descritos no capítulo 6, que aliás, como se viu, abarcam a quase totalidade dos algoritmos propostos anteriormente;
- verificar como se comportam os algoritmos descritos no capítulo 6 na presença do duplo constrangimento espacial e temporal, empreendendo a sua análise comparativa com um algoritmo *greedy* a desenvolver, uma versão para arquitecturas M-OLAP do algoritmo *greedy* de duas fases, que foi recomendado como mais eficiente quando o número de vértices é superior a 120 [Yu et al., 2004]; em alternativa, poder-se-á usar também uma versão baseada no algoritmo *greedy* integrado, também referenciado como escalável [Yu et al., 2004];
- ainda relativamente ao algoritmo genético, importa avaliar do interesse da sua utilização na forma estacionária e utilizando selecção tipo remoção por aptidão uniforme [Legg & Hutter, 2005], já atrás referenciado (secção 4.4.4).

Quanto ao próprio algoritmo MetaMorf M-OLAP, importa implementar a alteração do mecanismo de metamorfose para:

1. incluir um mecanismo probabilístico na transferência da solução da EB actual para a nova forma da entidade de busca, ou seja, com uma probabilidade Pc a solução é copiada e com probabilidade (1-Pc) é gerada uma nova solução;
2. permitir que a nova solução mencionada no ponto anterior possa ser gerada como na geração da população inicial ou aplicando uma forte modificação da solução (mutação em alta escala ou mudança de localização com uma distância de *Hamming* elevada).

Estas propostas de alterações ao mecanismo de metamorfose são motivadas pela verificação experimental da rápida convergência das soluções, o que se revelou algo penalizador da diversidade, com consequente degradação do desempenho. Espera-se que estas modificações, em conjunto com a selecção tipo *fitness uniform deletion* contribuam para melhorar o desempenho global do algoritmo.

Numa segunda fase, e ainda relativamente ao algoritmo MetaMorf M-OLAP importa estudar mais aprofundadamente o mecanismo de metamorfose, nomeadamente:

1. utilizar um mecanismo alternativo ao referido no ponto 1 da enumeração anterior, onde a geração de uma nova solução (como descrita no ponto 2 da enumeração anterior) é controlada não por um simples mecanismo probabilístico (com a sua carga estocástica), mas gerido em função da diversidade efectiva e grau de semelhança com a população de EBs existente;

- permitir que as metamorfoses possam seguir um ciclo diferente do descrito na secção 6.5, especificado pelo utilizador ou mesmo por inclusão de um mecanismo adaptativo; isto implicará a concepção e desenvolvimento de um maior número de métodos relativos às metamorfoses.

Noutra vertente, um dos algoritmos utilizados, o algoritmo genético, além da variante co-evolucionária, mostra agora uma profusão de novas variantes que importa avaliar, na sua aplicação ao problema definido nos capítulos 5 e 6.

Também a abordagem do algoritmo genético com pesquisa local, que deu boas indicações na abordagem centralizada do problema da selecção de cubos [Zhang et al., 2001], deverá merecer atenção especial, neste caso aplicado ao problema da selecção de cubos em M-OLAP. A sua inclusão no algoritmo MetaMorf M-OLAP poderá ser uma implementação a seguir, já que facilitará quer o desenvolvimento, quer o processo de teste. Será interessante efectuar a sua avaliação comparativa

com os restantes algoritmos, e mesmo utilizá-lo no algoritmo MetaMorf M-OLAP, como uma nova variante do algoritmo genético.

Outras duas heurísticas de optimização irão completar o componente "algoritmos de selecção e alocação" da Figura 7.2, onde se mostra o modelo da arquitectura M-OLAP e a utilização dos algoritmos de selecção e alocação de cubos já desenvolvidos, já propostos e a propor.

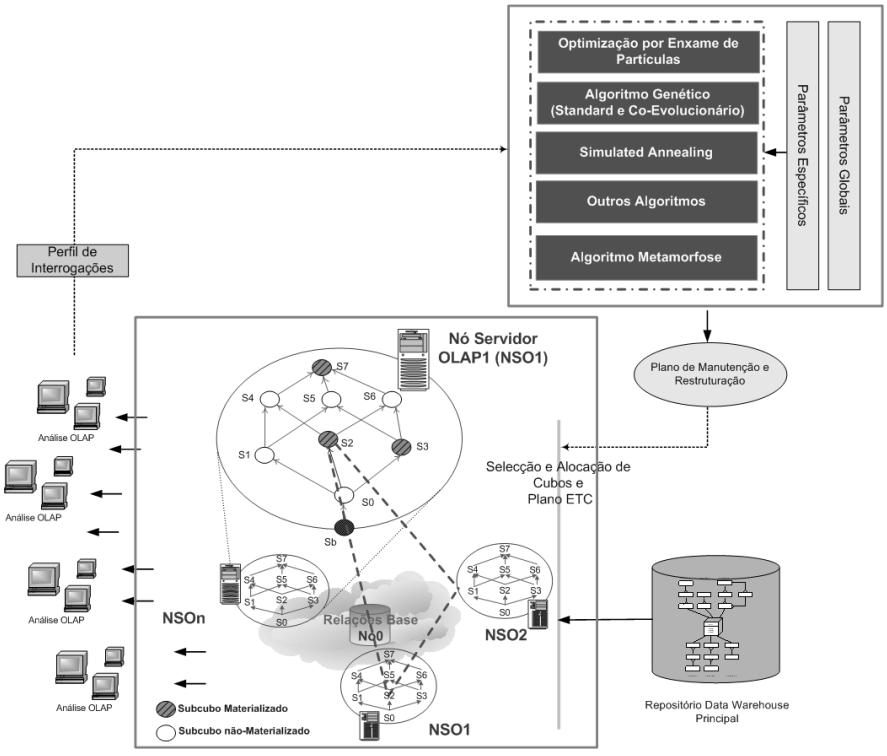


Figura 7.2. Processo de selecção e optimização de cubos em arquitectura M-OLAP.

Uma das heurísticas de optimização a utilizar é inspirada na vida e surgiu com o estudo do comportamento das colónias de formigas. Já atrás foi referida (ver secção 2.8) a sua utilização no problema da selecção de fragmentos de vistas a materializar num DW centralizado. Pretende aplicar-se também ao problema mais geral da selecção e alocação de cubos numa arquitectura M-OLAP. Como então foi referido, tal como muitas das estratégias de busca bem sucedidas, o processo é bastante simples, baseando-se na denominada "*stigmergy*" [Dorigo et al., 2000], uma forma de comunicação indirecta mediada por modificação do ambiente. As formigas, através da deposição de

uma feromona que são capazes de detectar, constroem "caminhos" químicos no seu espaço de manobra. Na prática, modificam o espaço de pesquisa, colocando sinalizadores que vão induzir um comportamento no que toca à tomada de decisão do caminho a seguir. Ao surgir uma encruzilhada, o nível de feromona de cada possível caminho terá um peso forte na decisão a tomar, sendo a outra parcela determinada por um custo apriorístico, se disponível (dependente do problema a ser solucionado). Este mecanismo foi transposto para meios computacionais, sendo proposto um algoritmo inicial denominado sistema de formigas (*Ant System* – AS) [Dorigo et al., 1996]. Mas muitas outras propostas se seguiram no que se convencionou denominar de optimização por colónia de formigas (OCF): *ASrank* [Bullnheimer et al., 1999], *Max-Min Ant System* [Stützle & Hoos, 2000] e *Ant Colony System* [Dorigo & Gambardella, 1997]. Se o paradigma de pesquisa é bastante diverso dos encontrados nas heurísticas atrás utilizadas, e bem adaptado a problemas onde o número de estados não seja muito elevado e onde haja uma medida de custos associada a cada arco de um grafo (como é o caso do TSP) pode, no entanto, ser parcialmente utilizado no problema da selecção e alocação de cubos.

Uma possível solução codificante para o problema pode ser equacionada, onde cada nó do grafo vai representar um subcubo possível a materializar num nó, e os arcos, os caminhos possíveis de selecção para o próximo subcubo. Cada formiga vai-se deslocando de nó em nó, construindo uma solução, considerada completa quando atingir um limite para um qualquer constrangimento imposto. A questão que se coloca à formiga em cada nó é decidir qual o próximo passo a dar (o próximo nó a visitar, ou seja, qual o próximo subcubo a materializar). Esta decisão vai basear-se numa medida de benefício relativa a alguns dos possíveis nós (não todos, para evitar o crescimento da complexidade e consequente tempo de execução) e nível de feromona em cada um dos arcos. Várias heurísticas podem ser adicionadas para melhorar este processo de selecção do próximo passo a dar, como, por exemplo, incluir uma lista tabu ou efectuar uma pesquisa dos caminhos possíveis e seleccionar logo um deles se o nível de feromona estiver acima de um determinado patamar. No final da solução construída, o algoritmo deverá incluir vários tipos de mecanismos de actualização do nível de feromona (de entre os propostos na literatura) e avaliar o seu desempenho.

Uma outra solução, que também vai modificar o próprio espaço de pesquisa, ainda que actuando ao nível dos nós e não dos arcos do grafo, é a pesquisa tabu (PT). É uma solução próxima do *hill climbing*, pois que também usa HCs, empreendendo uma pesquisa do tipo melhorativo com recurso a uma estrutura de vizinhança, que permite, à pesquisa, percorrer o espaço de soluções. A fase de melhoramento repete-se até que um critério de paragem seja accionado. Durante o processo de melhoramento, o algoritmo memoriza selectivamente alguns elementos chave do caminho percorrido,

o que permite direccionar, de forma inteligente, a pesquisa no espaço de soluções. A memorização faz-se, recorrendo a uma lista de soluções já exploradas (a lista tabu) ou informação relevante relativa às soluções, neste caso, o sucesso da materialização ou desmaterialização de um dado subcubo (o processo de *stigmergy*). A apetência do HC por um dado nó (a seleccionar ou remover da solução) irá ser determinado pela informação relativa ao sucesso anterior da operação, além da inclusão de estratégias de diversificação que levem a explorar regiões do espaço menos visitadas (evitando a rápida convergência do processo de pesquisa), forçando movimentos para longe dos óptimos locais. Glover e Laguna [Glover & Laguna, 1997] indicam estratégias de intensificação e diversificação que podem ser implementadas. A informação relativa à apetência (ou não apetência) de cada nó vai ser actualizada no final de cada iteração, usando um mecanismo autocatalítico (*feedback* positivo) semelhante à deposição de feromona na OCF.

Para finalizar, e como se depreende das propostas referidas nesta secção, percebe-se que o trabalho desenvolvido constituiu-se como uma parte, ainda que importante e estruturante, do sistema a desenvolver. Na verdade, a extensão do problema a solucionar e o conjunto de áreas de conhecimento envolvidas, impôs a escolha de algumas vertentes em detrimento de outras, ao seleccionar as linhas de acção da investigação a empreender. A concepção dos modelos de custos e algoritmos de estimativa de custos foram o referencial que permitiu depois prosseguir a investigação no domínio da aplicação dos algoritmos de optimização. Especialmente na vertente das soluções de distribuição das estruturas multidimensionais muito foi aqui feito, mas muito há ainda a fazer. Se as soluções propostas são já abrangentes, podendo ser aplicadas em arquitecturas diversificadas e heterogéneas na optimização das estruturas multidimensionais, importa avaliá-las melhor, para compreender mais cabalmente as suas especificidades de aplicação e, também, incluir os algoritmos de optimização já previstos e outros, a surgir neste domínio em constante evolução. Contudo, a arquitectura actual do componente de optimização disponibiliza já um conjunto alargado de opções passíveis de utilização fácil e num leque alargado de arquitecturas possíveis do sistema de processamento analítico, com especial ênfase na sua vertente distribuída, permitindo uma escalaridade fácil a preços controlados. A colaboração futura, mais alargada, com outros investigadores, permitirá, decerto, aprofundar o conhecimento em alguns quadrantes do problema, porventura, até agora, menos tratados, mas permitidos pela infra-estrutura já desenvolvida. Na verdade, a concepção arquitectural aberta permite acomodar novas ideias e prosseguir por novos rumos.

Bibliografia

[Abbot & Garcia-Molina, 1992] K. Abbot and Hector Garcia-Molina: "Scheduling Real-Time Transactions: A Performance Evaluation. ACM Transactions on Database Systems". Vol. 17, No. 3, Set. 1992, pp. 513-560.

[Adams, 2005] J. Adams: "Analyse Your Company Using SWOTs". Supply House Times, vol. 48, Issue 7, pp. 26-28.

[Al-Kazemi & Mohan, 2002] Buthainah Al-kazemi, and Chilukuri K. Mohan: "Multi-phase Discrete Particle Swarm Optimization". In Proc. of the Fourth International Workshop on Frontiers in Evolutionary Algorithms (FEA 2002), pp. 622-625.

[Angeline, 1998] P. Angeline: "Using Selection to Improve Particle Swarm Optimization". Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC'98), Anchorage, Alaska, USA, May 4-9, 1998.

[Baralis et al., 1997] E. Baralis, S. Paraboschi, and E. Teniente: "Materialized View Selection in a Multidimensional Database". In: Proceedings of the 23rd International Conference on Very Large Data Base (VLDB), Athens, Greece, August 1997, pp. 156-165.

[Baker, 1985] J. E. Baker: "Adaptive Selection Methods for Genetic Algorithms". Proceedings of an International Conference on Genetic Algorithms and their Applications, Carnegie Mellon Publishers, 1985, pp. 101-111.

[Bauer & Lehner, 2003] A. Bauer, and W. Lehner: "On Solving the View Selection Problem in Distributed Data Warehouse Architectures". In Proceedings of the 15th International Conference on Scientific and Statistical Database Management (SSDBM'03), IEEE (2003), 43-51.

[Belo, O., 2000] Orlando Belo: "Putting Intelligent Personal Assistants Working on Dynamic Hypercube Views Updating". Proceedings of 2nd International Symposium on Robotics and Automation (ISRA'2000), Monterrey, México, November, 2000.

[Bellman, R., 1958] Richard Bellman: "On a Routing Problem". Quarterly of Applied Mathematics, 16(1), 1958, pp. 87-90.

[Bergh & Engelbrecht, 2004] F. Van den Bergh, and A. P. Engelbrecht: "A Cooperative Approach to Particle Swarm Optimization". In IEEE Transactions on Evolutionary Computation, Vol. 8, No. 3, pp. 225-239, June 2004.

[Bertsekas, D., 2000] Dimitri P. Bertsekas: "Dynamic Programming and Optimal Control", 2ª edição. Athena Scientific. Vol. 1 e 2.

[Blickle & Thiele, 1995] T. Blickle, and L. Thiele: "A Mathematical Analysis of Tournament Selection". In L. J. Eshelman, editor, Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA'95), San Francisco, California, 1995, pp. 9–16.

[Blickle & Thiele, 1996] T. Blickle, and L. Thiele: "A Comparison of Selection Schemes used in Genetic Algorithms". In Evolutionary Computation, vol. 4, Issue 4, Winter 1996, the MIT Press, pp. 361-394.

[Bullnheimer et al., 1999] B. Bullnheimer, R. F. Hartl, and C. Strauss: "A new Rank-Based Version of the Ant System: A Computational Study". In Central European Journal for Operations Research and Economics, 7(1) (1999), pp. 25-38.

[Cao et al. 1998] P. Cao, J. Zhang, and P.B. Beach: "Active Caching: Caching Dynamic Contents on the Web". Proceedings of Middleware '98 Conference, 1998.

[Chaudury & Dayall, 1997] S. Chaudhuri, and U. Dayal: "An Overview of Data Warehouse and OLAP Technology". ACM SIGMOD Record 26(1), 1997, pp. 65-74.

[Chen & Roussopoulos, 1994] C. Chen and N. Roussopoulos: "The Implementation and Performance Evaluation of the ADMS Query Optimizer: Integrating Query Result Caching and Matching". Proceedings of the International Conference on Extending Database Technology, 1994.

[Clark & Wright, 1964] G. Clark, and J. W. Wright: "Scheduling of Vehicles from a Central Depot to a Volume of Delivery Points". Operations Research, 12, 1964, pp. 568-581.

[Codd. et al., 1993] E.F. Codd, S.B. Codd, and C.T. Sulley: "Providing OLAP (On-Line Analytical Processing) to User Analysts: An IT Mandate". Technical Report, 1993.

[Corloni et al. 1996] A. Colorni, M. Dorigo, F. Mafioli, V. Maniezzo, G. Righini, M. Trubian: "Heuristics from Nature for Hard Combinatorial Optimization Problems". International Transactions in Operational Research, 3, 1, 1996, pp. 1-21.

[Cormen et al., 2001] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein: "Introduction to Algorithms", Second Edition. MIT Press and McGraw-Hill, 2001. Section 24.3: Dijkstra's algorithm, pp. 595-601; representation of Graphs, pp. 527-531.

[Dantzig, 1963] G. B. Dantzig: "Linear Programming and Extensions". Princeton University Press, 1963.

[Darwin, 1859] C. Darwin: "On the Origin of Species". London: John Murray, 1859.

[Das & Chakrabarti, 2005] Arnab Das and K. Chakrabarti (eds.): "Quantum Annealing and Related Optimization Methods". Lecture Note in Physics, vol. 679, Springer, Heidelberg (2005).

[Dawkins, 1976] R. Dawkins: "The Selfish Gene". Oxford University Press, 1976.

[de Boer et al., 2005] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein: "A Tutorial on the Cross-Entropy Method". In: Annals of Operations Research, 134 (1), pp. 19-67.

[de la Maza & Tidor, 1993] Michael de la Maza, and Bruce Tidor: "An Analysis of Selection procedures with Particular Attention paid to Proportional and Bolzmann Selection". In: Stefanic Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, CA. Morgan Kaufmann Publishers, 1993, pp. 124-131.

[Desphand et al., 1998] P. Desphande, K. Ramasamy, A. Shukla, and J.F. Naughton: "Caching Multidimensional Queries Using Chunks". Proceedings of ACM SIGMOD, 1998.

[Diestel, 2005] Reinhard Diestel: "Graph Theory", 3rd edition. Springer-Verlag Heidelberg, New-York, 2005, edição electrónica, disponível em http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/.

[Dijkstra, 1959] E. W. Dijkstra: "A note on two problems in connexion with graphs". 1 (1959), S. 269-271.

[Dorigo et al., 1996] M. Dorigo, M. Maniezzo, and A. Colorni: "The Ant System: Optimization by a Colony of Cooperating Agents". In IEEE Transactions on Systems, Man, and Cybernetics – Part B, 26(1), 1996, pp. 29-41.

[Dorigo & Gambardella, 1997] M. Dorigo, and L. M. Gambardella:" Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem". In: IEEE Transactions on Evolutionary Computation, 1(1), Abril 1997, pp. 53-66.

[Dorigo et al., 2000] M. Dorigo, E. Bonabeau, and G. Theraulaz: "Ant Algorithms and stigmergy". In Future Generation Computer Systems, 16(8), 2000, pp. 851-871.

[Dueck & Scheuer, 1990] G. Dueck and T. Scheuer: "Threshold Accepting: A General Purpose Optimization Algorithm Superior to Simulated Annealing". In Journal of Cmputational Physics, col. 90, 1990, pp. 161-175.

[Eberhart & Kennedy 1995] Russell C. Eberhart, and James Kennedy, J.: "A new Optimizer Using Particle Swarm Theory". In Proc. Os the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, IEEE Service Center, Piscataway, NJ, pp. 39-43.

[Eberhart & Shi, 2001] Russel C. Eberhart, and Y. Shi: "Particle Swarm Optimization: Developments, Applications and Resources". Proceedings of the 2001 Congress on Evolutionary Computation. Vol. 1, 2001, pp. 81-86.

[Finnila et al., 1994] A. B. Finnila, M. A. Gomez, C. Sebenik, C. Stenson, and J. D. Doll: "Quantum Annealing: A New Method for Minimizing Muitidimensional Features". Chem. Phys Lett. 219, 1994, pp. 343-348.

[Fischetti & Toth, 1992] M. Fischetti, and P. Toth: "An Additive Bounding Procedure for the Asymmetric Traveling Salesman Problem". Mathematical Programming, vol. 53, 1992, pp. 173-197.

[Floyd, 1962] Robert W. Floyd: "Algorithm 97: Shortest Path". Communications of the ACM 5 (6): 345.

[Garey & Johnson, 1979] M. R. Garey, and D. S. Johnson: "Computers and Intractability: A Guide to the Theory of NP-Completeness". Freeman, San Francisco, CA, 1979.

[Glover, 1986] Fred Glover: Future Paths for Integer Programming and Links to Artificial Intelligence". Computers & Operations Research, 13, pp. 533-549.

[Glover & Laguna, 1997] Fred Glover, and Manuel Laguna: "Tabu Search". Kluwer Academic Publishers, 1997.

[Goldberg, 1989] D.E. Goldberg: "Genetic Algorithms in Search, Optimization, and Machine Learning". Addison-Wesley, Reading, MA, 1989.

[Gray et al., 1996] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh: "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals". Proceedings of the 12th International Conference on Data Engineering, New Orleans, Feb 1996, IEEE.

[Griffin & Likkin, 1995] T. Griffin, and L. Likkin: "Incremental Maintenance of Views with Duplicates". In Proceeding of the ACM SIGMOD 1995 International Conference on Management of Data, San Jose, CA, May 23-25.

[Goldberg, 1989] D.E. Goldberg: "Genetic Algorithms in Search, Optimization, and Machine Learning". Addison-Wesley, Reading, MA, 1989.

[Grefenstette & Baker, 1989] John J. Grefenstette, and James E. Baker: "How Genetic Algorithms Work: A Critical Look at Implicit Parallelism". In: J. David Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA. Morgan Kaufmann Publishers (1989), pp. 20-27.

[Gupta, A. et al., 1993] A. Gupta, T. Mumick, and V. Subrahmanian: "Maintaining Views Incrementally". In Proceedings of ACM SIGMOD 1993 International Conference on Management of Data, Washington, DC, May 26-28, 1993.

[Gupta, A. et al., 1995] A. Gupta, V. Harinarayan, and D. Quass: "Aggregate-Query Processing in a Data Warehousing Environments". In Proceedings of the 21st International Conference on Very Large Databases (VLDB'95), Zurich, Switzerland, 1995.

[Gupta, H. et al., 1997] Himanshu Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman: "Index Selection for OLAP". In: Proceedings of the Intl. Conf. on Data Engineering, Birmingham, UK, April 1997, pp. 208-219. Disponível em **Error! Reference source not found.**.

[Gupta, H., 1997] Himanshu Gupta: "Selection of Views to Materialize in a Data Warehouse". In: Proceedings of the 6th International Conference on Database Theory, Lecture Notes in Computer Science, vol. 1186, pp. 98-112.

[Gupta & Mumick, 1999] Himanshu Gupta, and Inderpal Singh Mumick: "Selection of Views to Materialize under a Maintenance-Time Constraint". In: Proceedings of the International Conference on Database Theory, 1999. Disponível em [6].

[Hancock, 1994] P. J. B. Hancock: "An Empirical Comparison of Selection Methods in Evolutionary Algorithms". Proceedings of the AISB Workshop on Evolutionary Computing, (T. C. Fogarty, ed.), Springer, 1994, pp. 80-94.

[Harinarayan et al., 1996] V. Harinarayan, A. Rajaraman, and J. Ullman: "Implementing Data Cubes Efficiently". Proceedings of ACM SIGMOD, Montreal, Canada, June 1996, pp. 205-216. Disponível em [6].

[Hart et al., 1968] P. E. Hart, N. J. Nilsson, B. Raphael: "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics SSC4(2), 1968, pp. 100-107.

[Hennessy & Patterson, 2002] J. Hennessy, and D. Patterson: "Computer Architecture: A Quantitative Approach", third edition. The Morgan Kaufmann Series in Computer Architecture and Design, 2002.

[Hillier & Lieberman, 2005] Hillier and Lieberman: "Introduction to Operations Research". McGraw-Hill, 2005, Eighth Edition.

[Holland, 1992] J.H. Holland: "Adaptation in Natural and Artificial Systems". MIT Press, Cambridge, MA, (2nd edition), 1992.

[Huang & Chen, 2001] Y.-F. Huang, and J.-H. Chen: "Fragment Allocation in Distributed Database Design". In Journal of Information Science and Engineering 17, 2001, pp. 491-506.

[Hunger & Wheelen, 2003] J. David Hunger, and Thomas L. Wheelen: "Essentials of Strategic Management". New Jersey: Pearson Educations Inc, 2003.

[Hutter, 2002] M. Hutter: "Fitness Uniform Selection to Preserve Genetic Diversity", Proc. of 2002 Congress on Evolutionary Computation (CEC-2002), Washington D.C., USA, May 2002, IEEE, pp. 783-788.

[Inmon, 1996] W. H. Inmon: "Building the Data Warehouse", second edition. Wiley Computer Publishing, 1996.

[Horng et al., 1999] J.T. Horng, Y.J. Chang, B.J. Liu, and C.Y. Kao: "Materialized View Selection Using Genetic Algorithms in a Data Warehouse". In Proceedings of World Congress on Evolutionary Computation, Washington D.C., July, 1999.

[Informatica, 1997] Informatica: "Enterprise-Scalable Data Marts: A New Strategy for Building and Deploying Fast, Scalable Data Warehousing Systems". White Paper, http://www.informatica.com, 1997.

[Jamil & Modica, 2001] H.M. Jamil, and G.A. Modica: "A View Selection Tool for Multidimensional Databases". In L. Monostori, J. Váncza, and M. Ali (eds.). Proceedings on the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Budapest, Hungary, June 2001. Lecture Notes in Computer Science 2070, Springer, Berlin, pp. 237-246.

[Kalnis & Papadias, 2001] Panos Kalnis, and Dimitris Papadias: "Proxy-Server Architectures for OLAP". Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, CA, 2001, pp. 367-378.

[Kalnis et al., 2002a] Panos Kalnis, W.S. Ng, B.C. Ooi, Dimitris Papadias and K.L. Tan: "An Adaptive Peer-to-Peer Network for Distributed caching of OLAP Results". Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD), Wisconsin, 2002.

[Kalnis et al., 2002b] Panos Kalnis, Nikos Mamoulis, and Dimitris Papadias : "View Selection Using Randomized Search". Data Knowledge Engineering, vol. 42, number 1, 2002, pp. 89-111.

[Karayannidis & Sellis, 2001] N. Karayannidis, and T. Sellis: "SISYPHUS: A Chunk-Based Storage Manager for OLAP Cubes". Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2001). Interlaken, Switzerland, June, 4, 2001.

[Kennedy & Eberhart, 1995] James Kennedy, and Russell C. Eberhart: "Particle Swarm Optimization". In Proceedings of IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV, pp. 1942-1948.

[Kennedy & Eberhart, 1997] Kennedy, J., and Eberhart, R.C. "A Discrete Binary Version of the Particle Swarm Optimization Algorithm.". In Proc. of the 1997 Conference on Systems, Man and Cybernetics (SMC'97), 1997, pp. 4104-4109.

[Kimball, 1996] R. Kimball: "Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses". John Wiley & Sons, 1996.

[Kirkpatrick et al., 1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi: "Optimization by Simulated Annealing". Science, vol. 220, pp. 671-680, 1983.

[Koza, 1992] John R. Koza: "Genetic Programming: On the Programming of Computers by Means of Natural Selection". The MIT Press, Cambridge, Massachusetts (1992).

[Kotidis & Roussopoulos, 1999] Y. Kotidis, and N. Roussopoulos: "Dynamat: A Dynamic View Management System for Data Warehouses". Proceedings of the ACM SIGMOD International Conference on Management of Data (Philadelphia, Pennsylvania, June 1999), pp.371-382.

[Krink & Løvbjer, 2002] Thiemo Krink, and Morten Løvbjer: "The LifeCycle model: Combining Particle Swarm Optimization, Genetic Algorithms and HillClimbers". Proceedings of Parallel Problem Solving from nature VII (PPSN-2002), 621-630.

[Lam & Hung, 1995] K. Lam, and S. Hung: "Concurrency Control for Time-constrained Transactions in Distributed Database Systems". The Computer Journal. Vol. 38, No. 9, 1995, pp. 704-716.

[Land & Doig, 1960] A. H. Land, and A. G. Doig: "An Automatic Method for Solving Discrete Programming Problems". Econometria, Vol. 28, 1960, pp. 497-520.

[Legg & Hutter, 2005] S. Legg, and M. Hutter: "Fitness Uniform Deletion: A Simple Way to Preserve Diversity". Proc. of Genetic and Evolutionary Computation Conference, GECCO 2005, Washington DC, USA, June 25-29, 2005 , pp. 1271-1278.

[Liang et al., 2001] W. Liang, H. Wang, and M.E. Orlowska: "Materialized View Selection Under the Maintenance Cost Constraint". Data and Knowledge Engineering, 37(2), 2001, pp. 203-216.

[Lin & Kuo, 2004] W.-Y. Lin, and I.C. Kuo: "A Genetic Selection Algorithm for OLAP Data Cubes". In Knowledge and Information Systems, Volume 6, Number 1, Springer-Verlag London Ltd., 2004, pp. 83-102.

[Loureiro, J., 1999] Jorge Loureiro: "Data Warehousing: Tecnologia e Estratégia para DSS". Dissertação de Mestrado, Universidade de Coimbra, 1999.

[Loureiro & Belo, 2005] Jorge Loureiro e Orlando Belo: "Genetic and Swarm Algorithms for the Selection of OLAP Data Cubes". Proceedings of the 4th WSEAS International Conference on Information Security, Communications and Computers (ISCOCO'05), Canary Islands, 16-18 Dezembro, 2005.

[Loureiro & Belo, 2006a] Jorge Loureiro e Orlando Belo: "Life Inspired Algorithms for the Selection of OLAP Data Cubes". WSEAS Transactions on Computers, Issue 1, Volume 5, Janeiro 2006, pp. 8-14.

[Loureiro & Belo, 2006b] Jorge Loureiro e Orlando Belo: "Estimating Querying and Maintenance Costs for Reestructuring Data Cubes". Proceedings of the IASTED International Conference on Databases and Applications (DBA 2006), Innsbruck, Áustria, 14-16 Fevereiro, 2006.

[Loureiro & Belo, 2006c] Jorge Loureiro e Orlando Belo: "A Discrete Particle Swarm Algorithm for OLAP Data Cube Selection". In Proceedings of 8th International Conference on Enterprise Information Systems (ICEIS 2006), Databases and Information Systems Integration, Paphos, Cyprus, 23-27 Maio, 2006, pp. 46-53.

[Loureiro & Belo, 2006d] Jorge Loureiro e Orlando Belo: "Non-Linear Cost Model for Multi-Node OLAP Cubes". In Proceedings of 18th International Conference on Advanced Information Systems Engineering (CAISE2006) Forum, Luxembourg, 5-9 Junho, 2006.

[Loureiro & Belo, 2006e] Jorge Loureiro e Orlando Belo: "Comparative Study of Genetic Algorithms Selection Schemes Performance when Applied to the Cube Selection Problem". In Proceedings of 2006 Knowledge Management in Organizations (KMO2006), Maribor, Slovenia, 13-14 Junho, 2006.

[Loureiro & Belo, 2006f] Jorge Loureiro e Orlando Belo: "An Evolutionary Approach to the Selection and Allocation of Distributed Cubes". Artigo a ser incluído nos Proceedings da 2006 International Database Engineering & Applications Symposium (IDEAS2006), Delhi, India, December 11-14, 2006.

[Loureiro & Belo, 2006g] Jorge Loureiro e Orlando Belo: "Evaluating Maintenance Cost Computing Algorithms for Multi-Node OLAP Systems". In Proceedings da XI Conference on Software Engineering and Databases (JISBD 2006), Sitges (Barcelona), Spain, 3-6 Outubro, 2006.

[Loureiro & Belo, 2006h] Jorge Loureiro e Orlando Belo: "Establishing more Suitable Distributed Plans for MultiNode-OLAP Systems". In Proceedings da 2006 IEEE International Conference on Systems, Man and Cybernetics (SMC 2006), Taipei, Taiwan, 8-11 Outubro, 2006, pp. 3573-3578.

[Loureiro & Belo, 2006i] Jorge Loureiro e Orlando Belo: "Swarm Intelligence in Cube Selection and Allocation for Multi-Node OLAP Systems". Artigo aceite para publicação nos Proceedings da 2006 International Conference on Systems, Computing Sciences and Software Engineering (SCSS 06), Dezembro, 2006.

[Loureiro & Belo, 2006-ra] Jorge Loureiro e Orlando Belo: "Query and Maintenance Estimation Cost Algorithms for a MultiNode OLAP Architecture Using a non-Linear Cost Model", Relatório Técnico, Universidade do Minho, Março 2006.

[Loureiro & Belo, 2006-rb] Jorge Loureiro e Orlando Belo: "Metomorphosis Algorithm for the Optimization of Multi-Node OLAP Systems". Relatório Técnico, Universidade do Minho, Setembro, 2006.

[Løvbjerg et al., 2001] M. Løvbjerg, T. Rasmussen, and T. Krink: "Hybrid Particle Swarm Optimization with Breeding and Subpopulations". In Proceedings of the 3rd Genetic and Evolutionary Computation Conference (GECCO-2001).

[Maniezzo, 1999] Vittorio Maniezzo: "Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem. INFORMS Journal on Computing, 11(4), 1999, pp. 358-369.

[Maniezzo et al., 2001] Vittorio Maniezzo, Antonella Carboraro, Matteo Golfarelli, and Stefano Rizzi: "An ANTS Algorithm for Optimizing the Materialization of Fragmented Views in Data Warehouses: Preliminary Results". Proceedings of the EvoWorkshops on Applications of Evolutionary Computing, 2001, Lectures Notes in Computer Science, vol. 2037, Springer-Verlag, pp. 80-89.

[Männer & Manderick, 1992] R. Männer, and B. Manderick (eds.): Parallel Problem Solving from nature, 2, North Holland, 1992.

[Merz & Freisleben, 1999] P. Merz and B. Freisleben: "Fitness landscapes and Memetic Algorithm Design". In New Ideas in Optimization, (D. Corne, M. Dorigo, and F. Glover, eds.), McGraw-Hill, 1999, pp. 245-260.

[Merz & Freisleben, 2000] P. Merz and B. Freisleben: "Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning". In Evolutionary Computation, vol. 8, no. 1, 2000, pp. 61-91.

[Mitchel et al., 1994] T. Mitchel, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski: "Experience with Learning Personal Assistant". Communications of the ACM, 37(7):80-91, July, 1994.

[Mitchel, 1996] M. Mitchell: "An Introduction to Genetic Algorithms". MIT Press, Cambridge, MA (1996).

[Moller & Pekny, 1991] D. L. Moller, and J. F. Pekny: "Exact Solution of Large Assymetric Traveling Salesman Problems". Science, vol. 251, 1991, pp. 754-761.

[Moore, 1965] Gordon E. Moore: "Cramming more components onto integrated circuits". Electronics, Volume 38, Número 8, 19 de Abril, 1969.

[Moscato, 1999] P. Moscato: "Memetic Algorithms: A Short Introduction". In New Ideas in Optimization, (D. Corne, M. Dorigo, and F. Glover, eds), McGraw-Hill, London, 1999, Capítulo 14, pp. 219-234.

[Mühlenbein & Schlierkamp-Voosen, 1994] Heinz Mühlenbein and Dirk Schlierkamp-Voosen: "The science of breeding and its application to the breeder genetic algorithm (BGA)". In Evolutionary Computation, 1(4), 1994, pp. 335–360.

[Mumick et al., 1997] I. Mumick, D. Quass, and B. Mumick: "Maintenance of Data Cubes and Summary Tables in a Warehouse". In Peckham J (ed): Proceedings of ACM SIGMOD International Conference of Management of Data, Tucson, Arizona (June 1997), pp. 100-111. Disponível em [6].

[Munneke et al., 1999] D. Munneke, K. Wahlstrom, and M. Mohania: "Fragmentation of Multidimensional Databases". Proceedings of 10th Australasian Database Conference, Auckland, 1999, pp. 153-164.

[Ozsu & Valduriez, 1997] M. T. Ozsu, and P. Valduriez. "Distributed and Parallel Database Systems". Handbook of Computer Science and Engineering. CRC Press. pp. 1093-1111.

[Padberg & Rinaldi, 1987] M. Padberg, and G. Rinaldi: "Optimization of a 532-city Symmetric Traveling Salesman Problem by Branch & Cut". Operations research Letters, vol. 6, 1987, pp. 1-7.

[Padberg & Rinaldi, 1991] M. Padberg, and G. Rinaldi: "A Branch-and-Cut Algorithm for the resolution of Large-Scale Symmetric Traveling Salesman Problems". SIAM Review, vol. 33, March 1991, pp. 60-100.

[Papadimitriou, 1994] C.H. Papadimitriou: "Computational Complexity". Addison-Wesley, Reading, MA, 1994.

[Park et al., 2003] C.S. Park, M.H. Kim, and Y.J. Lee: "Usability-based Caching of Query Results in OLAP Systems". Journal of Systems and Software, Vol. 68, Issue 2, November 2003, pp. 103-119.

[Pearce & Robinson, 2005] J. Pearce, R. Robinson: "Strategic Management", 9ª edição, New York: McGraw-Hill.

[Porter, 1980] Michael E. Porter: "Competitive Advantage: Techniques for Analysing Industries and Competitors", New York, NY The Free Press.

[Porter, 1985] Michael E. Porter: "Competitive Advantage: Creating and Sustaining Superior Performance", New York, NY The Free Press.

[Poter & Jong, 1994] M. A. Potter, K. A. Jong: "A Cooperative Coevolutionary Approach to Function Optimization". In The Third Parallel Problem Solving From Nature. Berlin, Germany: Springer-Verlag, (1994) 249-257.

[Roy et al., 2000] Prasan Roy, Krithi Ramamritham, S. Seshadri, Pradeep Shenoy, and S. Sudarshan: "Don't Trash your Intermediate Results, Cache 'em". CoRR cs.DB/0003005, 2000.

[Sapia et al., 1999] C. Sapia, M. Blaschka, G. Höfling, and B. Dinter: "Extending the E/R Model for the Multidimensional Paradigm". In Advances in Database Technologies (ER'98 Workshop Proceedings), Springer-Verlag, pp. 105-116.

[Sapia, 2000] C. Sapia: "PROMISE – Modelling and Predicting User Query Behaviour in Online Analytical Processing Environments". Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery (DAWAK'00), London, UK, Springer LNCS, September, 2000.

[Scheuermann et al., 1996] Peter Scheuermann, Junho Shim, and Radek Vingralek: "WATCHMAN: A Data Warehouse Intelligent Cache Manager". Proceedings of 22th International Conference on Very Large Data Base (VLDB'96), September 3-6, Mumbay (Bombay), India, 1996, pp. 51-62.

[Schwefel & Männer, 1990] H. P. Schwefel, and R. Männer: "Parallel Problem Solving from Nature", Proceedings of PPSN-I, 1990.

[Shi & Eberhart, 1998a] Y. Shi, and Russell C. Eberhart: "Communication in Particle Swarm Optimization Illustrated by the Travelling Salesman Problem". Evolutionary Programming VII: Proc. EP98, New York, Springer-Verlag, 1998, pp. 591-600.

[Shi & Eberhart, 1998b] Y. Shi, and Russell C. Eberhart: "Parameter Selection in Particle Swarm Optimization". Proceedings of the IEEE International Conference on Evolutionary Computation, Piscataway, NJ: IEEE Press, 1998, pp.69-73.

[Shi & Eberhart, 1999] Y. Shi, and Russell C. Eberhart: "Empirical Study of Particle Swarm Optimization". Proceedings of the 1999 Congress of Evolutionary Computation, vol. 3, IEEE Press, pp. 1945-1950.

[Shim et al., 1999] J. Shim, P. Scheuermann, and R. Vingralek: "Dynamic Caching of Query Results for Decision Support Systems". Proceedings of 11th International Conference on Scientific and Statistical Database Management, Cleveland, Ohio, July, 1999.

[Shukla et al., 1998] A. Shukla, P.M. Deshpande, and J.F. Naughton: "Materialized View Selection for Multidimensional Datasets". In: Proc. of Very Large DataBases, VLDB, 1998.

[Shukla et al., 2000] A. Shukla, P.M. Deshpande, and J.F. Naughton: "Materialized View Selection form Multicube Data Models". In C. Zaniolo, P.C. Lockemann, M.H. Scholl, and G. Torsten (eds.). Proceedings of Advances in Database Technology (EDBT'00), 7th International Conference on Extended Database Technology, Konstanz, Germany, March 2000. Lecture Notes in Computer Science 1777, Springer, Berlin, pp. 269.284.

[Soutyna & Fotouhi, 1997] E. Soutyina, and F. Fotouhi: "Optimal View Selection form Multidimensional Database Systems". In Proceedings of 1997 International Database Engineering and Applications Symposium, 1997, pp. 309-318.

[Stonebraker et al., 1994] Michael Stonebraker, Robert Devine, Marcel Kornacker, Witold Litwin, Avi Pfeffer, Adam Sah, Carl Staelin: "An Economic paradigm for Query Processing and Data Migration in Mariposa". In Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, Austin, TX, USA, 28-30, September 1994, pp. 58-68.

[Stützle & Hoos, 2000] T. Stützle, and H. Hoos: "Max-Min Ant System". In: Future Generation Computer Systems, 16(8), 2000, pp. 889-914.

[Syswerda, 1989] G. Syswerda, "Uniform Crossover in genetic Algorithms". Proceedings of the 3rd International Conference on Genetic Algorithms, (J.D. Shaffer, ed.), Morgan Kaufmann, 1989, pp. 2-9.

[Syswerda, 1991] G. Syswerda: "A Study of Reproduction in Generational and Steady State Genetic Algorithms". Foundations of Genetic Algorithms, (G. J. E. Rawlings, ed.), San Mateo: Morgan Kaufmann, 1991, pp. 94-101.

[Taillard et al., 1998] Éric D. Taillard, Luca M. Gambardella, Michel Gendreau, and Jean-Yves Potvin: "Adaptive Memory Programming: A Unified View of Metaheuristics". EURO XVI Conference Tutorial and Research Reviews Booklet (semi-plenary sessions), Brussels, July 1998.

[Thomson, 1997] E. Thomson. "OLAP Solutions: Building Multidimensional Information Systems". Wiley, 1997.

[Theodoratos & Selis, 1997] D. Theodoratos, and T. Sellis: "Data Warehouse Configuration". In M. Jarke, M.J. Carey, K.R. Dittrich et al. (eds.). Proceedings of the 23rd International Conference on Very Large Databases, Athens, Greece, August 1997, pp. 126-135.

[Theodoratos & Bouzeghoub, 2000] D. Theodoratos, and M. Bouzeghoub: "A General Framework for the View Selection Problem for Data Warehouse Design and Evolution". In Proceedings of the 3th ACM International Workshop on Data Warehousing and OLAP, Washington, DC, November, 2000, pp. 1-8.

[TPC-R, 2002] Transaction Processing Performance Council (TPC): TPC Benchmark R (decision support) Standard Specification Revision 2.1.0. tpcr_2.1.0.pdf, disponível em www.[7].

[Ulusoy & Belford, 1993] O. Ulusoy, and G. Belford. "Real-Time Transaction Scheduling in Database Systems". Information Systerns. Vol. 18, No. 8, 1993, pp. 559-580.

[Walrand & Varaiya, 2000] J. Walrand, and P. Varaiya: "High-Performance Communication Networks", The Morgan Kaufmann Series in Networking (2000).

[White & Pagurek, 1998] T. White, and B. Pagurek: "Towards Multi-Swarm Problem Solving in Networks". Proceedings of the Third International Conference on Multi-Agent Systems (1998), pp. 333-340.

[Xie et al., 2002] X.-F. Xie, W.-J. Zhang, and Z.-L. Yang: "Hybrid Particle Swarm Optimizer with Mass Extinction". In Proc. of Int. Conf. on Communication, Circuits and Systems (ACCCAS), Chengdhu, China (2002), pp. 1170-1173.

[Yang et al., 1997] J. Yang, K. Karlapalem, and Q. Li: "Algorithm for Materialized View Design in Data Warehouse Environment". In M. Jarke, M.J. Carey, K.R. Dittrich, et al. (eds.). Proceedings of the 23rd International Conference on Very Large Databases, Athens, Greece, August 25-29, 1997, pp. 136-145.

[Yoshida et. al., 2000] Hirotaka Yoshida, Kenichi Kawata, Yoshikazu Fukuyama, Shinichi Takayama, and Yosuke Nakanishi: "A Particle Swarm Optimization for Reactive Power and Voltage

Control Considering Voltage Security Assessment". IEEE Transaction on Power Systems, Vol. 15, N.º 4, November 2000, pp. 1232-1239.

[Yu et al., 2004] J.X. Yu, C.-H. Choi, G. Gou, and H. Lu: "Selecting Views with Maintenance Cost Constraints: Issues, Heuristics and Performance". Journal of Research and Practice in Information Technology, Vol. 36, No. 2, May 2004.

[Zhang et al., 1999] C. Zhang, X. Yao, and J. Yang: "Evolving Materialized Views in Data Warehouses". Proceedings of the IEEE Congress on Evolutionary Computation (CEC'99), Washington D.C., USA, 1999, pp. 823-829.

[Zhang et al., 2001] C. Zhang, X. Yao, and J. Yang: "An Evolutionary Approach to Materialized Views Selection in a Data Warehouse Environment". IEEE Trans. on Systems, Man and Cybernetics, Part C, Vol. 31, N.º 3, Setembro 2001.

[Zhao et al., 1997] Y. Zhao, P. Desphande, and J.F. Naughton: "An Array-Based Algorithm for Simultaneous Multidimensional Aggregates". Proceedings of ACM-SIGMOD, 1997.

[Zhuge et al., 1995] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom: "View Maintenance in a Warehousing Environment". In: Proceeding of the ACM SIGMOD 1995 International Conference on Management of Data, San Jose, CA, 23-25 de Maio, 1995.

Referências WWW

[1] http://www.kurzweillai.net/articles/art0134.html.
2001-03-07. Consultado em 24 de Junho, 2006. Artigo onde Kurzweil projecta que a continuação da evolução predita pela lei de Moore, em 2019, implicará em transístores com apenas alguns átomos de espessura, chegando-se ao limite físico da tecnologia. Conjectura que é provável que algum novo tipo de tecnologia substituirá a tecnologia actual de circuitos integrados, permanecendo válida a lei de Morre para lá de 2020.

[2] http://news.com.com/New+life+for+Mooress+Law/2009-1006_3-5672485.html?tag=nl "New life for Moore's Law", por Michael Writer, CNET News.com, 19 de Abril, 2006. Artigo que mostra os limites físicos e problemas que se colocam com a utilização da tecnologia actual de fabrico dos semicondutores e quais as perspectivas que se abrem para que a lei de Moore permaneça válida.

[3] http://ciberduvidas.sapo.pt/php/resposta.php?id=2018 Portal onde podem ser colocadas dívidas relativas a significado de palavras.

[4] http://www.portaldigitro.com.br/glossario_digitro.php Portal com glossário tecnológico.

[5] http://wikipedia.org Enciclopédia on-line, com conteúdos de acesso livre e escrito de forma colaborativa por pessoas de todo o mundo. Em cada artigo há hiperligações para outros o que permite um rápido conhecimento de todos os assuntos relacionados com tema em pesquisa. Pretende-se que o conteúdo da Wikipedia seja factual, notável, verificável através de fontes externas, e apresentado de forma neutral, com citações a fontes externas. Um conjunto de políticas e guias de orientação aplicam-se para a prossecução deste desiderato.

[6] http://www-db.stanford.edu/warehousing/warehouse.html
O projecto *WareHouse Information Prototype at Stanford* (WHIPS) teve como objectivo primordial analisar a criação e a manutenção de SDWs e desenvolver algoritmos e ferramentas que assegurem estas actividades. Aqui encontra-se toda a documentação disponível sobre este projecto.

[7] http://www.tpc.org Portal da Transaction Processing Performance Council (TPC). A TPC é uma empresa sem fins lucrativos fundada para definir benchmarks relativos a processamento e bases de dados e a disseminar dados de desempenho TPC objectivos e verificáveis para a indústria. Neste site encontram-se recomendações acerca das bases de dados e forma de efectuar os testes que são o padrão que permite depois avaliar comparativamente os diversos servidores ou bases de dados. O site disponibiliza também a lista actualizada comparativa dos servidores e bases de dados, relativa aos diversos benchmarks definidos, que avaliam um servidor ao executar: 1) o processamento transaccional típico sobre uma base de dados (TPC-C), centrado então sobre as actividades principais (transacções) num ambiente de encomendas/vendas; 2) o processamento típico num sistema de suporte à decisão (TPC-H), envolvendo grandes volumes de dados, execução de interrogações com alto grau de complexidade, dando resposta a questões críticas de negócio; 3) as aplicações e serviços Web simulando as actividades típicas de uma aplicação negócio-a-negócio (B2B) num servidor de aplicações transaccional a operar num ambiente 24x7.

[8] http://citeseer.ist.psu.edu Biblioteca digital e motor de busca com centenas de milhares de artigos científicos disponíveis para consulta, incluindo referências e outras ligações a fontes de metadados como a DBLP e o portal ACM. O objectivo da CiteSeer é melhorar a disseminação e acesso a literatura científica e académica. Trata-se de um serviço que pode ser acedido por todos, sendo considerado como fazendo parte do movimento de acesso aberto que tenta alterar a forma de publicação de artigos científicos e académicos de forma a permitir um maior acesso à literatura científica. Muitos dos artigos referidos nesta dissertação estão aqui disponíveis, pelo que este *link* não foi referenciado explicitamente.

[9] http://www.dw-institute.com O Data Warehousing Institute (TDWI) é dedicado a ajudar as organizações a melhorar a sua compreensão e uso da inteligência para o negócio através da educação dos agentes de decisão e profissionais de SI no desenvolvimento adequado de estratégias e tecnologias de *data warehousing*. Promove a partilha de informação acerca das

melhores práticas e lições recolhidas do dia-a-dia. Organiza conferências internacionais e cursos na área do *data warehousing*, além de publicar o Journal of Data Warehousing e muitas outras iniciativas, tal como o muito popular Flash Report. Também promove a edição de livros, colige casos de estudo e *white papers*, empreende pesquisas e suporta programas de investigação. Disponibiliza no seu site muita informação e *links* sobre a temática do *data warehousing*.

[10] http://ieeexplore.ieee.org/Xplore/dynhome.jsp Trata-se de um portal de acesso a literatura técnica acerca de engenharia electrotécnica, computação e electrónica. Disponíveis artigos de revistas e proceedings (um pouco abaixo dos 1.5 milhões). Muitos artigos são de acesso geral, outros de acesso restrito. A biblioteca de conhecimento on-line, b-on [12], que a maioria das instituições do ensino superior português subscrevem, permite o acesso aos artigos ou outros conteúdos sem restrições. Muitos dos artigos referenciados neste trabalho estão disponíveis neste portal, não sendo assim referenciado explicitamente.

[11] http://portal.acm.org/portal.cfm É um portal de acesso a artigos publicados pela ACM (todos os artigos publicados pela ACM, incluídos em 50 anos de arquivos. Também permite aceder a livros, artigos, proceedings, teses de doutoramento de mestrado e relatórios técnicos de outras proveniências, perfazendo um total de quase 1 milhão de itens. Permite aceder a resumos a qualquer utilizador, mas a visualização do conteúdo é de acesso restrito. A b-on permite o acesso só a zonas limitadas dos recursos do portal. São aqui também disponibilizados muitos dos artigos referenciados nesta dissertação.

[12] http://b-on.pt É uma Biblioteca do Conhecimento Online que reúne as principais editoras de revistas científicas internacionais de modo a oferecer um conjunto vasto de artigos científicos disponíveis on-line, abrangendo a maior parte das áreas científicas, e estimular as condições de acesso universal ao saber por parte da comunidade científica e académica, procurando gerar economias de escala e promovendo as condições de universalidade de acesso à produção científica. A b-on permitiu o acesso, em 2004, a mais de 3500 publicações electrónicas de seis editoras de referência internacional, nas principais áreas de investigação científica e académica. Pretende-se com a iniciativa criar condições para alavancar as condições de acesso, utilização e difusão desse conhecimento, esperando-se venha a ser um grande contributo para aumentar a produção científica, a inovação e, por consequência, o desenvolvimento económico em Portugal.

Anexos

Anexo 1: Caracterização Multidimensional Tabular das Propostas de Optimização para Sistemas OLAP

Objectivo	Lógica Selecção ou Técnicas	Tempo (Inércia)	Constrangi-mento	Espaço	Tipo Modelo	Abordagem (forma)	Artigo Referência	Ano
Min CI	Greedy "BPUS"	Estática	E	C	L	Mat.Vistas	Harinarayan et al. 1996 "Implementing Data Cubes Efficiently"	1996
Min CI+CM	Greedy Referencial teórico	Estática	E	C	L	Mat.Vistas	Gupta 1997 "Selection of Views to Materialize in a Data Warehouse"	1997
Min CI+CM	MDred-lattice Construction + Heuristic Reduction	Estática	E+T+ carga	C	L	Mat.Vistas	Baralis et al. "Materialized View Selection in a Multidimensional Database"	1997
Min CI	Greedy	Estática	E	C	L	Mat. Vistas e Índices	Gupta, H. et al. "Index Selection for OLAP"	1997
Min CI+CM	Optimiz. de Interr. Múltiplas, com referencial baseado em Planos de Processamento de Vistas Múltiplas	Estática	E+T, atendendo ao perfil de carga e frequência de actualização.	C	L	Mat.Vistas	Yang, J. et al. "Algorithms for Materialized View Design in Data Warehousing Environment"	1997
Min CI	Greedy, "Pick By Size"(PBS) Várias ordens de grandeza mais rápido do que o BPUS	Estática	E	C	L	Mat.Vistas, utilizando heurística que selecciona as vistas atendendo ao seu tamanho que, sob certas condições, tem uma complexidade O(n log n)	Shukla, A. et. al. "Materialized View Selection for Multidimensional Datasets"	1998
Min CI	Greedy de Árvore Invertida e A*	Estática	E+T	C	L	Benefício mínimo de 0.63 do óptimo. Complexidade exponencial com n, inaplicável	Gupta, H. and Mumick, I.S. "Selection of Views to Materialize under a	1999

Objectivo	Lógica Selecção ou Técnicas	Tempo (Inércia)	Constrangi-mento	Espaço	Tipo Modelo	Abordagem (forma)	Artigo Referência	Ano
						para lattice com muitos nós.	Maintenance-Time Constraint"	
Min CI	Greedy. CMP (cube Merging and Pruning) e variantes.	Estática	E+T	C	L	Permite a concepção do DW levando logo em conta o conhecimento prévio das consultas requeridas.	Cheung, D. et al. "Requirement-Based Data Cube Schema Design"	1999
Min CI	Greedy de duas fases e Greedy Integrado	Estática	E+T	C	L	Selecção de vistas que asseguram o mínimo de custo de resposta a interrogações atendendo a um constrangimento temporal de manutenção	Liang, W. et al. "Materialized View Selection Under the Maintenance Cost Constraint"	2001
Min CI+CM	Genético, com função de penalidade para castigar soluções impraticáveis	Estática	E	C	L	Algoritmo genético para optimizar o conjunto de vistas materializadas que minimizem os custos de interrogação e manutenção	Horng, J.T. et al. "Materialized View Selection Using Genetic Algorithms in a Data Warehouse"	1999
Min CI+CM	Genético	Estática	-	C	L	Algoritmo genético para optimizar a selecção de vistas que minimizam o custo relativo ao plano de processamento múltiplo das interrogações I	Zhang, C. et al. "Evolving Materialized Views in Data Warehouses"	1999
Min CI+CM	Algoritmos incrementais e híbridos	Estática	E	C	L	Two-phase algorithm, r-greedy algorithm and search space pruning	Theodoratos, D. And Sellis, T. "Incremental Design of a Data Warehouse".	2000
Min Ci+CM	Híbrido (heurístico e genético), Um nível (Multiple View Processing Plan), outro nível um	Estática	E	C	L	Algoritmo evolucionário híbrido para resolução de três problemas relacionados: a optimização das interrogações, a escolha do melhor plano de	Zhang, C. et al. "An Evolutionary Approach to Materialized Views Selection in a Data Warehouse Environment"	2001

Objectivo	Lógica Selecção ou Técnicas	Tempo (Inércia)	Constrangi-mento	Espaço	Tipo Modelo	Abordagem (forma)	Artigo Referência	Ano
	algoritmo genético.					processamento e selecção das vistas mais adequadas para um dado plano de processamento.		
Min CI+CM	Pesquisa aleatória (Pesquisa Iterativa, por têmpera simulada e optimização em duas fases)	Estática / Dinâmica	E+T	C / D	L	Mat. Vistas ou Política de admissão e remoção de cache	Kalnis, P. et al. "View Selection Using Randomized Search"	2002
Min CI+CM	Algoritmo Genético com reparação (algoritmo greedy invertido para corrigir os genes dos indivíduos em falta	Estática	E	C	L	Mat. Vistas	Lin, W.-Y. and Kuo, I-C. "A Genetic Selection Algorithm for OLAP Data Cubes"	2004
Min CI (considerando custos de comunicação e processamento por nó)	Greedy (Algoritmo Node Set Greedy)	Estática	E	D	L	Mat. Vistas em DW distribuídos	Bauer, A. and Lehner, W. "On Solving the View Selection Problem in Distributed Data Warehouse Architectures"	2003
Min CI	Subst. de cache (LNR-C) Admissão em cache (LNC-A). Usam "Métrica de lucro"	Dinâmica	E	D	L	Caches e controlo de cache	Scheuermann, P. e tal. "WATCHMAN: A Data Warehouse Intelligent Cache Manager"	1996
Min CI	Os mesmos que em Watchman	Dinâmica	E	D	L	Caches dividindo o cubo em pedaços (chunks) de forma a conseguir um armazenamento mais eficiente e igualmente eficiente rapidez de busca.	Desphande, P. e tal. "Caching Multidimensional Queries Using Chunks"	1998
Min CI	Algoritmo de Divisão	Dinâmica	E+T	D	L	Caches	Shim, J. et al. "Dynamic	1999

Objectivo	Lógica Selecção ou Técnicas	Tempo (Inércia)	Constrangi-mento	Espaço	Tipo Modelo	Abordagem (forma)	Artigo Referência	Ano
	da Consulta que permite encontrar resultados armazenados que possam incluir a consulta canónica colocada.					A política de gestão da cache é baseada nos benefícios atendendo à frequência de utilização, custo de actualização e tamanho.	Caching of Query Results for Decision Support Systems"	
Min CI	Métrica Smaller Penalty First (SPF)	Dinâmica	E+T	D	L	Caches com índices R-trees Utilizar fragmentos de consultas para resposta a novas consultas, utilizando uma arquitectura que inclui um índice de directório que permite saber rapidamente da disponibilidade de uma consulta ser respondido pela cache.	Kotidis, Y., and Roussopoulos, N. "Dynamat: A Dynamic View Management System for Data Warehouses"	1999
Min CI	Principio Patch-Working e Estratégia de substituição baseada na densidade de referência de cada objecto multidimensional, grau de relacionamento com a última consulta, custos de reconstrução, derivando daí o chamado benefício absoluto.	Dinâmica	E	D	L	Caches com patch-working	Albrecht, J. et al. "Management of Multidimensional Aggregates for Efficient Online Analytical Processing"	1999
Min CI	Algoritmo greedy para decidir os	Dinâmica	E	D	L	Caches. Exchequer. Um sistema automático de caches de	Roy, P. et al. "Don't Trash your Intermediate Results,	2000

Objectivo	Lógica Selecção ou Técnicas	Tempo (Inércia)	Constrangi-mento	Espaço	Tipo Modelo	Abordagem (forma)	Artigo Referência	Ano
	resultados a manter em cache.					consultas que assegura um elevado acoplamento entre o optimizador e o sistema de cache que permite que as decisões sejam consistentes.	Cache'em"	
Min CI	Três políticas de cache de acordo com as diversas configurações da rede	Dinâmica	E	DD	L	Rede de caches OLAP distribuídas um gestor de política de admissão e de decisão de onde fazer a resposta a uma consulta que pode ser centralizado, distribuído ou um balancear entre os dois	Kalnis, P. and Papadias, D. "Proxy-Server Architectures for OLAP"	2001
Min CI	Algoritmo LeastBenefitFirst com três políticas: Isolated Caching Policy, Hit Aware caching Policy e Voluntary Caching	Dinâmica	E	DD	L	OLAP em clientes numa arquitectura P2P com repositórios organizados em chunks	Kalnis et al. 2002 "An Adaptive Peer-to-Peer Network for Distributed caching of OLAP Results"	2002
Min CI	Políticas: Lowest-Usability-First-Past e Lowest-Usability-First-Future e sua combinação. Política que considera não só os custos de recalculo, tamanho, mas a utilidade passada e futura dos resultados das consultas.	Dinâmica / Pró-activa	E	D	L	Não assume apenas uma perspectiva dinâmica, mas na política de admissão e substituição de cache procura explorar a utilidade futura das consultas armazenadas. Propõe algoritmo lowest-usability-first-Future para gerir a política de admissão e substituição de cache.	Park, C.S. et al. "Usability-based Caching of Query Results in OLAP Systems"	2003
Min CI	IA: Base de conhecimentos, Motor de Inferência,	Pró-activa	E Oportunidade temporal para	D	L	Permitir a restruturação do cubos tendo em conta a sua utilidade futura	Belo, O. "Putting Intelligent Personal Assistants Working on	2000

Objectivo	Lógica Selecção ou Técnicas	Tempo (Inércia)	Constrangi-mento	Espaço	Tipo Modelo	Abordagem (forma)	Artigo Referência	Ano
	Regras		efectuar a restruturação				Dynamic Hypercube Views Updating"	
Min CI	Algoritmo de Predição PROMISE que permite conhecer a utilidade futura de consultas e efectuar eventual prefetch	Pró-activa	Tempo que medeia entre consultas numa sessão	D	L	Cache em Memória com Geração de subcubos on-the-fly ou por pre-fetch, baseando a predição num modelo Markov construído explorando a semântica das interrogações.	Sapia, C. "PROMISE - Modelling and Predicting User Query Behavior in Online Analytical Processing Environments"	2000

Anexo 2: Caracterização Tridimensional das Soluções Propostas para o Problema de Selecção de Cubos.