Universidade do Minho
Escola de Engenharia

Luís Filipe Pacheco Florêncio

A SearchCol Algorithm for the Unrelated Parallel Machine Scheduling Problem with Job Splitting

outubro de 2013

A SearchCol Algorithm for the Unrelated Parallel Machine Scheduling Problem with Job Splitting
Luís Filipe Pacheco Florêncio
UMinho | 2013

Universidade do Minho
Escola de Engenharia

Luís Filipe Pacheco Florêncio

A SearchCol Algorithm for the Unrelated Parallel Machine Scheduling Problem with Job Splitting

Dissertação de Mestrado
Engenharia Industrial

Trabalho efetuado sob a orientação de
Doutora Carina Maria Oliveira Pimentel
Doutor Filipe Pereira Pinto da Cunha e Alvelos

outubro de 2013

DECLARAÇÃO

Nome:

Luís Filipe Pacheco Florêncio

Endereço electrónico: luisfpflorencio@gmail.com Telefone: +351 962 663 565

Número do Bilhete de Identidade: 12553532 ZZ4

Título dissertação ☒ / tese ☐

A SearchCol Algorithm for the Unrelated Parallel Machine Scheduling Problem with Job Splitting

Orientador(es):

Doutora Carina Maria Oliveira Pimentel

Doutor Filipe Pereira Pinto da Cunha e Alvelos

_____ Ano de conclusão: 2013

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

Mestrado em Engenharia Industrial

Universidade do Minho, ____ / ____ / _____

Assinatura: _____

This work was partially funded by the FEDER through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, project ref. PTDC/EIA-EIA/100645/2008.

Acknowledgements

This dissertation is the accomplishment of an objective drawn long before starting the MSc and would not have been possible without a group of people I deeply value.

I would like to start by thanking Professor Carina Pimentel and Professor Filipe Alvelos for the opportunity to participate in the project.

To Professor Carina I must thank the constant support and incentive, the teaching, the suggestions and the long hours dedicated, helping overcome the difficulties of starting, executing and finishing this work. Her experience and knowledge were very important.

To Professor Filipe I would like to thank for the availability and support. I also want to pay tribute to him: his Operations Research lectures were especially memorable and brought me here. His knowledge and enthusiasm for this field are an example to me.

To my friends and colleagues who accompanied me throughout this year. To the lunch group for the enthusiastic discussions and some longer coffees: Pedro, Cátia, Cristina, Jorge. To the MSc classmates, particularly João and Sérgio.

Agradeço à minha família que desde início me incentivou e apoiou a realizar este projecto. Os meus pais, o meu irmão, Dona Guida e Seu Jorge, e o pequenito Gonçalo que cresceu ao longo deste trabalho e muito me animou com as suas brincadeiras e gargalhadas.

Finally, I wish to end by thanking *à minha Joana.* For everything, absolutely everything. The companionship, affection, understanding, encouragement, support, help, friendship and our love.

Sem ti era possível, mas não teria tanta piada.

Abstract

In this dissertation, the unrelated parallel machine scheduling problem with job splitting and sequence independent setup times is addressed, implementing a method to solve it in a recently proposed framework, SearchCol, short for 'Metaheuristic search by Column Generation'.

The study of scheduling problems is of high relevance due to its real-world application in multiple fields, as documented in its vast literature, and also due to its high complexity derived from the diverse environments, variables, restrictions and the combinations of these in different systems.

The problem consists in finding a scheduling plan for a set of independent jobs on a set of unrelated parallel machines, considering jobs and machines release dates, sequence independent setup times and the job splitting property, with due date related objectives. The introduction of setup times and job splitting properties in unrelated environments has not been extensively studied, even though its use can play an important role in scheduling.

A mixed integer programming model is developed featuring the aforementioned properties, which is then decomposed by machine using the Dantzig-Wolfe decomposition. To solve the decomposed model a hybrid approach entitled SearchCol is applied, which results from the interaction between column generation and metaheuristics.

Problem specific heuristics to use in the column generation component of the SearchCol are also developed and diverse alternatives within the global algorithm are tested. A problem specific algorithm for one of the main SearchCol components is also suggested.

To evaluate the effectiveness of the model and the proposed algorithms, computational tests are performed and their solutions analysed for a set of test instances.

Resumo

O trabalho que se apresenta nesta dissertação, aborda o problema de escalonamento em máquinas paralelas não relacionadas com dimensionamento de lotes e tempos de preparação independentes da sequência, recorrendo a uma ferramenta recentemente proposta, designada por SearchCol, abreviatura de '*Metaheuristic Search by Column Generation*'.

O estudo de problemas de escalonamento revela-se de grande importância devido à sua aplicação em diferentes áreas, documentado na sua vasta literatura, e também devido à sua elevada complexidade decorrente das diversas configurações e tipos de máquinas, variáveis e restrições, bem como as combinações destas nos diversos sistemas.

O problema consiste na determinação de um plano de produção para um conjunto de tarefas independentes em máquinas paralelas não relacionadas, considerando tempos de disponibilidade de tarefas e máquinas, tempos de preparação independentes da sequência e o dimensionamento de lotes. O estudo deste problema com incorporação de tempos de preparação e da propriedade de dimensionamento de lotes em máquinas paralelas não relacionadas não é comum na literatura, apesar de se revelar de extrema importância em problemas de escalonamento.

Um modelo de programação inteira mista é desenvolvido para o problema e é também efectuada uma decomposição por máquina através da decomposição de Dantzig-Wolfe. Para resolver o problema, estuda-se uma abordagem híbrida que consiste na interação entre a técnica de geração de colunas e metaheurísticas, de seu nome SearchCol.

São desenvolvidas heurísticas específicas para o problema, as quais são usadas na componente de geração de colunas do SearchCol, sendo testadas também diversas alternativas e ferramentas no contexto do algoritmo global. Além disso, um algoritmo específico para o problema é também sugerido, para incluir num dos principais componentes do SearchCol.

Para avaliar o desempenho e qualidade dos modelos e algoritmos propostos, são realizados testes computacionais e analisadas as suas soluções para um conjunto de instâncias de teste.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ATC Apparent Tardiness Cost

ATCS Apparent Tardiness Cost with Setup

B&B Branch & Bound

B&P Branch & Price

CG Column Generation

CO Combinatorial Optimization

DDW Due Date to Weight Ratio

DE Differential Evolution

EDD Earliest Due Date

ET Earliness-Tardiness

FR Fix and Relax

GA Genetic Algorithm

GRASP Greedy Randomized Adaptive Search Procedure

IP Integer Programming

JIT Just-In-Time

LP Linear Programming

LR Linear Relaxation

LWT Lowest Weighted Tardiness

MH Metaheuristic

MIP Mixed Integer Programming

MP Master Problem

NAJ Not All Jobs

NAM Not All Machines

OF Objective Function

PMS Parallel Machine Scheduling

RMP Restricted Master Problem

SA Simulated Annealing

SEA Self-Evolution Algorithm

SP Subproblem

TA Threshold-Accepting

TS Tabu Search

TSP Travelling Salesman Problem

UPMSP Unrelated Parallel Machine Scheduling Problem

UPMSPjs Unrelated Parallel Machine Scheduling Problem with job splitting

VNS Variable Neighbourhood Search

VRP Vehicle Routing Problem

WIP Work In Progress

Chapter 1

Introduction

In this dissertation the Unrelated Parallel Machine Scheduling Problem with job splitting (UPMSPjs) is approached and an application of the framework SearchCol is presented to solve the problem, along with problem specific implementations on the same framework.

Scheduling is the process of deciding the allocation in time and sequencing of jobs in available resources to satisfy demand, under certain constraints, in order to optimize a given and appropriate criterion.

The interest both of practitioners and researchers in studying scheduling problems exists for more than 40 years and its importance is well known among the literature, with different approaches and models developed. As industry's characteristics and demands evolved, new and different implementations on models have been proposed, with a huge variety of problem types and characteristics available today.

Scheduling plays a crucial role in today's enterprises, as appropriate timing of production is mandatory and has important financial impacts. Computational and theoretical developments have given the possibility to better accommodate the needs of industry and complex systems can now be modelled to provide better decision making. With these developments, several realistic features have been introduced around basic concepts of production's environment and other characteristics directly related to processing, setup, sequencing of jobs, lot sizing or job splitting, as well as a wide range of evaluation criteria.

This dissertation approaches the aforementioned problem with a two-fold aim. First, a new time-indexed formulation of the UPMSPjs is proposed, where jobs are assigned to

machines and a sequence-independent setup time is incurred whenever a machine switches jobs. The model features distinct due and release dates for jobs, distinct availability dates of machines and job splitting. Secondly, a new approach to the problem is presented, based on a machine scheduling decomposition of the compact model using the Dantzig-Wolfe decomposition [Dantzig and Wolfe, 1960]. The solution of its linear relaxation is obtained through Column Generation (CG) and an integer solution is determined through a hybrid exact-heuristic method, that performs a meta-heuristic search for the optimal solution on the set of generated columns during CG (SearchCol). Heuristics were also designed for both CG and perturbations phases of the overall algorithm.

SearchCol is a framework for obtaining approximate solutions to Mixed Integer Programming (MIP) and Combinatorial Optimization (CO) [Alvelos et al., 2013]. The approach relies on combining CG procedures and Metaheuristic (MH) search to obtain good quality solutions in efficient amount of time, with the CG providing subproblem solutions (as a search space) to be used in the search phase by the chosen MH. The SearchCol algorithm applied to machine scheduling exploits the combinatorial optimization structure of large problems by associating a Subproblem (SP) solution with a decision to assign jobs to machines.

The main contributions that, we believe, result from this work are stated in the following paragraphs.

A compact model featuring several and complex system characteristics was developed. Among these characteristics we emphasize the introduction of: setup times (independent of job's sequence); the property of job splitting, allowing a job to get its processing divided into more than one machine, which allows for a more agile scheduling; and the possibility to preserve an initial state for each machine, where job's setup can be transferred from a previous scheduling to an actual one. Also, to our best knowledge, not much work has been done for the unrelated parallel machine production environment, when combining the previous characteristics on a same problem.

A machine scheduling decomposition of the above mentioned model was developed using Dantzig-Wolfe decomposition [Dantzig and Wolfe, 1960], upon which the SearchCol algorithm was applied to solve the problem. With the decomposition model, the aim was to develop a faster solving approach which would allow us to obtain better, or at least equal, lower bounds (than the ones obtained with the compact model), provided by the Linear Relaxation (LR) of the Restricted Master Problem (RMP) obtained from Dantzig-Wolfe

decomposition.

The solution's evaluation method, common in both models, is based on the determination of the inventory over the planning horizon, which we believe it is an innovation in the conjuncture of the evaluation methods used for parallel machine scheduling problems. This is achieved through the Objective Function (OF), in which a penalty is considered for processing taking place both before or after the due dates of the jobs. Moreover, this penalty has a weight associated, which can be defined, accordingly, to the situation where the model is being applied to.

Some specific heuristics were devised for the decomposition model. Although SearchCol provides exact resolution of the SP, in the context of the problem the development of specific heuristics brought advantages, especially in the CG phase. Two different heuristics were developed to solve the SPs, designed around the problem's characteristics and using the dual information provided by the RMP.

Moreover, other specific heuristics were also created to build initial solutions, to include on the first RMP, aiming to speed up the CG process and to guarantee that feasible columns are present in the RMP before starting the CG process. These columns, representing machine scheduling of jobs, take into account one, or more, specific characteristics and constraints of scheduling jobs to machines in an unrelated parallel machines environment.

A third and important step in the SearchCol algorithm is the perturbation phase. Even though several general methods are implemented in the framework and tested in this work, a specific perturbator algorithm is suggested to implementation using problem specific characteristics (as previous heuristics) with three different variants.

The dissertation is organized as follows. In Chapter 2 the scheduling literature is reviewed stating the different problems and models in the scheduling category, with information on techniques used to solve the problems being studied. A short historical and existing literature's revision of hybrid methods to solve optimization problems is also presented.

Chapter 3 presents the scheduling problem being studied, framing it in the different classifications and properties used for the general problem. Moreover, the evaluation method being used in this work is introduced in detail. Lastly, an example to better demonstrate it is also provided.

In Chapter 4 both models are presented. First, the notation to be used along this work is introduced. The compact model is then presented, explaining the different notation

used and the model constraints. The decomposition model, deriving from the previous compact one, is also presented, defining its Master Problem (MP) and SP models, with an introduction to the technique being used to solve its Linear Relaxation (LR).

SearchCol framework is introduced in Chapter 5. This chapter is divided in four main sections: first an introduction to the framework and its overall algorithm, features and possibilities; then the CG within SearchCol is introduced as well as the developed heuristics to create the initial solutions and to solve the SP; the Search phase is then described as to the existing possibilities inside the SearchCol; also, the existing SearchCol Perturbations as well as a suggestion of specific perturbator types for the problem are detailed in the chapter's last section.

Computational tests are presented in Chapter 6. The implementations developed for the decomposition model are analysed within the use of the general SearchCol algorithm and its features. The results of the decomposition model are also compared against the results of the compact model, as SearchCol allows for user implementation of both compact and decomposition models using the CPLEX libraries for C++ [ILOG, 2010] embedded within the SearchCol++, the framework of SearchCol built in C++.

Finally, Chapter 7 presents the conclusions and suggests future improvements and directions of research.

Chapter 2

Literature Review

In this chapter scheduling notations and definitions are introduced based on the literature, introducing some review works and overall considerations on the scheduling problems. Existing research work approaching the different Parallel Machine Scheduling (PMS) environments are reviewed with a following section dedicated exclusively to literature for the Unrelated Parallel Machine Scheduling Problem (UPMSP) analysing their approach, if it is the case, and other similarities to our own problem. Furthermore, hybrid methods to solve MIP and CO problems are also introduced.

2.1 Scheduling

The study of scheduling problems goes back to the mid-1950s and since then, several works have been published on the subject [Allahverdi et al., 2008]. Johnson [1954]; Smith [1956]; Jackson [1955] are considered pioneers for their scientific and systematic analysis of scheduling problems [Yang, 1999], and first approaches to the PMS problems, according to Nait et al. [2006] appeared a few years later with the works of McNaughton [1959] and Hu [1961].

Scheduling is defined by Pinedo [2002] as:

> The allocation of jobs to machines and their sequencing, subject to given constraints, in order to optimize one or more performance criteria.

Yang [1999] defined production scheduling as:

> The allocation of production resources over time in order to satisfy a pre-set criterion.

Pinedo [2002] describes the importance of scheduling as a decision-making process in manufacturing systems, transportation, distribution and other types of services and industries, providing also theoretical aspects and applications of scheduling.

To better systematize the different problem variations Graham et al. [1979] first introduced a three field $\alpha/\beta/\gamma$ classification system, which was adopted by most of later works and adapted to new circumstances.

The three field classification system reflects various job, machine and scheduling characteristics. Let's assume n jobs J_j $(j = 1...n)$ have to be processed on m machines M_i $(i = 1...m)$. The α field specifies the machine environment, which is 1 for the single machine case and adopts the following notations for the multiple machines cases:

P - identical parallel machines;

Q - uniform parallel machines;

R - unrelated parallel machines;

O - open shop;

J - job shop;

F - flow shop.

For the shop environment, flexible variants of the last notations are used. In this work, the focus will be on the PMS environment problem and its literature, as the shop environment implies that machines do not function in parallel but are dedicated.

In the previous notations, for the α field, an extension is made using **m**, meaning the number of machines is assumed to be variable; otherwise, a positive integer represents a constant number of machines.

The β field alludes to processing constraints such as preemption, job splitting, release dates, setup information and can contain multiple entries or no entries at all.

The γ field refers to the evaluation criterion used, which can be a regular function of the completion times of the jobs or their due dates, such as:

C_{max} - Makespan

L_{max} - Maximum lateness

$\sum w_j C_j$ - Total weighted completion time

$\sum T_j$ - Total tardiness

$\sum w_j T_j$ - Total weighted tardiness

$\sum w_j U_j$ - Weighted number of tardy jobs

$\sum E_j + \sum T_j$ - Total Earliness-Tardiness (ET)

$\sum w_j E_j + \sum w_j T_j$ - Total weighted ET

For the case of a non-standard OF the γ field can be noted as $\gamma = X$.

This way, for example, in an unrelated PMS problem to minimize total tardiness, with a variable number of machines and release dates for jobs, one can use the following notation:

$$R_m/r_j/\sum T_j$$

In the literature of machine scheduling there has been few attempts at tackling the problem of optimizing the scheduling and splitting of jobs subject to release dates and sequence-independent setup times in an unrelated PMS environment. Most research on this problem has been done for the single machine case [Allahverdi et al., 2008; Zhu and Heady, 2000]. Shim and Kim [2008] also note the lack of research results when a job splitting property is applied to a PMS problem.

According to Nait et al. [2006] the difference between job splitting and preemption properties, is that in the case of preemption, different machines cannot process the same job simultaneously, whether with job splitting, jobs can be split into different lots and processed in different machines at the same time or not.

For both an overview of the state of the art of scheduling problems after 1999 and a historical perspective see Allahverdi et al. [2008] which follows previous works by Allahverdi et al. [1999] and by Potts and Kovalyov [2000]. A comprehensive survey is done on scheduling problems involving setup times or costs, classifying them according to the environment previously referred and to batching and non-batching considerations (a batch can be defined as a set of jobs to be processed in batches so setup times or costs are unique to the batch, instead of incurring a setup time/cost for each job). Potts and Kovalyov [2000] and Potts and Wassenhove [1992] reviewed scheduling, focusing on batching and lot-sizing decisions and proposing a general model and sub-models cases integrating batch and lot sizing.

Allahverdi et al. [2008] state:

> (...) there are tremendous savings when setup times/costs are used and explicitly incorporated in scheduling decisions in various real world industrial/service environments.

Yang [1999] also surveys scheduling research involving setup times. In this work, important definitions and classifications are summarized, involving job, class, sequence dependence and separability setup situations. The review paper from Cheng et al. [2000] also focus on research done on scheduling with setup times, although considering the flow shop scheduling environment only.

Pinedo [2002] offers an exhaustive study on the scheduling problem, approaching the deterministic and stochastic models and numerous variants in each one, providing several and important definitions and classifications, as well as formulations, examples and possible approaches to solve the problems.

Other works approach scheduling focusing on more specific details.

Zhu and Heady [2000] summarized the main restrictive assumptions used in the field, regarding due dates, penalty costs, setups and number of machines.

Unlu and Mason [2010] made a comparison in order to identify - for various types of objective functions and machine environments - promising MIP formulation paradigms based on the types of variables such as job completion time, assignment and positional, linear ordering, time indexed and network types. Keha et al. [2009] also tested and compared four different MIP formulations for the single machine scheduling problem to identify, based on computational results, the best formulation type for the various problems.

2.2 Parallel Machine Scheduling

A set of resources or machines which are able to execute the same tasks are defined as Parallel Machines. According to Pinedo [2002] it is not only important in practice (as it is a common situation in real world) but also theoretically, as it is a generalization of the single machine case and a special case of the flexible flow shop.

The PMS environment is defined by Unlu and Mason [2010] according to the speed of processing of the machines for the different jobs: identical machines operate at the same speed (identical machine environment: P_m); non-identical machines operate at different speeds but its speed/processing rate is consistent for all machines when processing different jobs (non-identical machine environment: Q_m); unrelated machines can process different jobs at different speeds from the others, meaning that even though machine M_a can have a better speed/processing than M_b for job J_j it does not mean it has necessarily a better rate for any other job, as speed/processing rate is machine and job dependent (unrelated machine environment: R_m). The unrelated PMS environment is, in fact, a generalization of the non-identical case as an unrelated set of parallel machines can include a set of non-identical machines [Pinedo, 2002]. Also, when considering an unrelated PMS environment certain machines can be defined to have processing times for some jobs close to infinite or with a large integer number if, in fact, they do not have the ability to process the referred jobs[1].

Although processing characteristics are different, it is useful to study and understand other PMS environments, especially the most prominent identical parallel machines case, as existing research work on PMS problems with job splitting properties is sparse. The

[1]See Logendran and Subur [2004] for a practical application. This procedure could be used in a practical approach, to understand the needs to produce outdoors or expand production capacity.

following paragraphs are dedicated to the literature of PMS where job splitting properties and setup considerations are applied, as well as other important and differentiating characteristics and approaches. Works approaching the Unrelated PMS environment are reviewed in the following section.

Most of works on problems with job splitting properties were done for the identical PMS case. Yalaoui and Chu [2003] considered the problem of identical PMS with job splitting and sequence dependent setup times to minimize maximum makespan within the set of all machine scheduling plans. They developed a two-phase heuristic. In the first phase, they approached it as a single machine problem, transforming it into a Travelling Salesman Problem (TSP) and assigning jobs to machines using Little's method [Little et al., 1963]. Then, they created a feasible schedule for each machine, with the previously assigned jobs, which is improved taking advantage of the problem's characteristics. This method was used by Nait et al. [2006] for the same problem, introducing a heuristic based on a Linear Programming (LP) formulation to improve the approach of Yalaoui and Chu [2003]. Xing and Zhang [2000] also studied the job splitting property on an identical PMS problem with independent setup times to minimize the makespan, discussing cases with splitting properties and analysing a heuristic for this problem by extrapolating preemption properties.

The identical PMS case with job splitting properties was also addressed by Shim and Kim [2008], Park et al. [2012], Sarıçiçek and Çelik [2011] and Kim et al. [2004] with the objective of minimizing total tardiness.

Shim and Kim [2008] considered the problem with independent setup times, the same due dates for all jobs and machines available from the beginning of the planning horizon, using the example of Printed Circuit Boards as an industry with these characteristics. They further developed a Branch & Bound (B&B) algorithm that directly assigns jobs or sub-jobs to machines at each iteration and builds partial schedules following a set of dominance rules. Shim and Kim [2008] stated the existence of very few research results on the parallel machine scheduling problem with job splitting property.

Kim et al. [2004] approached the problem with sequence independent setup times developing a heuristic that reschedules an initial schedule, by splitting jobs through rules to select jobs, sub-jobs and machines. The authors compared the proposed methods to a modified Apparent Tardiness Cost with Setup (ATCS) heuristic that supports job splitting (this modification was done because no algorithm for this problem's characteristics

existed).

Park et al. [2012] considered the problem with major/minor sequence dependent setup times creating a heuristic that accounts for the problem's properties (job splitting, setup dependency), embedding it in three existing algorithms and comparing them to the original ones. The proposed algorithm divides a job into batches and assigns them to machines so total setup times can be reduced.

Sarıçiçek and Çelik [2011] recognized the difficulty of solving large scale integer programs, and proposed both a Tabu Search (TS) and Simulated Annealing (SA) meta-heuristic for the problem with independent setup times and developing a MIP formulation with positional variables, finding that the SA approach significantly outperforms the TS in terms both of computational time and deviation from the optimal solution (in terms of medium and long setup times).

As the majority of the literature does not consider job splitting, it is also important to analyse existing work on the PMS without this property, considering both similar and different properties.

Kaplan and Rabadi [2011] presented a practical application of the identical PMS to the aerial refueling, considering ready times and due date-to-deadline windows to minimize total weighted tardiness. A MIP formulation is presented and a modified Apparent Tardiness Cost (ATC) method developed by taking ready times and due date-to-deadline time windows into account, comparing its results to the SA meta-heuristic.

A Greedy Randomized Adaptive Search Procedure (GRASP) approach with path relinking to minimize total tardiness was developed by Armentano and de Franca Filho [2007] for the (less common) uniform PMS problem with job sequence dependent setup times, performing computational tests and comparing their approach against a TS algorithm on benchmark instances developed in earlier work.

Chen and Powell [1999a,b] propose decomposition approaches to a general case PMS problem [Chen and Powell, 1999b] considering machines can be either identical, uniform or unrelated (denoting this class as PMAC) for a minimization additive OF; and a just-in-time formulation [Chen and Powell, 1999a] for the identical case to minimize total weighted earliness and tardiness.

Sourd [2005] handles the single machine scheduling problem within the framework of what the author calls the assignment model, developing a time-indexed formulation for

the ET problem with setup considerations and setting forth a set of dominance rules for the schedules and an adaptation of the B&B procedure is applied. The time-indexed formulation developed in this work for the processing structure of the model, has similar structure to the one developed in this dissertation and presented in Chapter 4.

The ET problem was also adressed by Kedad-Sidhoum et al. [2008] who focus on the creation of upper and lower bounds for the single machine problem extending existing considerations for the identical PMS, and providing a local search heuristic.

Dunstall and Wirth [2005b] presented heuristic methods for the identical PMS with family sequence-independent setup times to minimize total weighted completion times, evaluating them computationally against exact methods. Dunstall and Wirth [2005a] also approached the same problem comparing the performance of three different rules in a branching scheme embedded on a B&B algorithm for the weighted completion time that significantly improves previous results in terms of computational time. For a technical treatment of B&B algorithms for PMS, as well as an overview of dominance rules see also Azizoglu and Webster [2003].

Akker et al. [1999, 2006] approaches the PMS developing CG algorithms to minimize total weighted completion times [Akker et al., 1999] and a non-explicit OF to minimize some function of the type minimax, such as maximum lateness or maximum cost [Akker et al., 2006].

Less common OFs to evaluate performance are proposed by Joo and Kim [2012] and Crauwels et al. [2006]. Joo and Kim [2012] presented a MIP with a linear ordering formulation for the identical PMS problem with ready times, due times and sequence-dependent setup times to minimize the weighted sum of setup, delay and tardy times. Their main focus was to present and compare two alternative meta-heuristics from evolution theory: two versions of a general purpose Genetic Algorithm (GA) heuristic (one with a special character chromosome that separates jobs assigned to the same machines, and another with a dispatching rule that assigns jobs to machines according to the completion times of jobs) and a newly introduced heuristic, Self-Evolution Algorithm (SEA). They found that the latter produces significantly better results in terms of deviation from the optimal scheduling solution, besides exploring a larger solutions' space. Crauwels et al. [2006] proposed a model with sequence independent setup times for jobs not belonging to the same family that minimizes the number of overloaded periods on a set of identical parallel machines. In order to construct a feasible schedule, they solved a knapsack problem for each machine

that transfers jobs from overloaded to underloaded periods for each machine.

The OFs present in the reviewed literature have no resemblance to the one being studied in this work. Most works rely on evaluating scheduling plans through earliness and/or tardiness, makespan or other factors related to setup and completion times. In the survey of Pfund et al. [2004], the authors report that unrelated PMS environments remained relatively unstudied, noting that there were few solution approaches to minimize due date related functions, and making aware that research in this area should include the development of solution algorithms to minimize due date related criteria, especially when conflicting objectives need to be optimized.

The use of initial solutions is a common practice. Chen and Wu [2006] generated initial solutions for an UPMSP by allocating each job to its most efficient machine and sorting them by Earliest Due Date (EDD) rule; Chen [2009] used a modified ATCS to obtain initial schedules to improve further on; Kim et al. [2004] and Wang et al. [2013] created initial solutions based on the PSK heuristic [Koulamas, 1997]; Armentano and de Franca Filho [2007] constructed initial solutions as their first phase when using a GRASP method; Logendran et al. [2007] used four different methods to create initial solutions: EDD, Lowest Weighted Tardiness (LWT), Due Date to Weight Ratio (DDW) rules and a proposed hybrid critical ratio rule. Logendran and Subur [2004] also used four different methods to create initial solutions in a problem with job splitting properties: EDD, EDD with consideration for job splitting, least flexible job and machine, and a modified ATC.

As pointed out by Xing and Zhang [2000] and Zhu and Heady [2000], the NP-hardness of the problem of scheduling n jobs on m machines with distinct release dates for jobs and machines, and distinct due dates for jobs, implies that alternatives to exact approaches must be sought.

2.3 Unrelated Parallel Machine Scheduling

The following paragraphs will focus on research work for the unrelated parallel machines environment.

A survey of the literature focusing on the UPMSP without side conditions was done by Pfund et al. [2004]. The authors reviewed the several performance evaluation methods and compiled existing algorithms for the various OF.

Logendran and Subur [2004] studied the UPMSP, with job splitting and distinct release dates for jobs and machines, to minimize total weighted tardiness. The authors present a MIP model using assignment and positional decision variables. In this work, the splitting property considers a job can only be split in two parts to prevent higher Work In Progress (WIP), with a predetermined number of jobs to be split. Also, neither setup times or costs are explicit, assuming they are included in the processing times (non-separable setup times or costs). Though the authors study an unrelated case with job splitting, the presented model constraints force jobs to be processed in the same machine in case a splitting occurs. To solve the problem, different initial solutions are created based on classic heuristics, which are then used by a TS based heuristic to find a better solution, comparing then the initial solutions that provide better results after applying TS. The authors claim that the use of different strategies to create an initial solution makes no difference in generating better solutions values for larger dimensions' problems, though it influences the computational times. Logendran et al. [2007] studied a similar problem with a similar approach, considering six different TS algorithms and four different initial solution methods that act as seeds of the algorithms. This work does not consider the possibility to split jobs and has setup sequence dependent times properties, as well as distinct release dates, with the objective of minimizing total weighted tardiness.

Zhu and Heady [2000] developed a MIP for the ET case for the unrelated PMS problem with sequence dependent setups to provide optimal solutions for small scale problems regarding future research and validation on industrial-scale heuristics.

Shim and Kim [2006] also considered the problem of scheduling jobs on unrelated PMS to minimize total tardiness without a job splitting property and without setup considerations, using a B&B algorithm approach with several developed dominance rules.

Liaw et al. [2003] considered the problem of unrelated PMS to minimize the total weighted tardiness without setup considerations. They first created upper and lower bounds, through a two-phase heuristic and an assignment approach respectively, and use a B&B algorithm with dominance rules to eliminate unpromising partial solutions.

Chen and Wu [2006] presented an heuristic combining the Threshold-Accepting (TA) method with the TS method and designed improvement procedures to minimize total tardiness for an UPMSP with auxiliary equipment constraints. The effectiveness of this approach was compared with an ATCS procedure and a basic SA method, outperforming both and obtaining optimal solutions for small-sized problems. Chen [2009] combined the

SA method, ATCS and designed improvement procedures to minimize total tardiness for an UPMSP with setup times that are dependent both on job sequence and machine used.

Wang et al. [2013] modeled the PMS problem with job splitting, for both identical and unrelated cases, to minimize the makespan, approaching it through a hybrid Differential Evolution (DE) method and creating a new crossover and mutation method in the global search according to job splitting properties, as well as a specific local search method. The authors made no considerations or assumptions regarding setup times.

Vallada and Ruiz [2011] proposed a genetic algorithm approach for the UPMSP to minimize the maximum makespan of a schedule with sequence dependent setup times for both jobs and machines. They developed a MIP with positional variables and an inter-machine insertion local search rule that decreases the computational burden of analysing all candidate solutions by examining the neighbourhood between pairs of machines while searching for a solution.

Rocha et al. [2008] considered the problem of unrelated PMS, with sequence dependent setup times for both machines and jobs, in order to minimize the maximum makespan and the total weighted tardiness (both are added in the same OF). They developed a B&B algorithm and compared it to two existing MIP models that use positional variables. The authors derive upper bounds using a GRASP method and calculate lower bounds separately for each of the components of the OF at each node.

Kim et al. [2002] presented a SA approach for the UPMSP with job sequence dependent setup times to minimize total maximum tardiness. This SA approach uses six different techniques to rearrange jobs or items and was compared to a neighbourhood search method, outperforming it. A particularity in this work is the existence of indexed and already defined and divided work part of lots or jobs.

Lopes and Carvalho [2007] studied the unrelated PMS problem with sequence dependent setup times to minimize total weighted tardiness, developing a Branch & Price (B&P) algorithm and proposing a new column generation acceleration method reducing significantly the number of explored nodes.

Fanjul-Peyro and Ruiz [2012] approached the UPMSP under makespan minimization. The unrelated environment was also extended to two possible situations. First, a situation where Not All Machines (NAM) are desirable to process certain jobs, using only a subset of parallel machines in order to understand if production capacity needs to be increased.

Secondly, a situation where Not All Jobs (NAJ) are obliged to be processed. Two different MIP formulations are developed for each situation and three algorithms are developed for the NAM problem, combining them with CPLEX or between them and comparing the results with the ones obtained using only CPLEX [ILOG, 2010].

Lee et al. [2013] suggested a TS algorithm to solve the unrelated PMS problem with sequence and machine dependent setups to minimize total tardiness. The TS approach was compared to an existing SA algorithm and an iterated greedy algorithm. The proposed method outperformed significantly the SA values of solutions and number of optimal solutions, and gave quicker solutions than the iterated greedy solution without improving solution values.

Lin et al. [2011] approached the UPMSP using different heuristics and a GA comparing its results. In this work, neither setup times nor job splitting properties are considered. For each developed heuristic, a different performance evaluation criterion was used: minimizing maximum makespan, minimizing total weighted completion times and minimizing total weighted tardiness. Each developed heuristic was compared against the GA that outperformed the other heuristics. Rodriguez et al. [2013] also approached the unrelated environment without setup times and splitting properties to minimize the total weighted completion times but using an iterated greedy algorithm to solve large-scale size instances.

2.4 Hybrid Methods

The use of hybrid algorithms is of great importance in the optimization fields, where a large number of problems are difficult and complex to solve by exact methods. To contour these difficulties, hybrid methods are a serious and main alternative to solve these problems [Talbi, 2013].

Hybrid approaches can result from the combination of different types of optimization techniques. Among them, the hybridizations that are useful to study in this work result from combining decomposition techniques based on Mathematical Programming with MH.

Raidl [2006] classifies hybridization of MH using four properties: the techniques being hybridized, the level of hybridization (how the original identity of the algorithms is maintained), the order of execution (sequential, interleaved or parallel), and the control strategy

(integrative, if one algorithm is subordinate, or collaborative, if information is exchanged between them).

Using the classification system of Raidl [2006], SearchCol is classified in Alvelos et al. [2013] as:

> The kind of algorithms that are hybridized define a first differentiation criterion in that classification scheme. In SearchCol, at least two types of algorithms are combined: a linear programming algorithm for solving the RMP of CG and a (hybrid) MH for the search. A problem-specific algorithm for solving the subproblem is also usual. The second differentiation criterion is the strength of combination. In SearchCol, CG influences a MH and the reverse is also true. As the algorithms retain their own identities, there is a low level of hybridization. Note that this weak coupling is a consequence of the generality of the approach. The order of execution is interleaved and the combination is collaborative: the CG and the MH exchange information but none is strongly subordinated to the other.

A taxonomy of hybrid algorithms to provide common terminology and classifications is also proposed by Talbi [2013, 2009, 2002]. The book by Talbi [2009] offers a comprehensive background on design and implementation of MHs with useful information to help solve complex optimization problems using MH with diverse approaches and techniques. Gendreau and Potvin [2010] compiled the different concepts, implementations and applications in the optimization fields of a vast number of MHs being a good reference to researchers who start working in the optimization field and, especially, MHs.

Blum et al. [2011] reviewed the existing techniques using MH in combinatorial optimization categorizing the different hybridizations and providing examples and short literature reviews for each category combining MH with: (meta-)heuristics, constraint programming, tree search methods, problem relaxation and dynamic programming. The combination of MH with tree search methods and problem relaxation fall upon the scope of our work. Search tree methods contain both approximate algorithms such as (meta-)heuristics and complete techniques and are considered to be one of the most popular hybridization approaches. According to the authors, the use of a problem relaxation to enhance MH has turned into a popular approach in recent years, namely the use of linear relaxation in integer problems which has already led to successful algorithmic approaches. The authors

advise to consider important aspects before developing a hybrid approach regarding the optimization goal, the existence of possibility to improving existing results using MH approaches and/or exact techniques, and which type of hybridization is well-suited for the considered problem. This last consideration is still a barrier due to lack of existing general guidelines, though research with hybrid MH is still in its early days with publications appearing concerning hybrids.

Barnhart et al. [1998] surveyed the different B&P formulations and discussed the issues of implementing a computational B&P algorithm. B&P can be considered as a hybridization of B&B and CG methods which has seen useful applications in scheduling problems. The authors suggest several reasons to use B&P for huge integer programs, namely, its better performance when solving the LP relaxation (opposed to the compact formulation), the poor performance of compact formulations in presence of a symmetric structure, the possibility of adding important characteristics to the problem in the contextual setting of subproblems and the master problem, as well as being the only possibility to solve the problem. Though the decomposition may provide a better LP relaxation, it is wisely noted that the decomposition should not be applied to obtain faster and shorter solving times but just to improve the LP bound.

For an overview on decomposition methods, see also the works by Desaulniers et al. [2005], Desrosiers and Lübbecke [2005] and Wilhelm [2001].

Danna and Le Pape [2005] and Boschetti et al. [2010] proposed frameworks using the same combination as Searchcol. Danna and Le Pape [2005] combined B&P and local search in the nodes of the search tree to generate new columns and improve the incumbent solution. This approach was applied within a Vehicle Routing Problem (VRP) with time windows. Boschetti et al. [2010] presented three different decomposition techniques, such as Dantzig-Wolfe, Lagrangean and Benders, and derive a MH framework for each technique by proposing a general structure algorithm that can be applied to specific problems.

From the reviewed literature of PMS problems and hybrid methods, the work by Anghinolfi and Paolucci [2007] stands out as one of the few papers working on the two fields of research. The authors approach the identical PMS case with the objective of minimizing total tardiness, by combining TS, SA and Variable Neighbourhood Search (VNS) MHs. The hybrid MH borrows from the SA approach the idea that randomness in searching for solutions in a neighbourhood can improve the effectiveness of the algorithm, from TS the idea of obtaining through a set of rules a list of candidate solutions, and from VNS the

idea that varying the neighbourhood scope and structure during the search can improve the quality of the results, independently of the quality of the first solution.

Chapter 3

Unrelated Parallel Machine Scheduling Problem with Job Splitting

In this chapter, a detailed description of the Unrelated Parallel Machine Scheduling Problem with job splitting (UPMSPjs) is made presenting definitions for the problem with characterization of jobs and machines' environment, introducing and explaining the OF used. Moreover, an example is also presented to illustrate the problem.

3.1 Definition

In this work a variant of the UPMSP is studied, in which a set of jobs needs to be scheduled in a set of available unrelated parallel machines to meet a given demand, in order to minimize an OF of processing of jobs before and after its due dates. Any job can be processed in any machine at any available time, being possible to process the same job at the same time in one or more than one machine, as long as a setup time is incurred for each machine or the initial state configuration of the machine is being respected. Moreover, at any given time period, each machine can only process, at most, one job.

The time periods are considered to be discrete and limited to a set of periods, to be calculated depending on the problem data (as will be detailed in Chapter 6). For each

time period, four different machine states are possible: unavailable, idle, being setup or processing. When unavailable, it means the machine is not yet released from a previous schedule. The idle status refers to a period of time where the machine is available but no setup or processing of any given job is incurred.

Solving this problem will result in a set of machine schedules indicating what job to process and when it will be processed. A machine schedule can be seen as a set of scheduled periods with a status associated to each period. Also, if the status of a period in a machine corresponds to setup or processing, a job is also associated to it.

The job splitting property allows a job to be split in multiple lots to be allocated in any available machine at any period of time.

More detailed definitions of the problem are given in Subsections 3.1.1-3.1.3.

3.1.1 Machines' Characterization

A set M of m machines is considered, such that $M = \{M_1, M_2, ..., M_i, ..., M_m\}$.

All m machines belong to an unrelated parallel environment and each machine can only process one job at a time, as long as it has been previously prepared for it.

The main characteristic in the UPMSP when dealing with the machines' environment comes from its unrelated property, which refers to the ability of each machine to process a given job. In the unrelated case, no relation or proportion exists between any machine, with each machine having its own processing speed for each job (or amount of time needed to fully process a job).

The problem requires machines to be defined as to its set size (number of machines), its release date and its initial setup state or pre-programmed job.

3.1.2 Jobs' Characterization

A set J of j jobs is considered, such that $J = \{J_1, J_2, ..., J_j, ..., J_n\}$.

Each job must be processed to satisfy its demand. This demand is not explicitly given, being defined by the number of processing periods required to satisfy it. Being an unrelated machine's environment, this processing time differs from machine to machine.

A job to be processed, must have its corresponding machine previously prepared. The preparation of any given machine occurs by incurring defined setup times, whether in the actual scheduling plan or as the last preparation for the previous schedule. The introduction of setup times must respect availability of machines but not availability of certain job to be processed; thus, setup times can be scheduled to preceding periods of job's release but must be scheduled after machine's release.

The main distinguish feature of the UPMSP under study is its job splitting property which allows any job to be divided in different lots and processed in more than one machine, being possible to process it at the same time in different machines. As it is an unrelated environment, the sum of processing times in different machines does not guarantee a full processing of the referred job. Therefore, demand is satisfied when the sum of proportions of processing of a job in all machines is met. Furthermore, for each lot a setup time must be incurred to be able to process the job. The setup times are sequence independent so each job has a unique setup time, independently of what job precedes it or which machine will process it.

A due date is associated to each job, so that by the end of that period the job should be fully processed and ready, although it is possible to process it after this date, meaning it will result in tardiness. Also, this due date must be posterior to the job's release date.

Jobs are classified as to its importance by a given weight, with the biggest weights corresponding to the jobs with the utmost importance.

The problem requires jobs to be defined as to its set size (number of jobs), availability or release date, due date, weight, sequence independent setup time and processing times (one processing time for each machine).

3.1.3 Objective Function

The aim of the OF is to obtain a schedule that minimizes production occurring both before and after the job's due date. The OF will penalize early and tardy processing (related to the due date), with the possibility to define if the penalization should be bigger for earlier or tardier production, by using a convex combination of a parameter β. Also, within the set of processing periods before and after the due date, the more distant the processing is done from the due date the more it penalizes the solution's value.

The proposed OF reaches a compromise by combining the main idea behind Earliness-Tardiness (ET) models, the guarantee that processing happens accordingly to job's demand and without early or tardy unwanted production, as it can happen with a time indexed formulation, when using an OF that accounts only for beginning and/or conclusion dates.

Moreover, this evaluation method motivates processing to be done the closest possible to the due date, thus avoiding unnecessary WIP (caused, for example, by the job splitting property).

Figure 3.1 shows a scheduling example from which a calculation for the solution value can be obtained, using the developed OF.

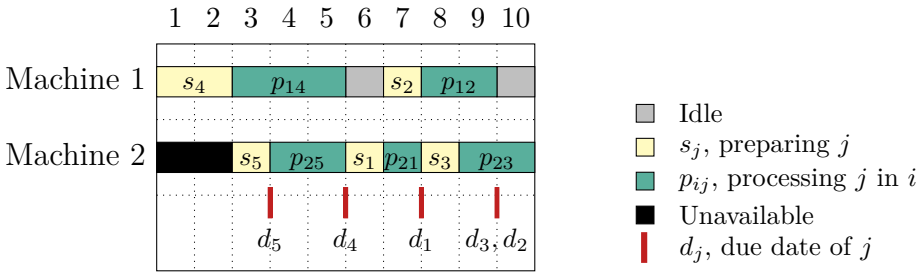

Figure 3.1: A scheduling example to calculate OF value

The example shows a set of 5 jobs with respective due dates below, scheduled in 2 machines accordingly to its setup and processing times and release dates. All jobs' weights are considered to be 1 (to simplify) and the β parameter 0.9. Using this value for β implies tardy processing has a weight of 0.9 and early processing a weight of 0.1.

Analyzing the job scheduling presented, job 5 is processed in tardy periods as its due date occurs very early in the set of periods. Job 1 has the best performance, as by having a processing time of one unit only, is able to process on its due date, so does not incur in any early or tardy penalization. Remaining jobs have mixed performances.

Considering the example and respective scheduling plan, its OF value can be calculated the following way:

$$Z = \underbrace{\underbrace{0.2 + 0.1 + 0}_{\text{job 4}} + \underbrace{0.1 + 0}_{\text{job 2}}}_{\text{machine 1}} + \underbrace{\underbrace{0.9 + 1.8}_{\text{job 5}} + \underbrace{0}_{\text{job 1}} + \underbrace{0 + 0.9}_{\text{job 3}}}_{\text{machine 2}} = 4.0$$

3.2 Example

In Figure 3.2 an example is shown representing a set of 4 machines and 20 jobs, its scheduling in time using some of the definitions and properties previously introduced.

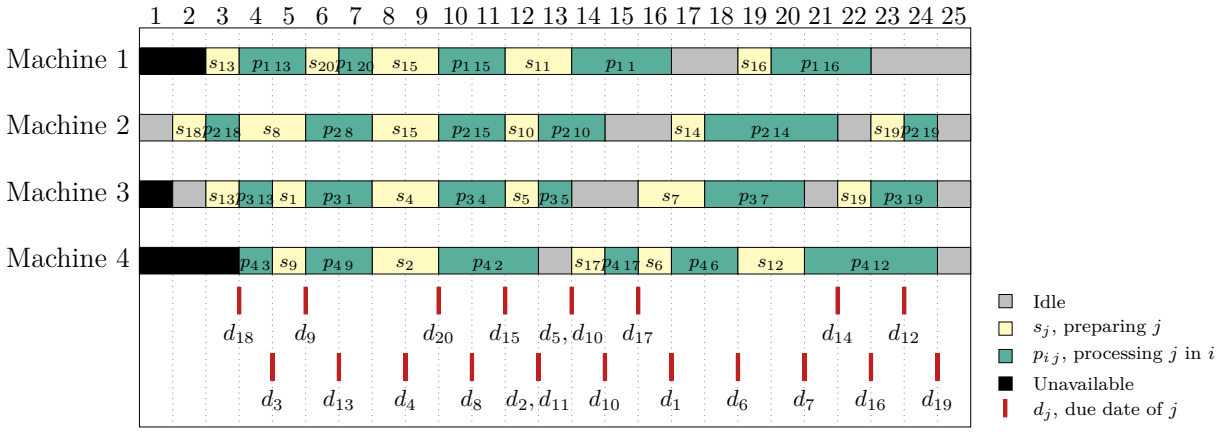

Figure 3.2: Scheduling example

The schedule exemplified is comprised of 25 periods of time, which does not represent the implementation developed in this work. This example shows a feasible solution for the UPMSPjs, with the job splitting property being applied in jobs 13, 15 and 19 to process different lots in two different machines, and repeating setup times. The advantages of splitting processing through different machines are visible in the example.

The due date of job 15 is in a congested interval of periods, so the split of processing between machine 1 and 2, frees previous time periods to process different jobs and reduces WIP, which is easily verified by calculating the cost of this processing with and without job splitting (considering the processing would occur in one machine and in the preceding periods). The weight of the job is represented by w_{15}.

$$Z_{15split} = \underbrace{\underbrace{(1+0}_{t_{10}+t_{11}}}_{\text{machine 1}} + \underbrace{\underbrace{1+0)}_{t_{10}+t_{11}}}_{\text{machine 2}} (1-\beta)\, w_{15} = 2\,(1-\beta)\, w_{15}$$

$$Z_{15 \text{ without split}} = \underbrace{(3+2+1+0)}_{t_8+t_9+t_{10}+t_{11} \text{ of machine 1}} \quad (1-\beta)\, w_{15} = 6\,(1-\beta)\, w_{15}$$

In this example, the splitting of processing results in a cost savings of $\frac{2}{3}$.

Also, in machine 4, the initial setup state of the machine is preserved, processing without incurred setup times by using the configuration of the previous schedule.

Chapter 4

Models

In this chapter the mathematical models for the UPMSPjs are presented. The chapter is divided in three sections. First the developed notation is introduced. In the following section the Compact Model is detailed, taking into consideration the definitions presented in the previous chapters and the notations developed in previous section. In the last section, the Machine Schedule Decomposition model will be presented, based on the Compact Model of Section 4.2, using the Dantzig and Wolfe [1960] decomposition technique.

4.1 Notation

In this section, the notation used in this work is introduced, organized by considered sets and parameters needed for the problem (introduced in the previous Chapter).

Considering the $\alpha/\beta/\gamma$ Graham's notation introduced in Chapter 2, the UPMSPjs is classified as:

$$R_m/r_j, q_i, s_j, d_j, split/X$$

The sets considered in this work are represented by:

J - Set of jobs, indexed by $j = 1...n$
T - Set of discrete, integer time periods, indexed by $t = 0...T_{max}$

M - Set of machines, indexed by $i = 1...m$

The parameters have the following notation:

p_{ij} - Processing time of job j in machine i, in time units

r_j - Release date of job j, the moment in time it becomes available for processing

q_i - Release date of machine i, the moment in time after which machine i can process jobs

d_j - Due date of job j

w_j - Priority or weight of job j

s_j - Sequence-independent setup time of job j

$z[i]$ - Vector indicating the programmed job for each machine, at the beginning of the scheduling horizon

β - Constant between 0 and 1

4.2 Compact Model

The proposed model to the UPMSP is a time indexed based one[2]. In this model, beside the processing decision variables, it's also taken into account setup decision variables and an extra decision variable associated with the first change in setup status of a machine. Processing and setup runs comprise integer multiples of the time periods and the decision variables assign either setup or processing decisions to the available time slots of the schedule.

The model's decision variables to be used are:

x_{ijt} - Boolean variable assigning integer time slots of the schedule to the production of job j in machine i at time t, defined as:

$$x_{ijt} = \begin{cases} 1 & \text{if job } j \text{ is assigned to machine } i \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

[2]In a time-indexed formulation, time is divided into a pre-set of identical periods of a unit length. Note that this unit length can be adjusted accordingly to the problem's data.

y_{ijt} - Boolean variable assigning integer time slots of the schedule to the setup of job j in machine i at time t, defined as:

$$y_{ijt} = \begin{cases} 1 & \text{if a setup for job } j \text{ is incurred in machine } i \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

e_{it} - Boolean variable accounting for the first change in the setup status of machine i at time t, defined as:

$$e_{it} = \begin{cases} 1 & \text{if the setup status of machine } i \text{ changes from } z[i] \text{ to } j \text{ at time } t \\ & \text{or in a previous period of time} \\ 0 & \text{otherwise} \end{cases}$$

The developed model, considering the previous developed notations and constraints is the following:

$$Min\, Z = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t>d_j}^{T_{max}} \beta(t-d_j)w_j x_{ijt} + \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{t\leq d_j}(1-\beta)(d_j-t)w_j x_{ijt} \tag{4.1}$$

Subject to:

$$\sum_{i=1}^{m}\sum_{t>max\{r_j,q_i\}}^{T_{max}} \frac{1}{p_{ij}} x_{ijt} \geq 1 \qquad \forall j \tag{4.2}$$

$$\sum_{k=t-s_j}^{t-1} y_{ijk} \geq (x_{ijt} - x_{ij(t-1)})s_j \qquad \forall i, \forall j : j \neq z[i], \forall t : t \geq 1 \tag{4.3}$$

$$s_{z[i]}(1-e_{i(t-1)}) + \sum_{k=t-s_{z[i]}}^{t-1} y_{iz[i]k} \geq (x_{iz[i]t} - x_{iz[i](t-1)})s_{z[i]} \qquad \forall i, \forall t : t \geq 1 \tag{4.4}$$

$$\sum_{j=1:j\neq z[i]}^{n}\sum_{k=0}^{t}(y_{ijk} + x_{ijk}) \leq te_{it} \qquad \forall i, \forall t : t \geq max\{q_i, 1\} \tag{4.5}$$

$$\sum_{j=1}^{n} x_{ijt} + \sum_{j=1}^{n} y_{ijt} \leq 1 \qquad \forall i, \forall t : t \geq 1 \tag{4.6}$$

$$x_{ijt} = 0 \qquad \forall i, \forall j, \forall t : t \leq max\{r_j, q_i\} \tag{4.7}$$

$$y_{ijt} = 0 \qquad \forall i, \forall j, \forall t : t \leq q_i \tag{4.8}$$

$$x_{ijt}, y_{ijt}, e_{it} \in \{0, 1\} \tag{4.9}$$

The OF is represented in this model by (4.1). The developed OF is composed of different characteristics which will be detailed below and it is first introduced in Subsection 3.1.3 of previous Chapter.

The processing of a given job is penalized according to how distant it occurs from the its due date. The distance in time between the processing and the due date takes two forms, depending whether it is before or after, with all processing for each job being summed, which corresponds, when considering a job j, to:

$$\sum_{t>d_j}^{T_{max}} (t - d_j)x_{ijt} + \sum_{t=1}^{t\leq d_j} (d_j - t)x_{ijt}$$

Weights introduced before (job's weight and β parameter) are then included. The parameter β is applied to both terms, so that the sum of the multiplier is equal to 1, for periods of a machine schedule with equal time distance to the due date of a given job j. Job's weight is equally multiplied to both parts of the equation.

$$\sum_{t>d_j}^{T_{max}} \beta(t - d_j)w_j x_{ijt} + \sum_{t=1}^{t\leq d_j} (1-\beta)(d_j - t)w_j x_{ijt}$$

If the objective is to minimize the processing after the due date, the β should be user defined in the interval $]0.5, 1]$. The higher the β, the bigger weight the tardier processing will have in the OF.

The OF is completed when considering the sum of all machines and jobs with the minimization of the following expression:

$$Min\, Z = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t>d_j}^{T_{max}} \beta(t - d_j)w_j x_{ijt} + \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{t\leq d_j} (1-\beta)(d_j - t)w_j x_{ijt}$$

The developed OF penalizes all periods with processing status, except when in the due date period. If parameter β is defined closer to 1, scheduling will only cause tardiness if there are not enough available periods before the due date, either because the due date is in the beginning of the schedule, or because periods before due date are already unavailable (preparing or processing a different job).

There are several constraint sets that need to be considered in this model. A set of con-

straints must be defined to guarantee the satisfaction of demand, considering that processing can be executed in any machine with different relations between the speed/processing and the total demand or needed processing. Constraints (4.2) simply state the sum of the processing variables for each job must be at least equal to its total processing time (each job is completely executed and demand is satisfied).

A second set of constraints in this model relates to setup considerations, guaranteeing not only the mandatory machine setup before any new job is processed, but also the preservation of an initial setup state inherited from the previous schedule. Constraints (4.3) ensure that a setup time is incurred whenever a machine starts processing a new job. Constraint set (4.4) has the same role as (4.3), but considers the case where the incoming job is the preprogrammed one, and allows for initial setup preservation through the change of status variable e_{it}. By (4.5), the setup status of a machine changes in time t if the incoming job is not the preprogrammed one (in which case a setup must be incurred before any production takes place).

A third set of constraints aims at limiting the status of a machine, if not idle or unavailable, to one of the two active possible states: being setup or processing. By (4.6), it is ensured that at any given time t, a machine is either processing, being setup for a job, or idle.

A set of constraints must also be considered to guarantee that the release dates are respected, so that machines cannot process or be setup before being available and jobs cannot be processed before being also available. Constraints (4.7) guarantees that no production takes place before the maximum between the release dates of job j and machine i. By (4.8) it is stated that setups cannot take place before machine i is available.

Finally, constraints (4.9) bound the variables of the problem.

The developed Compact Model presented in this section will be solved using the callable libraries of a general purpose solver, CPLEX ([ILOG, 2010]), through the SearchCol computational framework (to be presented in the Chapter 5).

4.3 Machine Schedule Decomposition Model

In this section, a Dantzig-Wolfe decomposition is applied to the Compact Model of Section 4.2, and a Master Problem (MP) and a set of smaller and independent problems are created - the Subproblems (SPs).

In this work, each SP solution will correspond to a machine scheduling, so each problem represents a single machine scheduling problem. To decompose the model two different sets of constraints must be considered: a set of constraints that link the different machines, and a set of constraints where the machine being considered is the same. Considering a matrix of the constraints, in order to apply the decomposition, a set of sub-matrices must be identified and regrouped as connected or coupled constraints (set of constraints that link the different machines) and another subset of the remaining sub-matrices that are not coupled are also identified.

Using the previously presented Compact Model, the set of constraints (4.2) is the only coupling sub-matrix identified, considering a sum of processing in the set of all machines; whereas, all the other constraints are not linking different machines. The first referred constraint will remain in the MP, and all the others (constraints (4.3) to (4.9)) will form the SPs.

The SPs being solved will result in different machine scheduling plans where not all jobs must be scheduled (no satisfaction demand must be guaranteed) and where allocation of processing has exactly the same procedure as in the compact model, indicating for that SP which job and when it shall be processed. Each SP has its own characteristics, as they depend on machine properties and its job processing times, resulting in a set of different problems.

For the MP new decision variables will be needed, to represent the extreme points generated by the SPs. The solution of the MP will represent a convex combination of these points, in order to guarantee that only one plan is chosen for each machine, with its MP constraints being respected.

All notation being used has already been introduced, except for the following set, decision variables and parameters.

A new set must be defined representing the total number of scheduling plans generated by the SPs:

H_i - Set of machine i scheduling plans, indexed by $h = 1...g_i$

The new decision variables to be used are the following:

λ_i^h - Weight of scheduling plan h of machine i, defined as:

$$\lambda_i^h = \begin{cases} 1 & \text{if scheduling plan } h \text{ of machine } i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

A new set of parameters to be used must be defined such that:

α_{ijt}^h - Boolean parameter assigning integer time slots of the scheduling plan h in machine i to job j, defined as:

$$\alpha_{ijt}^h = \begin{cases} 1 & \text{if job } j \text{ is processed in machine } i \text{ at time } t \text{ on scheduling plan } h \\ 0 & \text{otherwise} \end{cases}$$

It must be noted that the parameter α_{ijt}^h is directly related to the previously defined x_{ijt}, though this one is now used in the MP whereas the original one is being used by the SPs.

4.3.1 Master Problem

With all the notation developed in Subsections 4.1 and 4.3 the following MP can be defined:

$$\begin{aligned} Min \sum_{h=1}^{g_i} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{t>d_j}^{T_{max}} \left(\beta(t-d_j)w_j\alpha_{ijt}^h\right)\lambda_i^h \quad + \\ \sum_{h=1}^{g_i} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{t=1}^{t\leq d_j} \left((1-\beta)(d_j-t)w_j\alpha_{ijt}^h\right)\lambda_i^h \end{aligned} \tag{4.10}$$

Subject to:

$$\sum_{h=1}^{g_i} \lambda_i^h = 1 \qquad \forall i \qquad (\eta_i) \tag{4.11}$$

$$\sum_{h=1}^{g_i}\sum_{i=1}^{m}(\sum_{t>max\{r_j,q_i\}}^{T_{max}}\frac{1}{p_{ij}}\alpha_{ijt}^{h})\lambda_i^h \geq 1 \qquad \forall j \qquad (\Pi_j) \tag{4.12}$$

$$\lambda_i^h \in \{0,1\} \tag{4.13}$$

The OF in (4.10) follows the one used in the compact model, minimizing processing occurring distant from the due date of the jobs in the chosen scheduling plans. A new set of constraints is introduced. The set of constraints (4.11) are the convexity constraints of the model, that guarantee that a combination of the SPs is chosen. Constraints (4.12) derive from the compact model set of constraints that ensure processing and satisfaction's demand is met for all jobs (in all chosen plans for all machines). The last set of constraints (4.13) defines the decision variables domain.

When solving the Restricted Master Problem (RMP), a set of dual variables is obtained: Π_j from (4.12) giving information of whether it is attractive to process job j and from (4.11) the convexity constraint dual variable η_i.

4.3.2 Subproblem

Using remaining constraints and the dual variables provided by the MP, the following SPs is formed, for machine i:

$$\begin{aligned} Min\, Z^{SP_i} = & \sum_{j=1}^{n}\sum_{t>d_j \wedge t>max\{r_i,q_i\}}^{Tmax}\left(\beta(t-d_j)w_j - \frac{1}{p_{ij}}\Pi_j\right)x_{ijt} + \\ & \sum_{j=1}^{n}\sum_{t\leq d_j \wedge t>max\{r_i,q_i\}}^{Tmax}\left((1-\beta)(d_j-t)w_j - \frac{1}{p_{ij}}\Pi_j\right)x_{ijt} + \\ & \sum_{j=1}^{n}\sum_{t\leq d_j \wedge t\leq max\{r_i,q_i\}}^{Tmax}(1-\beta)(d_j-t)w_j x_{ijt} + \\ & \sum_{j=1}^{n}\sum_{t>d_j \wedge t\leq max\{r_i,q_i\}}^{Tmax}\beta(t-d_j)w_j x_{ijt} \; - \; \eta_i \end{aligned} \tag{4.14}$$

Subject to:

$$\sum_{k=t-s_j}^{t-1} y_{ijk} \geq (x_{ijt} - x_{ij(t-1)})s_j \qquad \forall i, \forall j : j \neq z[i], \forall t : t \geq 1 \tag{4.15}$$

$$s_{z[i]}(1 - e_{i(t-1)}) + \sum_{k=t-s_{z[i]}}^{t-1} y_{iz[i]k} \geq (x_{iz[i]t} - x_{iz[i](t-1)})s_{z[i]} \qquad \forall i, \forall t : t \geq 1 \tag{4.16}$$

$$\sum_{j=1:j\neq z[i]}^{n} \sum_{k=0}^{t} (y_{ijk} + x_{ijk}) \leq te_{it} \qquad \forall i, \forall t : t \geq max\{q_i, 1\} \tag{4.17}$$

$$\sum_{j=1}^{n} x_{ijt} + \sum_{j=1}^{n} y_{ijt} \leq 1 \qquad \forall i, \forall t : t \geq 1 \tag{4.18}$$

$$x_{ijt} = 0 \qquad \forall i, \forall j, \forall t : t \leq max\{r_j, q_i\} \tag{4.19}$$

$$y_{ijt} = 0 \qquad \forall i, \forall j, \forall t : t \leq q_i \tag{4.20}$$

$$x_{ijt}, y_{ijt}, e_{it} \in \{0, 1\} \tag{4.21}$$

In this formulation, each SP corresponds to a machine upon which the reduced cost of production plans are evaluated at each iteration of the algorithm.

The OF of the SP uses the dual information provided by the MP. This dual information indicates which jobs are attractive to produce in each SP and each iteration.

The set of constraints has the same meaning as in the case of the Compact model (see Section 4.2).

The SPs can be solved using any suitable method. In Chapter 5 will be presented a set of results from using three different approaches to solve the SPs: a general purpose solver and problem specific heuristics using two different visions of the SPs. The referred methods are used within the context of CG which is introduced in next Section.

4.3.3 Column Generation

The approach being used to solve the LR of this decomposition model is known as Column Generation (CG), typically used to solve large LP problems or to obtain good lower bounds for Integer Programming (IP) problems.

CG is an iterative process where the MP and SPs change information between them. When in the first iteration, the MP is solved using a reduced form version that contains only

a meaningful subset of all variables and through relaxation of the integrality constraints. This reduced form is called the first Restricted Master Problem (RMP), and solving it a dual optimal solution is obtained. The dual information resulting from the previous RMP, is then used by the SPs which are solved using the updated dual information. The solutions resulting from solving the SPs are then sent to the RMP until a new iteration results in a SP solution of non-negative value, meaning the current solution of the RMP is the optimal solution of the MP.

Solving the decomposition model using CG results in a linear solution, where one machine schedule may be composed of partial schedules. The proposed framework in this work is able to solve the integer problem (find an integer solution) through diverse methods.

These methods, as well as the whole CG process, its implementation and problem specific heuristics to solve the SPs and to start the CG, are presented in Chapter 5.

Chapter 5

SearchCol

In this chapter, the SearchCol framework (short for 'Metaheuristic search by Column Generation') is introduced. Several aspects and possibilities of the SearchCol will be presented, as well as the overall *modus operandi* of the global algorithm. After the introduction of the framework, the chapter will be divided in sections corresponding to each main step of the SearchCol algorithm. In each step, behind the general considerations, the application of the algorithm to the UPMSPjs will also be presented, with special focus on the problem specific implementations developed.

5.1 Introduction

In this section, the basic ideas of the SearchCol will be introduced, showing also how its three main components interact between them. For a detailed introduction to the SearchCol framework, see Alvelos [2012] and Alvelos et al. [2013].

Searchcol is an optimization method combining two approaches: CG and MH. Within the SearchCol, CG is used to find linear solutions to the decomposition model, resulting from the compact one. As previously mentioned, during the CG process, several SPs are iteratively solved and its solutions inserted into the RMP (each SP solution of each iteration of the CG is associated with a column of the RMP). These solutions to the several SPs can be treated as components of the overall problem solution. The set of solutions provided by the CG acts as the search space for the chosen searcher (which can be a MH or an exact

solver), that will return an incumbent (and feasible) integer solution to the overall problem. SearchCol algorithm iterates by modifying, after the conducted search, the structure of the decomposition model, namely the constraints present in the RMP, which perturbates the CG and enables the creation of new problem solutions.

The SearchCol's UPMSPjs implementation followed the approach provided in Figure 5.1.

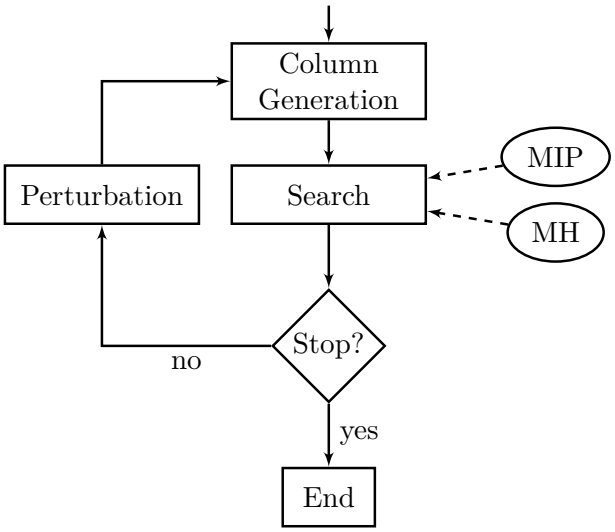

Figure 5.1: An overview of a SearchCol iteration

The SearchCol's global algorithm can be divided in three main steps or phases (upon which this work will focus). These steps can be identified as the CG, the Search and the Perturbation phases. In each one of these steps, several methods and possible problem specific implementations are available in the core SearchCol framework.

The algorithm starts by applying the CG using the CG heuristic detailed in Section 5.2. In this step the decomposition model is solved and an optimal LR solution to the problem is obtained, which also provides a lower bound to the MIP model. The CG phase is problem dependent as it demands user implementation of the model (RMP and SPs), and if desired, problem specific implementations to either build initial solutions or to heuristically solve the SP. Moreover, a specific algorithm can be implemented to solve the SPs more efficiently. On the other way, the interaction between the RMP and the SP is already implemented and can still be parametrized.

In the second phase of the algorithm, the set of previously generated solutions define a search space, and work as components of the overall solution [Alvelos et al., 2013].

This phase is problem independent as the several search tools of the MH search are fully implemented and can also be parametrized. Two different search algorithms were used in this work: VNS metaheuristic and MIP. The search phase and the used search components are presented in Section 5.3.

If SearchCol is parametrized to have more than one iteration, a new phase of the global algorithm is available. This phase, when running SearchCol in an iterative mode is important to avoid the algorithm to become redundant, by solving the same CG and obtaining the same solutions. In this third step a perturbation to the RMP that will be solved in the following CG is added. This perturbation is, in fact, a procedure that adds a set of new constraints to the RMP, so that promising SPs variables are forced to have a (binary) value, in order to generate new SP solutions and try to improve the value of the previous incumbent solution. The perturbation phase and its components are presented and detailed in Section 5.4.

After the perturbation phase, a new CG is run - perturbed CG - which will lead to a new search phase. Afterwards, if no stopping criteria is met, a new perturbation is applied and the process iterates again. In SearchCol, the perturbation to be executed can be problem independent, with several perturbations algorithms available, considering the incumbent and/or the optimal solution and with deterministic or probabilistic characteristics. Problem specific perturbations can also be implemented to improve the quality of the constraints added to the RMP, taking into account problem characteristics.

The stopping criterion in SearchCol can be met using: a time limit, a limited number of search iterations, a certain improvement on the value of the incumbent solution or a limited number of total iterations without improvement (a total iteration comprises the execution of the three referred steps).

A SearchCol iteration results in a node of a tree. If a pure SearchCol approach is applied, the result from several iterations is a *forest* composed of several trees where each tree can be made of nodes with one or more descendants [Alvelos, 2012] as in each iteration is possible to restart from the root node.

SearchCol manages the methods being used in each iteration and in each phase of the iteration. This is done considering the different alternatives in the components and type of nodes obtained from each step.

A SearchCol run can be configured through several general parameters that guide the

global algorithm, through phase parameters that define how the component behaves, and through a configurator parameter that guides each SearchCol iteration depending on the type of node resulting from a previous iteration. This way, different searchers or *perturbators* can be used depending on the type of nodes obtained.

The computational implementation of the SearchCol is introduced in Chapter 6 with more detailed information about the available SearchCol configurations and parameters being used in this work.

5.2 Column Generation

In the UPMSPjs approach to the SearchCol algorithm, the solution method for the decomposition model relies on solving CG and obtaining then an integer solution based on the columns of the last RMP. This approach is introduced in this section regarding the CG itself, the development of initial solutions and the solving of the SPs (whether through exact methods or with heuristic methods).

In figure 5.2 a representation of the CG is shown.

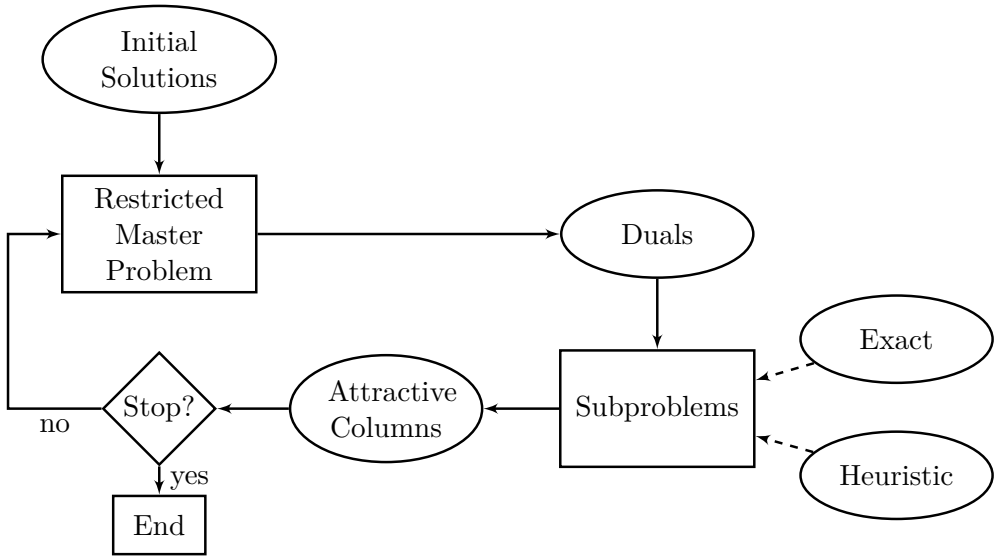

Figure 5.2: Column Generation

Before the first generation of columns, initial solutions that act as columns, are inserted into the first RMP using different problem specific heuristics (see Subsection 5.2.1).

The CG is then solved using the two possible SearchCol techniques: a general and exact method, and problem specific heuristics (to be discussed in Subsection 5.2.2 with results compared in Chapter 6).

The heuristics to solve the SPs differ by using either one SP or all the SPs at each time. Considering one SP at a time implies SPs are solved independently and an overall feasible solution is not guaranteed, whether when building solutions within the context of all the SPs, allows feasible solutions to be obtained for the overall problem.

When solving the SPs, attractive columns are generated and inserted into the RMP until the optimal solution of the RMP is found, i.e. the optimal solution of the overall linear problem is found (optimal LR solution). The other way, each time an optimal LR solution is found for the current RMP, dual information is provided for the SPs that will generate new and attractive columns, until the optimal solution of the overall linear problem is found. SearchCol allows SPs to be solved either exactly or with specific heuristics.

5.2.1 Initial Solutions

The introduction of initial solutions provides an upper bound to the optimal solution, as well as quality and feasible solutions to be inserted in the RMP as columns. Five different types of initial solutions were designed taking into consideration the characteristics of the problem, such as due dates, setup times, processing times and weight of the jobs.

Table 5.1: Initial Solutions' Overview

Type 1	due date rule with sequential processing
Type 2	average processing time rule with processing on most attractive periods of chosen machine (chosen through number of available periods and processing time ratio)
Type 3	average processing time rule with processing on most attractive periods of chosen machine (chosen through average weight on objective function of the available periods and processing time ratio)
Type 4	average processing time rule with processing on most attractive periods of chosen machine (machine chosen through average weight on objective function of the available periods and processing time ratio. If processing causes delay, the machine chosen will be the one where the delay is minimal)
Type 5	due date rule with processing on most attractive periods of chosen machine (chosen through average weight on objective function of the available periods and processing time ratio)

For an overview of the developed types, see Table 5.1.

The created heuristics function in two steps. First, an ordered list of jobs is created (using different sorting rules for each type of initial solutions), and then each job is allocated to a certain machine and certain periods of its schedule accordingly to each solution's type objective.

An example of each type of initial solution is provided in the following sections using problem data from Table 5.2, with 2 machines and 5 jobs. The T_{max} value indicates the number of periods used for the example problem.

Table 5.2: Parameters for the Initial Solutions' Example

Machine i	$i=1$	$i=2$
q_i	0	1
$z[i]$	3	2

Job j	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
p_{1j}	3	2	2	2	4
p_{2j}	3	3	2	1	1
r_j	0	1	3	2	2
d_j	1	9	5	4	9
w_j	3	2	1	2	3
s_j	2	1	2	2	1

$$\beta = 0.99 \qquad T_{max} = 15$$

Initial Solution - Type 1

In this type of initial solutions, the EDD rule is applied. Jobs will be sorted increasingly by due date and jobs with the same date are to be sorted by the release date so that jobs released earlier are placed first in the sorted list.

Applying this rule for single machines is trivial, but for the parallel machines case other impositions must be made. Release dates for both machines and jobs ought to be

respected, although if the release date for the first job of each machine is smaller than the machine's release date, setup times can be incurred before the job's release date. Initial setup configuration is also taken into account.

To do this, jobs are picked (activated) through the ordered list until no job is left to plan. Initially, a virtual marker is set to each machines at its release date. For the job under analysis a conclusion time is calculated for every machine. The earliest calculated time will indicate which machine will process the job. That will then be allocated to that machine, the marker for the activated machine will be updated and the following job in the ordered list will be picked and analysed. This process is repeated until no jobs are left on the order list to be planned.

An algorithmic representation of the heuristic is provided in Algorithm B.1 in Appendix and a detailed step by step description is given below.

1. Sort jobs increasingly by due date.
   (a) For jobs with the same due date, sort subset of jobs non-increasingly by weight of the job.
   (b) For jobs with the same due date and weight, sort subset of jobs increasingly by release date of the job.
   (c) For jobs with the same due date, weight and release date, sort subset of jobs increasingly by index.
2. From the list of ordered jobs, pick the first job j not yet scheduled.
3. For every machine i, calculate at which period job picked in step 2 will finish processing, starting setup (if this job is not the first and already prepared job for the machine) from the first available period (first period after machine's release date or previous processing) and choose machine where processing finishes earlier. If finishing time is the same, machine with lowest index number is chosen.
4. Process job j from step 2 in machine i from step 3. Go back to step 2 until no job left to plan.

This type of initial solutions provides a fairly easy and quick method to build a scheduling but, with its sequential planning, though the due date is the first sorting criterion, it will neglect the same criterion when planning, with job processing occurring fairly distant from its due date in most cases, whether before or after (in least or most congested instances, respectively).

Table 5.3: Type 1 Initial solution's Sorted list

#	Job j	d_j	w_j	r_j
1	1	1	3	0
2	4	4	2	2
3	3	5	1	3
4	5	9	3	2
5	2	9	2	1

The resulting schedule for the type 1 heuristic is demonstrated in Figure 5.3 with the sorting list given in Table 5.3. This schedule has a solution value of 46.62.

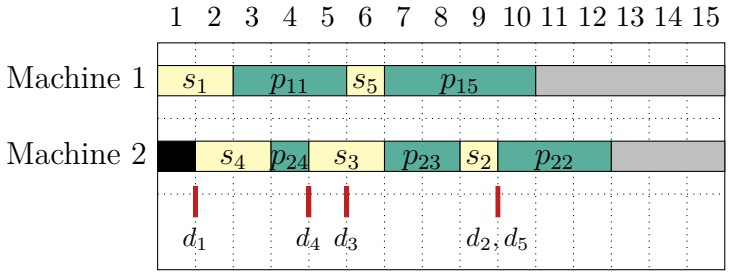

Figure 5.3: Scheduling example with Initial Solution Type 1

Initial Solution - Type 2

For type 2 and the following types of initial solutions, a different method is used to allocate jobs to the machine in the available periods so that the due date is not neglected. That is to say that available periods are inspected to retain the most attractive ones for the job and machine being analysed. Attractive periods are considered to be the ones where the job allocation implies the lowest possible weight on the full solution value. The attractiveness of each period differs from job to job and can be calculated through:

$$attractiveness_{jt} = \begin{cases} (d_j - t) \times (1 - \beta) & \text{if } t \leq d_j \\ (t - d_j) \times \beta & \text{if } t > d_j \end{cases}$$

The aim in this type of initial solution is to allow the most difficult jobs to be planned first, considering also its due dates. Difficult jobs are considered to be, in this type, the ones

with the highest average processing times, and so, the ones which imply bigger solution values considering all jobs would finish on the due date.

The job splitting property is applied in this type of initial solution, when in the first sequence of a job processing, the following most attractive period is not following or preceding the previous processing periods.

An algorithmic representation of the heuristic is provided in Algorithm B.2 in Appendix and a detailed step by step description is given below.

1. Sort jobs non-increasingly by average processing time of job for all machines.
   (a) For jobs with the same average processing time, sort subset of jobs increasingly by due date.
   (b) For jobs with the same average processing time and due date, sort subset of jobs increasingly by index.
2. From the list of ordered jobs, pick the first job j not yet scheduled.
3. For every machine i, choose machine with the highest ratio of:

   $$\frac{available\,periods_i}{processing_{ij}}$$

   (a) If there is more than one machine with the same value of highest ratio, the machine with the lowest index will be chosen.
4. Process job j (from step 2) in machine i (from step 3) on the most attractive periods.
5. Repeat the process starting from step 2 until no job is left to schedule.

Table 5.4: Type 2 Initial solution's Sorted list

#	Job j	avg_proc_j	d_j	M_1_ratio	M_2_ratio	r_j
1	1	3	1	5	4.6(6)	0
2	2	2.5	9	5	6.5	1
3	5	2.5	9	1.75	13	2
4	3	2	5	3.5	5	3
5	4	1.5	4	3.5	8	2

The resulting schedule for the type 2 heuristic is demonstrated in Figure 5.4 with the sorting list given in Table 5.4. This schedule has a solution value of 42.60.

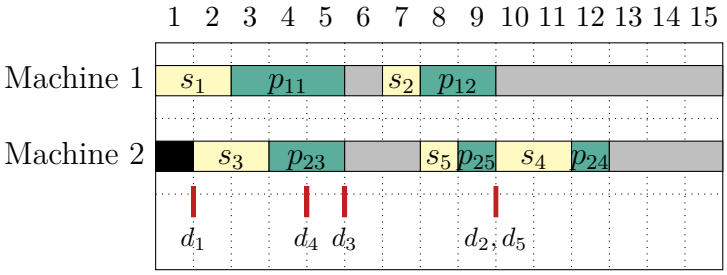

Figure 5.4: Scheduling example with Initial Solution Type 2

Initial Solution - Type 3

Type 3 of initial solutions follows the same principle of Type 2, except for the way the machine is chosen to process the already chosen job. The ratio of each machine is now calculated using the average weight of the available periods on the OF, so machines with equal available periods and same processing times for a given job, can be differentiated on the quality of the available periods. Machines where the available periods fall more upon pre due date periods than post due date periods have a better chance to be chosen.

The job splitting property is applied in this type of initial solution, when in the first sequence of a job processing, the following most attractive period is not following or preceding the previous processing periods.

An algorithmic representation of the heuristic is provided in Algorithm B.3 in Appendix and a detailed step by step description is given below.

1. Sort jobs non-increasingly by average processing time of job for all machines.
   (a) For jobs with the same average processing time, sort subset of jobs increasingly by due date.
   (b) For jobs with the same average processing time and due date, sort subset of jobs increasingly by index.
2. From the list of ordered jobs, pick the first job j not yet scheduled.
3. For every machine i, choose machine with the lowest ratio of:

$$(average\,period\,weight\,on\,\mathrm{OF})_i \times processing_{ij}$$

with

$$(average\,period\,weight\,on\,\mathrm{OF})_i = \frac{total\,weight\,of\,available\,periods\,on\,\mathrm{OF}}{number\,of\,available\,periods}$$

(a) If there is more than one machine with the same value of lowest ratio, lowest index machine will be chosen.

4. Process job j (from step 2) in machine i (from step 3) on the most attractive periods.
5. Repeat the process starting from step 2 until no job is left to schedule.

Table 5.5: Type 3 Initial solution's Sorted list

#	Job j	avg_proc_j	d_j	M_1_ratio	M_2_ratio	r_j
1	1	3	1	20.79	22.28	0
2	2	2.5	9	4.17	4.52	1
3	5	2.5	9	11.90	1.62	2
4	3	2	5	13.01	8.65	3
5	4	1.5	4	14.99	6.93	2

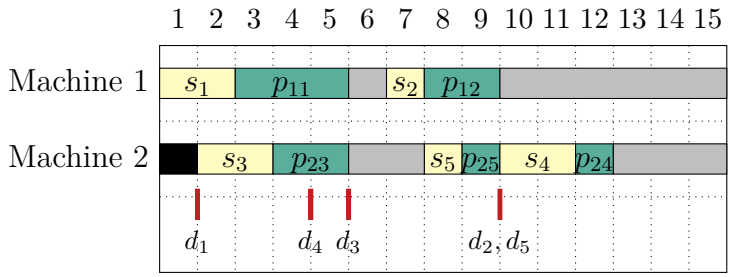

Figure 5.5: Scheduling example with Initial Solution Type 3

The resulting schedule is the same as the previous type, as the ratio calculation for this example results in the same machine allocation for each job. The schedule for the heuristic is demonstrated in Figure 5.5 with the sorting list given by Table 5.5. This schedule has a solution value of 42.60.

Initial Solution - Type 4

Type 4 of initial solutions is derived from the previous type 2 and, mainly, type 3. All criteria in this type are alike type 3, with the difference being that while allocating processing lots, with job and machine already chosen, if one lot is at a tardy position in time related to the due date, a new machine is chosen (the machine where the job will

finish remaining processing earlier) and the remaining proportion of processing (the one that was to be processed after the due date) is done in the earliest possible periods.

The job splitting property is applied in this type of initial solution, when in the first sequence of a job processing, the following most attractive period is not following or preceding the previous processing periods, or in the case it is a period implying tardiness. In the case it is a period of tardiness, splitting occurs so processing may occur in a machine different from the one being used in the first sequence.

The job splitting property is applied in this type of initial solution, the first sequence of a job processing becomes tardy or the following period is unavailable.

An algorithmic representation of the heuristic is provided in Algorithm B.4 in Appendix and a detailed step by step description is given below.

1. Sort jobs non-increasingly by average processing time of job for all machines.
   (a) For jobs with the same average processing time, sort subset of jobs increasingly by due date.
   (b) For jobs with the same average processing time and due date, sort subset of jobs increasingly by index.
2. From the list of ordered jobs, pick the first job j not yet scheduled.
3. For every machine i, choose machine with the lowest ratio of:

   $$(average\,period\,weight\,on\,\mathrm{OF})_i \times processing_{ij}$$

   with

   $$(average\,period\,weight\,on\,\mathrm{OF})_i = \frac{total\,weight\,of\,available\,periods\,on\,\mathrm{OF}}{number\,of\,available\,periods}$$

   (a) If there is more than one machine with the lowest ratio, lowest index machine will be chosen.
4. Process job j (from step 2) in machine i (from step 3) on the most attractive periods.

   (a) If any processing unit of the chosen job and machine is about to be allocated to a tardy period, the remaining machines are tested and the remaining job's processing proportion is processed in the machine where it finishes earlier.
5. Repeat the process starting from step 2 until no job left to schedule.

The resulting schedule for the example presented on Table 5.2 on page 42 produces the same schedule as type 2 and 3. To provide a better demonstration, the following changes in the input data for job 4 must be considered:

$$d_4 = 7 \qquad p_{14} = 2 \quad p_{24} = 2 \quad s_4 = 1$$

Table 5.6: Type 4 Initial solution's Sorted list

#	Job j	avg_proc_j	d_j	M_1_ratio	M_2_ratio	r_j
1	1	3	1	18.02	19.31	0
2	2	2.5	9	4.17	4.52	1
3	5	2.5	9	11.90	1.62	2
4	3	2	5	13.01	9.51	3
5	4	2	7	9.34	8.17	2

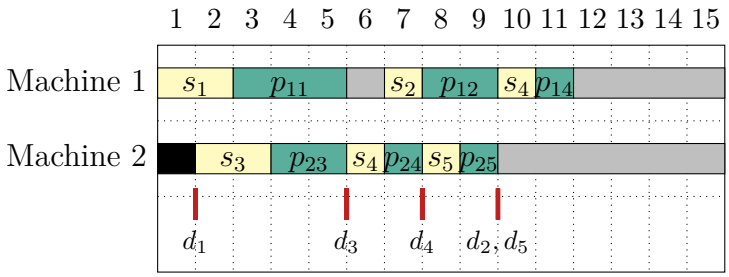

Figure 5.6: Scheduling example with Initial Solution Type 4

The scheduling will result in tardiness for jobs 1 and 4. For job 1 the extra step from this heuristic produces the same result as previous heuristics, but for job 4, machine 2 processes it idle periods between processing jobs 3 and 5, with following processing being necessarily tardy. Calculation to check which machine finishes remaining production earlier indicates both machines will finish at the same time, so machine 1, with the lowest index, is chosen to process remaining periods. This type of solution is demonstrated in Figure 5.6 with the sorting list given by Table 5.6. This schedule has a solution value of 34.68[3].

[3]This value cannot be compared to the other types' value as parameters of the job 4 have been changed for this example.

Initial Solution - Type 5

Type 5 of initial solutions was built around using the EDD rule from type 1 and the period allocation of the types 2 and 3 - processing on most attractive periods.

The job splitting property is applied in this type of initial solution, when in the first sequence of a job processing, the following most attractive period is not following or preceding the previous processing periods.

An algorithmic representation of the heuristic is provided in Algorithm B.5 in Appendix and a detailed step by step description is given below.

1. Sort jobs increasingly by due date.
   (a) For jobs with the same due date, sort subset of jobs non-increasingly by average weighted processing time:

$$w\_avg\_proc_j = \frac{w_j \sum_i p_{ij}}{M}$$

   (b) For jobs with the same due date and average weighted processing time, sort subset of jobs increasingly by index.
2. From the list of ordered jobs, pick the first job j not yet scheduled.
3. For every machine i, choose machine with the lowest ratio of:

$$(average\,period\,weight\,on\,\mathrm{OF})_i \times processing_{ij}$$

   with

$$(average\,period\,weight\,on\,\mathrm{OF})_i = \frac{total\,weight\,of\,available\,periods\,on\,\mathrm{OF}}{number\,of\,available\,periods}$$

   (a) If there is more than one machine with the lowest ratio, lowest index machine will be chosen.
4. Process job j from (step 2) in machine i from (step 3) on the most attractive periods.
5. Repeat the process starting from step 2 until no job left to schedule.

The example to demonstrate this type of initial solutions is the original one presented on Table 5.2 on page 42.

Table 5.7: Type 5 Initial solution's Sorted list

#	Job j	$w\_avg\_proc_j$	d_j	M_1_ratio	M_2_ratio	r_j
1	1	1	-	20.79	22.28	0
2	4	4	-	12.87	5.03	2
3	3	5	-	10.89	9.90	3
4	5	9	7.5	8.34	2.97	2
5	2	9	5	4.17	11.88	1

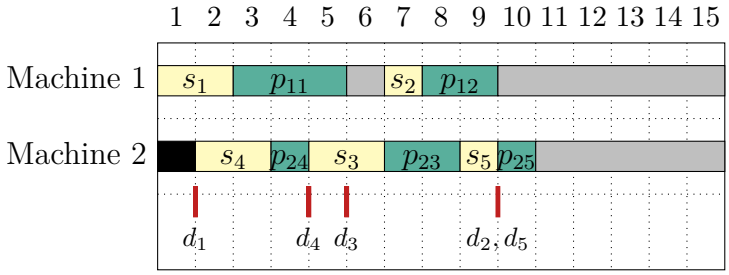

Figure 5.7: Scheduling example with Initial Solution Type 5

The resulting schedule is demonstrated in Figure 5.7 with the sorting list given by Table 5.7. This schedule has a solution value of 34.67.

5.2.2 Subproblems

In SearchCol, as stated previously, it is possible to solve SPs heuristically or exactly. It is also possible to configure how the RMP and the SPs interact between each other, specifically in the number of SPs being solved in each iteration and the amount of attractive columns being added to the RMP.

In each iteration, it is possible to define which columns are being inserted in the RMP: all attractive columns per iteration, the best column per iteration or one column per iteration. The last option can only be used if the SPs are not being solved heuristically, and in each iteration the SP being solved after solving the RMP can either be the same or the next SP. In the case of inserting the best column, all SPs are being solved and only the best is considered.

To solve the SPs exactly SearchCol uses a general MIP solver. To solve the SPs heuris-

tically, two methods are available: Independent and Global SP heuristics. In the next two subsections, the developed heuristics to solve the SPs are introduced, as well as the methods they refer to. Results from solving the decomposition model, comparing the strictly exact approach and the heuristic approaches to solve the SPs, are presented in Chapter 6.

Independent Subproblem Heuristic

SearchCol allows user implementation of a specific problem heuristic to solve the SPs in some iterations. If the developed heuristic is of the independent type, it can be used in the following cases:

1. the heuristic is used until it doesn't generate any attractive columns and then the exact method is used only once and the process is repeated (the heuristic is used again);

2. the heuristic is always used (does not assure optimality of CG);

3. the heuristic is used until it doesn't generate any attractive columns and then the exact method is used until the end.

An heuristic to solve each SP is defined here. This heuristic is run independently, as each SP is solved independently of other SPs solutions. To determine which jobs (and consequently, which periods) are attractive to produce in a new schedule of a given machine, the heuristic uses, besides the intrinsic problem data, the dual information from the last RMP. The attractiveness of jobs (and periods) is measured by the 'cost' in the OF of producing the referred jobs in each period, which is calculated using the dual Π_j obtained from the RMP.

The 'cost' is directly related to the SP's OF (see 4.14) and takes the form of Equation 5.1 to calculate the value of a given period for a given job in a given SP. This 'cost' is calculated for all periods of all given jobs in the SP being solved. In this equation, i represents the

SP being solved and the remaining notation follows the notation developed in Chapter 4.

$$cost_{ijt} = \begin{cases} \beta(t-d_j)w_j - \frac{1}{p_{ij}}\Pi_j & \forall i, \forall j, \forall t : t > d_j \wedge t > max\{r_i, q_i\} \\ (1-\beta)(d_j-t)w_j - \frac{1}{p_{ij}}\Pi_j & \forall i, \forall j, \forall t : t \leq d_j \wedge t > max\{r_i, q_i\} \\ \beta(t-d_j)w_j & \forall i, \forall j, \forall t : t > d_j \wedge t \leq max\{r_i, q_i\} \\ (1-\beta)(d_j-t)w_j & \forall i, \forall j, \forall t : t \leq d_j \wedge t \leq max\{r_i, q_i\} \end{cases} \tag{5.1}$$

The heuristic consists in:

1. Sort jobs for the machine associated with the current SP:
   (a) Increasingly by the most negative 'cost' from the set of the job's periods;
   (b) Increasingly by $\frac{d_j}{w_j \times (s_j + p_{ij})}$;
   (c) Increasingly by Index;
2. On sorted list, pick the first job not scheduled and process it:
   (a) from the period with the most negative 'cost' to the least negative:
      i. until job's processing is complete;
      ii. until no available periods to plan;
      iii. until a job splitting is needed to complete processing;
3. Repeat step 2 until no more jobs left or no available periods in the scheduling plan.

It must be noted that when in step 2, if conditions 2a.ii) and 2a.iii) are met, the remaining processing is not processed in this scheduling plan.

Due to the characteristics of the developed model (in Chapter 4), one disadvantage when solving the SPs using the dual information, either exactly or through problem specific heuristics, is the quality of an overall solution (the solution of the RMP), considering all SPs or the scheduling plans of all the machines. This overall solution, despite its advantages on obtaining better lower bounds, may have poor quality when forcing a complete solution (or even be infeasible if no feasible initial solution was inserted in the first RMP). This is caused by the type of dual information provided by the RMP, as it provides the SPs with the same dual information (variable π_j) for all periods of a given job, with the reduced cost for each machine (and its associated SP) differing only because of the different processing

times for the same job on different machines, which may be insufficient for an overall solution when building the solution of each machine.

This aspect results in the achievement of very similar schedules for different machines in a given iteration of the CG. Moreover, the most attractive jobs have a high probability of being schedules in all the machines (in a given iteration of the CG) and other jobs have a high probability of not being scheduled on any machine. Being so, when building a machine schedule, it is important to know which machine will process each job. By either solving the problem exactly or using the previous heuristic, it is impossible to understand the overall behaviour of the schedule and define the exact amount of jobs to be processed in each iteration for each machine.

An algorithmic representation of the heuristic is available in the Appendix (Algorithm B.6).

Global Subproblems Heuristic

In order to contour this drawbacks, a new heuristic was developed in the SearchCol implementation of the UPMSPjs that would be able to solve the overall problem, defining all SPs solutions at one step in an iteration of the CG and providing the RMP with all SPs solutions that can be easily seen as an unique solution, as it results in overall feasible solutions (where all the jobs are scheduled).

As in previous heuristic, the iterations where the heuristic is used can also be configured and follows the same principle presented on Page 52.

Taking into account the previously presented Independent SP Heuristic to solve the SPs and Equation 5.1, the following heuristic was developed considering the complete set of machines (SPs):

1. Sort jobs:
   (a) Increasingly by the most negative 'cost' from the set of the job's periods;
   (b) Increasingly by due date;
   (c) Decreasingly by weight;
   (d) Increasingly by index.
2. On sorted list, pick the first job not scheduled.
3. Sort machines:

(a) Increasingly by 'cost' of processing in available periods.

4. Schedule job taking into account the first machine of the sorted list of machines from step 3, and on first available period with the most negative cost on the OF.
   (a) If no available period, change to the next machine on the sorted list;
   (b) If no available period and all machines checked, restart from step 3 allowing processing on periods with non negative cost (only for the job being scheduled).
5. Repeat the process starting from step 2 until all jobs are scheduled.

The 'cost' of processing in available periods (referred in Step 3a) is calculated through the sum of values for the current available periods of the SP's OF and it is represented in Equation 5.2 for all periods respecting conditions in Equation 5.3.

$$\begin{aligned} cost_{processing\_j\_in\_available\_t} = & \sum_{t>d_j \wedge t>max\{r_i,q_i\}}^{Tmax} \left(\beta(t-d_j)w_j - \frac{1}{p_{ij}}\Pi_j\right) + \\ & \sum_{t\leq d_j \wedge t>max\{r_i,q_i\}}^{Tmax} \left((1-\beta)(d_j-t)w_j - \frac{1}{p_{ij}}\Pi_j\right) + \\ & \sum_{t\leq d_j \wedge t\leq max\{r_i,q_i\}}^{Tmax} (1-\beta)(d_j-t)w_j + \\ & \sum_{t>d_j \wedge t\leq max\{r_i,q_i\}}^{Tmax} \beta(t-d_j)w_j \end{aligned} \tag{5.2}$$

$$\sum_{j=1}^{n} x_{ijt} = 0 \qquad \forall t \tag{5.3}$$

An algorithmic representation of the heuristic is available in the Appendix (Algorithm B.7).

5.3 Search

In the search phase, two different approaches were used within the range of possibilities: MIP and VNS. Besides the mentioned methods, other approaches are already implemented

or being implemented on the SearchCol framework, namely GRASP, Tabu Search, Simulated Annealing [Alvelos, 2012; Alvelos et al., 2013] and Genetic Algorithms [Barbosa et al., 2013].

In the next two subsections a brief explanation for the search algorithms is provided, namely the MIP searcher and VNS and its additional component of Local Search.

5.3.1 MIP Searcher

SearchCol's search phase was designed to be conducted through (meta)heuristics, though it is possible to use a MIP solver as a searcher.

When using the MIP searcher, taking into account the last RMP (the one associated with the optimal solution of the LR of the decomposition model), it is forced the selection of one column of each SP in order to find a feasible integer solution. However, it is not guaranteed the achievement of a feasible integer solution unless, as in the case of this work, feasible initial solutions are inserted in the beginning of the CG.

In a general way, using the MIP searcher it could be said the search phase is solved by an exact method, though, it is not guaranteed that the columns present in the RMP associated with the optimal solution of the CG - that is the one that will be used during the MIP search - are the ones corresponding to the problem's optimal integer solution. Thus, the solution provided by the Searcher is the optimal solution within the set of SPs solutions available from solving the decomposition model.

5.3.2 VNS Searcher

VNS is a MH largely known and studied [Hansen et al., 2010]. The Local Search method is used in VNS's SearchCol as a component within the searcher.

A k-neighbourhood is defined as being the solution obtained when changing k or less SPs from the set of solutions. In our problem, a 1-neighbourhood results from changing 1 machine schedule and a 2-neighbourhood results from changing the schedule of 1 or 2 machines.

The size of a 1 and 2-neighbourhood can be calculated using, respectively:

$$S_1 = \sum_{k \in K} (n_k - 1)$$

$$S_2 = \sum_{k_1 \in K} (n_{k_1} - 1) \times \sum_{k_2 \in K : k_2 > k_1} (n_{k_2} - 1)$$

where n_k is the number of SP solutions for SP k.

Besides the value of k, the descent strategy being used can be either by first or best improvement. If using a first improvement descent strategy, each time a better neighbourhood solution is found, this becomes the current solution. Otherwise, using a best improvement descent strategy, a search is done for all solutions in the neighbourhood and if the best of the neighbours is better than the current, it becomes the current solution.

In the SearchCol's VNS algorithm (see Algorithm 5.1), the current neighbourhood is set to the first of the hierarchy ($k = 1$) and local search is applied from an initial solution in the current neighbourhood of the current solution, by changing the SP solution of k SPs. When the local optimum is better than the current solution a new iteration starts from the $k = 1$ neighbourhood, otherwise a more distant neighbourhood becomes the current (by incrementing k until a maximum number of SPs is included in the neighbourhood).

The change in the overall solution of k SPs is done randomly: for example, for a solution s formed by 4 SPs with $k = 2$, a possible change of $s = (1, 2, 1, 2)$ could be $s = (3, 2, 1, 1)$ where SP_1 and SP_4 have their solution changed.

The stopping criterion can be set to a time limit or to one outer iteration without improvement of the incumbent solution.

Constructors

To start the search, SearchCol's VNS requires an initial solution to be determined. The component responsible for creating the initial solutions is the constructor.

This initial solution can based on the CG optimal solution, based on CG history, through a greedy constructor or randomly defined.

Two methods were used in this work from the range of available constructors: Higher Weights and Incumbent. For the Higher Weights constructor, the solution is built using

Algorithm 5.1: SearchCol's VNS

```
s = initialsolution()
k = 1
while stopping criterion not met do
    if k == k_max then
        k = 1
    end
    s' = perturbRandomly(s, k)
    s'' = localSearch(s')
    if s'' better than s then
        k = 1
        s = s''
    else
        if k ≠ k_max then
            k = k + 1
        end
    end
end
return s
```

for each SP the solution associated with the highest value in the optimal solution of the last solved RMP. For the Incumbent constructor, the solution generated is copied from the incumbent. In the case of a first iteration in the UPMSPjs SearchCol algorithm, this incumbent is the first initial solution added to the RMP. For other SearchCol algorithms not initializing the CG with an heuristic solution, this incumbent is formed by one column for each SP with null reduced cost.

5.4 Perturbations

A perturbation in SearchCol is a method that inserts into the RMP new constraints not present in the original decomposition model. The constraints being added, fix SP variables[4] to 0 or 1 depending on the method being used or on a problem specific implementation objective.

SearchCol's perturbation component offers several standard possibilities using the incumbent solution or the optimal LR solution, or even both by combining the values of each

[4]Variables x_{ijt} in the case of the UPMSPjs.

and following the chosen method rules. The *perturbators* (SearchCol's perturbations component) can also have a stochastic component by using some probabilistic method to decide a certain perturbation. In every node, except for the root node, a set of perturbations is necessarily present.

In this section the SearchCol *perturbators* being used are introduced and an implementation of problem specific *perturbators* is suggested.

5.4.1 SearchCol *perturbators*

The use of SearchCol *perturbators* has two main alternatives for whether the incumbent solutions are feasible or infeasible. Considering feasible initial solutions are already inserted when starting the UPMSPjs implementation of the SearchCol algorithm, this subsection will focus on *perturbators* for feasible incumbent solutions. Detailed information of both types of *perturbators* is available in Alvelos [2012] and Alvelos et al. [2013].

Current SearchCol *perturbators* can be of the following types:

- *Branch*
- *CombProb*
- *CombType*
- *Duals*
- *Memory*
- *SP*
- *ViolsType*

In this work, the focus will be on *CombProb* and *CombType perturbators*. Both of the *perturbators* use the incumbent solution and the optimal LR solution of the current SearchCol iteration.

CombProb

In this perturbation a probabilistic combination between the incumbent solution and the optimal LR solution is used to define the new constraints to be added to the RMP. First a convex combination between the two solutions is defined, through a user defined parameter (PAR_{weight_X}) to weight both solutions.

$$x_{comb} = PAR_{weight_X} \times x + (1 - PAR_{weight_X}) \times x_{inc}$$

From this combination of values(x_{comb}), a second parameter ($PAR_{Threshold}$) is used as a threshold to help round the combination value.

$$x_{comb} \leq PAR_{Threshold} \rightarrow x_{comb} = 0$$

$$x_{comb} > 1 - PAR_{Threshold} \rightarrow x_{comb} = 1$$

Finally, two user defined parameters (PAR_{Prob10} PAR_{Prob11}) are applied as probability reference values to determine whether a variable x_{comb} with value 1 is fixed to 0 or to 1.

CombType

This *perturbator* also uses a combination between the incumbent solution and the optimal LR solution, offering four different types of combinations, presented below.

Type=0
i) Variables with value 1 in the optimal LR solution and value 1 in the incumbent solution are fixed to 1;
ii) Variables with a fractional value in the optimal LR solution and value 0 in the incumbent solution are fixed to 0.

Type=1
i) Variables with a fractional value in the optimal LR solution and value 0 in the incumbent solution are fixed to 0.

Type=2
i) Variables with value 0 in the optimal LR solution and value 0 in the incumbent solution

are fixed to 0;
ii) Variables with a fractional value in the optimal LR solution and value 0 in the incumbent are fixed to 0;
iii) Variables with value 1 in the optimal LR solution and value 1 in the incumbent solution are fixed to 1.

Type=3
i) Variables with value 1 in the optimal LR solution and value 1 in the incumbent solution are fixed to 1;
ii) Variables with value 0 in the optimal LR solution and value 0 in the incumbent solution are fixed to 0.

Type=4
i) Variables with value 0 in the optimal LR solution and value 1 in the incumbent are fixed to 0.

As in the previous *perturbator*, a user defined parameter ($PAR_{Threshold}$) is applied to round the fractional values in the optimal LR solution.

$$x \leq PAR_{Threshold} \rightarrow x = 0$$

$$x > 1 - PAR_{Threshold} \rightarrow x = 1$$

In the case of the UPMSP SearchCol implementation, the *perturbator CombType* will be tested with Type 0. Results from the tests from this and previous SearchCol *perturbators* are available in Chapter 6 on Page 82.

5.4.2 Specific *perturbator*

In this subsection a specific *perturbator* is suggested for the problem being studied in this work, by combining the optimal solution of the last CG and the current incumbent solution.

Three different types of perturbations are suggested by defining parameters for the specific *perturbator*: type 1, type 2 and type 3 ($PAR_{pert\_spec\_type}$, equal to 1, 2 or 3, respectively).

The main idea behind the *perturbator*'s implementation is to force new solutions (new columns that will be generated in the next CG step) to have, for a given job, specific time periods in a specific machine set to value 1 for the processing variables x_{ijt}. This *perturbator* will run in each new SearchCol iteration (and while the SearchCol stopping criterion is not met) keeping previously created perturbations and generating at least one new perturbation in each new iteration, so the CG does not become redundant.

In the UPMSP taking into account the OF being used, the obvious would be to fix processing when it is done on the most attractive periods, both in the linear and integer solution - considering that the attractive periods are the ones that correspond to the due date of each job. Also, an extension can be made considering that it is impossible to satisfy all the job's demand by processing only on job's due date period, therefore fixing also the processing variables in the periods just before the due date[5] and until processing is completed, or until no available periods exist that precede the due date. This extension is what distinguishes the types of *perturbators* developed.

For type 1 and type 3, the *perturbator* starts by comparing only the job's due date periods and in each new iteration, if a due date period was already fixed, the next attractive period is also fixed, considering conditions are met in terms of the linear and integer solution and that demand has not yet been satisfied. The referred conditions are presented next when detailing each type of specific *perturbator*.

For type 2, the extension previously referred is considered at the start of the procedure, so the *perturbator* will try to fix the maximum variables it is allowed to fix until demand is satisfied.

As the optimal solution of the last CG will typically contain fractional variables, a second parameter $PAR_{pert\_spec\_threshold}$ is considered to define the threshold for which a variable can be rounded 1. Variables x_{ijt} in the optimal LR solution are considered to have value 1 if:

$$x_{ijt} > 1 - threshold_{iteration}$$

For type 1, the $threshold_{iter}$ value starts by being defined by the $PAR_{pert\_spec\_threshold}$. In every new iteration, if (and only if) no perturbation is added, this threshold is relaxed

[5]If more weight is given to processing without delay through β in the OF.

and the perturbation iteration is repeated. In a future SearchCol iteration, when starting the perturbator phase again, the relaxation is ignored and the parameter is defined as configured with $PAR_{pert\_spec\_threshold}$.

For type 2 and type 3 (type 3 which derives from the previous type 1), a relaxation is done on the threshold parameter which, in each iteration, is updated[6] (see Table 5.8) by the following equation:

$$threshold_{iter} = PAR_{pert\_spec\_threshold} + (iter \times PAR_{pert\_spec\_threshold})$$

Table 5.8: Iteration's threshold value on perturbators Type 2 and 3

	$PAR_{pert\_spec\_threshold} = 0.1$	
	$threshold_{iter}$	$1 - threshold_{iter}$
$iter = 1$	0.1	0.9
$iter = 2$	0.2	0.8
$iter = 3$	0.3	0.7
$iter = 4$	0.4	0.6
$iter = 5$	0.5	0.5

It can be said that, generally, at an extreme situation the *perturbator*, by forcing part of the solution in each iteration, would reach after a given number of iterations the full solution - a complete schedule - for the problem. This happens as in each new iteration a new job is *fixed* through the perturbations created in past iterations, until no jobs are left open to schedule. The SearchCol iteration will meet its stopping criteria by either reaching the total time limit or by being unable to improve the incumbent solution value.

Next, the algorithms for each type of *perturbator* suggested are detailed step by step.

[6] $threshold_{iter}$ is set to a maximum of 1.0.

Type 1

This type of *perturbator* consists in analysing, comparing and combining all the variables x_{ijt} in a determined interval of the optimal LR solution and the incumbent solution in each iteration of the problem.

The interval of periods to consider is determined by the number of times the *perturbator* was called, increasing the interval's size in each new iteration.

From the result of the combination of values, restrictions are added to the RMP of the CG in a future SearchCol iteration, fixing determined variables x_{ijt} to value 1 if given conditions are met.

Type 1 *perturbator* requires user configuration of the parameter $PAR_{pert\_spec\_threshold}$ to define the threshold and round to value 1 the variables x_{ijt} of the optimal LR solution. In every iteration the threshold value is used as defined by the parameter, but can be relaxed if in a given iteration no perturbation is added to the RMP, in order to avoid overall redundancy.

Consider $iteration = 0$ to start:

i) Retrieve information of previous perturbations (if they exist);
ii) Set $threshold_{iteration} = PAR_{pert\_spec\_threshold}$.
iii) Fix to value 1 variables x_{ijt} in the interval of periods:

$$[d_j - \min\{iteration\ ,\ proc_{ij}\}\ ,\ d_j]$$

of any of the jobs, if all the following conditions are met:

a) no previous perturbation was added for the same period and machine;
b) the value of the variable x_{ijt} is 1 in the optimal LR solution and 1 in the incumbent solution;
c) all the variables x_{ijt} between the due date and the period being analysed have the value 1 in the optimal LR solution and value 1 in the incumbent solution.
d) the variable is associated to a period greater than the release date of the machine and of the job;
e) a complete setup can be incurred considering all the perturbations included on that machine.

iv) If the number of perturbations created in step iii) is equal to 0, update:

$$threshold_{iteration} = threshold_{iteration} + PAR_{pert\_spec\_threshold}$$

and go back to step iii).

v) Update quantity of perturbations added;

vi) Update $iteration = iteration + 1$

Type 2

This type of *perturbator* consists in analysing, comparing and combining all the variables x_{ijt} in a determined interval of the incumbent solution with the average value of the variables of the optimal LR solution in each SearchCol iteration:

$$X_{average\_in\_interval\_of\_variables} = \frac{\sum_{t \geq d_j - p_{ij}}^{t \leq d_j} x_{ijt}}{d_j - p_{ij}} \qquad \forall i, \forall j$$

The interval considered is stated in the previous formulation and it comprises all periods preceding the due date of a given job in the length of periods related to the processing time of a given job in the corresponding machine. This interval, unlike previous type of specific *perturbator* remains the same size in every new iteration.

From the result of the comparison and combination of values, restrictions are added to the RMP of the CG in a future SearchCol iteration, fixing determined variables x_{ijt} to value 1 if given conditions are met.

Type 2 *perturbator* requires user configuration of the parameter $PAR_{pert\_spec\_threshold}$ to define the threshold and round to value 1 the variables x_{ijt} of the optimal LR solution. In every new SearchCol iteration the threshold value is updated and relaxed.

For type 2 (consider $iteration = 0$ to start):

i) Retrieve information of previous perturbations;

ii) Set: $threshold_{iter} = PAR_{pert\_spec\_threshold} + (iter \times PAR_{pert\_spec\_threshold})$.

iii) Fix to 1 variables x_{ijt} which correspond to the interval of periods:

$$[d_j - processing_{ij}\ ,\ d_j]$$

of any of the jobs, if all the following conditions are met:

a) no previous perturbation was added for the same period and machine;
b) all the variables x_{ijt} in the considered interval have value 1 in the incumbent solution.
c) the average value of the variables x_{ijt} on the optimal LR solution in the considered interval of periods is:

$$X_{average\_in\_interval\_of\_variables} >= 1 - threshold_{iteration}$$

d) the variable is associated to a period greater than the release date of the machine and of the job;
e) a complete setup can be incurred considering all the perturbations included on that machine;

iv) If the number of perturbations created in step iii) is equal to 0, update:

$$threshold_{iteration} = threshold_{iteration} + \frac{PAR_{pert\_spec\_threshold}}{njobs}$$

and go back to step iii).
v) Update quantity of perturbations added;
vi) Update $iteration = iteration + 1$

Type 3

This type of *perturbator* consists in analysing, comparing and combining all the variables x_{ijt} in a determined interval of the optimal LR solution and the incumbent solution in each iteration of the problem.

The interval of periods to consider is determined by the number of times the *perturbator* was called, increasing the interval's size in each new iteration.

From the result of the combination of values, restrictions are added to the RMP of the CG in a future SearchCol iteration, fixing determined variables x_{ijt} to value 1 if given conditions are met.

Type 3 *perturbator* requires user configuration of the parameter $PAR_{pert\_spec\_threshold}$ to define the threshold and round to value 1 the variables x_{ijt} of the optimal LR solution. In every new SearchCol iteration the threshold value is updated and relaxed.

Consider $iteration = 0$ to start:

i) Retrieve information of previous perturbations;

ii) Set:

$$threshold_{iter} = PAR_{pert\_spec\_threshold} + (iter \times PAR_{pert\_spec\_threshold})$$

iii) Fix to 1 variables which correspond to the interval of periods:

$$[d_j - \min\{iteration\ ,\ proc_{ij}\}\ ,\ d_j]$$

of any of the jobs, if all the following conditions are met:

a) no previous perturbation was added for the same period and machine;

b) the value of the variable x_{ijt} is equal to 1 in the optimal LR solution and 1 in the incumbent solution;

c) all the variables between the due date and the period being analysed have the value 1 in the solution and value 1 in the incumbent.

d) the variable is associated to a period greater than the release date of the machine and of the job;

e) a complete setup can be incurred considering all the perturbations included on that machine;

iv) If the number of perturbations created in step iii) is equal to 0, update:

$$threshold_{iteration} = threshold_{iteration} + PAR_{pert\_spec\_threshold}$$

and go back to step iii).

v) Update quantity of perturbations added;

vi) Update $iteration = iteration + 1$.

Chapter 6

Computational Tests

In this chapter, the computational tests' results obtained from the models and methods detailed on the previous chapters are analysed.

Firstly, the computational implementation of the SearchCol framework in C++ programming language, **SearchCol++** [Alvelos, 2012], is introduced and presented. In the following section, general assumptions and considered test conditions are outlined. The third section presents and analyses results from solving the Compact Model through its implementation in SearchCol++ (using the Cplex C++ libraries). After, the decomposition model's implementation and its results are introduced: results for each SearchCol's step and corresponding specific implementations will be presented, namely, the developed heuristics to built the initial solutions and the algorithms to solve the SPs. SearchCol non-specific components were also tested for the Search and Perturbations phases.

6.1 SearchCol++

SearchCol++ is the SearchCol framework implementation using the object oriented programming language C++.

The implementation of the SearchCol algorithm to the UPMSPjs relied on using a core class for the decomposition model and an auxiliary class to read the problem instances data. Through SearchCol++ is also possible to solve the compact model using the CPLEX

libraries. As in the decomposition model, a core class is also used to implement the compact model.

The two core classes contain virtual functions to be used by a derived class, specific to a problem implementation. For the compact class, a function was derived to build the compact model with all the specific information. The decomposition class required extra efforts due to extra implementations. The decomposition specific class derives at least, in a basic implementation, a function to load the decomposition and initialize the SP solver, a function that computes the modified costs, a function that solves the subproblem, a function that converts a SP into a column of the RMP and a function that sets the values of an artificial cost. Moreover, for specific algorithmic implementations the decomposition specific problem class derived: a function to add initial solutions, a function that solves a SP heuristically and a function that solves all SPs heuristically.

Each model, compact and decomposition, are built and compiled independently. Although the use of the initial solutions are automatic, as long as they are implemented, the SPs algorithm to run in each test is chosen through an input file.

The input file is composed of three different types of information: parameters, components and configurators. The parameters guide the overall decomposition and SearchCol algorithms, and help defining the outputs and the stopping criteria being used. The components are used to define which method will be used and how it will be used in the Search (and Construction) phase and the Perturbation phase. The configurators are a combination of components to be applied in each type of node.

A SearchCol++ run starts by the executable file calling parameters that indicate the test instance to use, its file type and the input configuration file of all SearchCol components. A file with results is provided in the end, containing information with results, values, time spent in components and globally, as well as all the parameters and components being used.

All information related to SearchCol++ configuration, implementation and structure is available in Alvelos [2012] and in the software package through HyperText Markup Language (HTML) documentation of all classes and data structures.

SearchCol++ and problem specific classes implementations were coded using integrated development environment Microsoft Visual C++ in Microsoft Visual Studio 2010 with CPLEX 12.2 libraries [ILOG, 2010] for a *x64* platform.

6.2 Testing Conditions and Assumptions

The problem instances were adapted from the work of Lopes and Carvalho [2007] and are composed of 80 instances classified by number of jobs and machines as can be seen in Table 6.1. Throughout this work, the computational tests presented result from the whole set of 80 instances with no subsets tested apart from others.

Table 6.1: Instances' characteristics

M (machines)	2	2	2	2	4	4	4	4	6	6	6	8	8	8	10	10
J (jobs)	20	30	40	50	30	40	50	60	40	50	70	40	60	80	70	100

In the work of Lopes and Carvalho [2007], the setup times are sequence dependent whereas in this work they are sequence independent. These test instances embody all remaining characteristics of the UPMSP being studied, except for the setup times which were adapted, by calculating the average setup time for all the possible sequences of any job j.

An important characteristic to consider in these instances is their scheduling system congestion level (q). For each pair of instances (machines and jobs - in Table 6.1) there are five different levels of congestions. The authors considered that the larger the value q, the more congested the system will be and vice versa. This parameter has particular importance not only because it helps to define the values of the due dates for each instance (as it can be seen in Lopes and Carvalho [2007]), but also because it causes a greater number of tardy jobs, as it becomes impossible (in more congested instances) to allocate all jobs before their respective due date.

In our MIP formulation, using time indexed variables, setup and processing times duration have impact on solving the problem, since the total number of periods for a scheduling plan is calculated with basis on processing and setup times. Therefore original processing times and the adapted setup times were divided by 24 to diminish the periods dimension.

The number of periods for each instance, is calculated by:

$$T_{max} = \max \left\{ \max_j d_j + 1 \, , \; \frac{\max_j r_j + \max_i \left[\sum_j (p_{ij} + s_j \times m) \right]}{m} \right\} \tag{6.1}$$

The parameter β in the OF of both compact and decomposition models was set to 0.99 for all tests.

All tests were run with a time limit of 0.5 hours (1800 seconds).

The following equation was used to calculate the integrality gap in the decomposition model:

$$gap = \frac{Z_{INC} - Z_{LR}}{Z_{INC} + 1e^{-10}}$$

where Z_{INC} is the value of the incumbent solution and Z_{LR} is the value of the optimal LR solution.

Solutions are evaluated based on feasibility and infeasibility values. The infeasible component of a solution represents the impossibility of finding an overall solution (a set of SP solutions) that, in this case, is able to fully satisfy all job's demand. Infeasibility values are given by the number of violated rows times 1000 plus the amount of violations. For the proposed decomposition model, the infeasibility values will correspond to jobs not being completely processed and are given by violation of constraints (4.12) (see Chapter 4):

$$\sum_{h}^{g}\sum_{i=1}^{m}(\sum_{t>max\{r_j,q_i\}}^{T_{max}}\frac{1}{p_{ij}}\alpha_{ijt}^{h})\lambda_i^h \geq 1 \qquad \forall j$$

If, for example, an incumbent solution has 2 jobs with total processing being less than demand, the number of violated rows will be 2. If for one job only 50% is processed, and for the other only 25% is processed, this will result in 0.75 of amount of violation. The total infeasibility value would be 2000.75.

$$Z_{INC\_infeas} = number\_violated\_rows \times 1000 + amount\_violation$$

$$Z_{INC\_infeas} = 2 \times 1000 + (0.5 + 0.25) = 2000.75$$

The artificial cost is calculated using Equation 6.2, with the squared number of periods multiplied by the sum of all job's weight.

$$artificial\_cost = {T_{max}}^2 \sum_{j=1}^{n} w_j \tag{6.2}$$

All the computational tests were run on a PC Intel Core i7 3610QM 2.3GHz and 8 GB RAM under MS Windows 7.

6.3 Compact model results

The results from the compact model implementation in SearchCol++, solved through the CPLEX callable libraries [ILOG, 2010], are presented in Table 6.2 for all the tested instances. Values for the LR of the compact model and the integer solutions (optimal or incumbents) are given (represented by Z_{LR} and Z_{INC} respectively,) as well as its respective integrality gap (represented by *gap*), and the time needed in seconds to solve the LR and to solve the integer problem. The status column will gives information about the optimality condition, or any given error while running the problem (the status codes are presented below Table 6.2).

All instances in this Chapter are represented by M-J (M machines and J jobs), or M-J-q with q representing the congestion level. In Table 6.2 the number of periods calculated, and used for each instance, is represented by P.

Table 6.2: Compact Model Results

			Value			Time (sec)		
M-J	q	P	Z_{LR}	Z_{INC}	*gap*(%)	Z_{LR}	Z_{INC}	Status *
2-20	1	68	0.131	0.18	27.07	0.049	0.7	101
2-30	1	99	0.593	0.69	14.02	0.167	3.5	101
2-40	1	134	0.468	0.59	20.75	0.410	6.7	101
2-50	1	166	0.431	0.56	23.03	0.847	10.0	101
4-30	1	51	0.038	0.06	36.11	0.090	1.1	101
4-40	1	67	0.066	0.07	5.10	0.214	2.2	101
4-50	1	84	0.045	0.08	43.75	0.424	5.4	101
4-60	1	99	0.088	0.17	48.04	0.704	11.1	101
6-40	1	54	0.000	0	0.00	0.220	2.6	101
6-50	1	68	0.070	0.08	12.50	0.443	5.3	101
6-70	1	94	0.010	0.01	0.00	1.165	15.1	101
8-40	1	51	0.000	0	0.00	0.260	3.1	101
8-60	1	76	0.000	0	0.00	0.885	9.5	101

* CPLEX Status:

101 Optimal Integer Solution Found

102 Optimal Solution within the tolerance Found

107 Time Limit Exceeded, Integer solution exists

108 Time Limit Exceeded, No Integer solution

109 Error termination, caused by memory, Integer Solution Exists

Table 6.2 – *from previous page*

			Value			Time (sec)		
M-J	q	P	Z_{LR}	Z_{INC}	$gap(\%)$	Z_{LR}	Z_{INC}	Status *
8-80	1	100	0.000	0	0.00	2.047	28.5	101
10-70	1	85	0.000	0	0.00	1.646	19.1	101
10-100	1	121	0.000	0	0.00	4.986	76.7	101
2-20	2	44	0.383	0.65	41.04	0.041	1.0	101
2-30	2	62	3.009	4.08	26.26	0.119	6.3	101
2-40	2	80	2.025	4.38	53.77	0.283	6.8	101
2-50	2	101	0.847	1.36	37.70	0.610	7.9	101
4-30	2	47	1.745	6.21	71.90	0.140	1.7	101
4-40	2	61	0.553	9.23	94.01	0.317	10.9	101
4-50	2	76	0.317	1.45	78.17	0.671	10.8	101
4-60	2	91	12.193	18.04	32.41	1.165	21.1	101
6-40	2	54	0.036	0.05	28.32	0.376	4.0	101
6-50	2	68	3.262	5.21	37.40	0.797	11.7	101
6-70	2	94	2.610	7.09	63.19	2.267	29.1	101
8-40	2	51	2.741	4.02	31.81	0.454	4.4	101
8-60	2	76	10.293	18.87	45.45	1.593	36.8	101
8-80	2	100	2.970	2.97	0.00	3.845	31.5	101
10-70	2	85	3.099	11	71.83	2.960	44.4	101
10-100	2	121	0.051	0.06	14.20	9.549	88.8	101
2-20	3	44	0.638	14.83	95.70	0.040	2.2	101
2-30	3	62	32.717	170.98	80.86	0.149	1674.4	102
2-40	3	80	3.615	189.17	98.09	0.287	1800.0	107
2-50	3	101	2.630	193.16	98.64	0.657	1800.0	107
4-30	3	47	36.057	94.27	61.75	0.196	134.0	101
4-40	3	61	27.385	90.59	69.77	0.320	1800.0	107
4-50	3	76	1.672	33.93	95.07	0.662	1437.8	109
4-60	3	91	28.121	111.58	74.80	1.175	1800.0	107
6-40	3	54	0.816	19.17	95.74	0.451	124.9	101
6-50	3	68	12.035	77.63	84.50	0.797	1800.0	107
6-70	3	94	10.420	68.58	84.81	2.274	1800.0	107
8-40	3	51	10.308	37.66	72.63	0.463	48.6	101
8-60	3	76	27.947	107.3	73.95	1.869	1639.8	109
8-80	3	100	3.921	20.15	80.54	3.925	1370.2	109
10-70	3	85	32.845	88.34	62.82	3.956	1800.0	107
10-100	3	121	8.552	69.66	87.72	13.156	1800.1	107
2-20	4	44	18.719	88.74	78.91	0.051	8.4	101
2-30	4	62	151.555	367.23	58.73	0.239	1800.0	107
2-40	4	80	126.841	515.83	75.41	0.942	1800.0	107
2-50	4	101	70.497	755.22	90.67	1.908	1800.0	107

* CPLEX Status:

101 Optimal Integer Solution Found

102 Optimal Solution within the tolerance Found

107 Time Limit Exceeded, Integer solution exists

108 Time Limit Exceeded, No Integer solution

109 Error termination, caused by memory, Integer Solution Exists

Table 6.2 – *from previous page*

			Value			Time (sec)		
M-J	q	P	Z_{LR}	Z_{INC}	$gap(\%)$	Z_{LR}	Z_{INC}	Status *
4-30	4	47	86.367	166.38	48.09	0.213	82.9	102
4-40	4	61	68.466	198.32	65.48	0.525	1349.7	102
4-50	4	76	6.448	170.17	96.21	1.241	1800.1	107
4-60	4	91	74.093	416.31	82.20	2.437	1800.0	107
6-40	4	54	29.969	100.35	70.14	0.706	471.2	102
6-50	4	68	76.411	232.79	67.18	1.585	1800.0	107
6-70	4	94	51.050	270.38	81.12	5.124	1800.0	107
8-40	4	51	42.970	94.06	54.32	0.578	124.2	101
8-60	4	76	85.130	224.81	62.13	4.492	1800.1	107
8-80	4	100	27.853	248.1	88.77	12.800	1800.1	107
10-70	4	85	81.840	217.95	62.45	7.489	1800.1	107
10-100	4	121	60.481	∞	-	37.917	1800.4	108
2-20	5	44	66.129	146.27	54.79	0.058	8.3	101
2-30	5	62	272.444	521.3	47.74	0.291	1800.0	107
2-40	5	80	322.453	764.09	57.80	1.148	1800.0	107
2-50	5	101	350.001	1246.43	71.92	2.995	1800.0	107
4-30	5	47	128.500	231.69	44.54	0.276	140.4	102
4-40	5	61	133.271	290.24	54.08	0.675	1641.2	102
4-50	5	76	67.775	319.44	78.78	1.912	1800.1	107
4-60	5	91	193.831	644.9	69.94	4.061	1800.0	107
6-40	5	54	72.627	163.56	55.60	0.619	354.6	102
6-50	5	68	156.755	347.59	54.90	1.385	1800.0	107
6-70	5	94	141.078	462.42	69.49	6.093	1800.2	107
8-40	5	51	66.956	116.82	42.68	0.555	29.4	101
8-60	5	76	162.005	316.96	48.89	3.250	1800.1	107
8-80	5	100	143.228	471.44	69.62	11.401	1800.2	107
10-70	5	85	157.629	317.82	50.40	6.376	1800.0	107
10-100	5	121	203.640	∞	-	34.421	1800.8	108

* CPLEX Status:

101 Optimal Integer Solution Found

102 Optimal Solution within the tolerance Found

107 Time Limit Exceeded, Integer solution exists

108 Time Limit Exceeded, No Integer solution

109 Error termination, caused by memory, Integer Solution Exists

As Table 6.3 demonstrates, for 97.5% of the tested instances, CPLEX was able to solve the problem to optimality and provided at least an integer solution for the remaining ones, except for two of the bigger tested instances where the time limit was reached before finding at least one integer solution.

The average gap calculation excludes the two instances where the compact model was unable to find an integer solution, which explains a shorter average gap for congestion

levels 4 and 5, when compared to level 3.

The increase in the average total time spent is not directly related to an increase in number of machines, jobs, periods or congestion level of the instance, but to a combination of these factors. The average time spent in each congestion level, indicates an accentuated rising in computational effort from instances of congestion level 2 to 3, which is visible by comparing computational times one by one, for instances with same number of jobs and machines.

Table 6.3: Overview of results of the Compact Model

	q=1	q=2	q=3	q=4	q=5	Total
Nº Optimal Solution	16	16	5	5	5	47
Nº Integer Solution	0	0	11	10	10	31
Nº No Integer Solution	0	0	0	1	1	2
Average Time Solution (sec.)	12.5	19.8	1302.0	1364.8	1373.5	814.5
Average $gap(\%)$	14.4	45.5	82.3	72.1	58.8	54.2

The number of periods in low congestion levels is typically higher, when comparing to instances with same characteristics but higher congestion levels. Problem instances with congestion level equal to 1 ($q = 1$) have a longer scheduling horizon, as applying the maximum function between the two values of Equation 6.1 (to calculate the number of periods), falls upon the maximum due date from the set of jobs J.

6.4 Decomposition model results

In this section, results for the decomposition model are presented and discussed, from a meaningful selection of executed tests. The selected tests will be presented following the succession of implementations.

Results from the decomposition model without any specific implementation were highly unsatisfactory. These tests were run without the introduction of initial solutions and using the CPLEX to solve the SPs. Despite being able to solve the linear relaxation for the problem, it failed at finding a feasible integer solution for the set of instances under study. The results are presented in Table C.1 in the Appendix.

Though this *pure* decomposition model solutions are not feasible, the LR values give good leads for the LR and lower bounds that could be provided by the CG.

In the next subsections, an analysis will be done of the results obtained from the initial solution's implementation, from the performance of the subproblem's heuristics, from the efficiency of the tested searchers, and finally, from the results obtained using the SearchCol perturbations.

6.4.1 Initial Solutions Implementation

Information on the typology of the heuristics is detailed in Subsection 5.2.1.

Results from the implementation of the initial solutions are presented in Table 6.4 with the best initial solution value for each instance (represented by Z) and its corresponding type.

Table 6.4: Values of Initial Solutions

	$q=1$		$q=2$		$q=3$		$q=4$		$q=5$	
M-J	Z	Type	Z	Type	Z	Type	Z	Type	Z	Type
2-20	0.28	2	0.82	3	79.64	3	142.52	2	219.39	3
2-30	1.04	4	12.72	5	276.61	3	491.62	3	652.06	2
2-40	1.25	3	13.21	1	324.31	2	706.34	2	1088.14	2, 3
2-50	1.16	2	2.89	4	449.1	2	1059.04	3	1511.43	3
4-30	0.29	4	28.06	5	145.66	2	208.03	2	269.31	2
4-40	0.56	5	37.99	5	141.25	3	267.77	2	372.49	2
4-50	0.47	4	3.05	3	69.03	3	249.78	3	450.45	2
4-60	4.49	3	35.11	3	250.49	5	639.34	3	838.14	2
6-40	0.22	3	2.83	5	77.77	3	153.92	3	214.2	3
6-50	0.41	5	18.45	5	170.91	2	293.36	2	406.14	3
6-70	0.47	4	21.52	5	146.52	5	403.22	3	582.24	2
8-40	0.14	5	16.13	1	83.22	5	120.9	3	162.44	2
8-60	5.2	5	35.02	5	168.3	5	305.11	2	382.29	2
8-80	0.21	5	7.55	5	127.96	5	389.52	3	555.05	3
10-70	0.28	5	27.94	5	174.24	5	350.79	2	433.83	2
10-100	0.39	3	7.62	5	119.85	5	558.28	3	756.97	3

Most lower congestion levels ($q = 1...3$) have better results using Type 5 of initial solutions. On the other hand, Type 5 has no better solution for the two highest congestion levels, which are fairly divided by Type 2 and Type 3. Type 3 of initial solutions is the most

regular among all congestion levels. Type 1 of initial solutions has the worst performance, as expected. This is an important remark of the importance of building heuristics OF oriented.

The instance with 2 jobs, 40 machines and congestion level 5, has the particularity of having two types with same solution value, not meaning the resulting schedule was the same for both heuristics.

In Table 6.5 the number of better solutions provided by each type of initial solutions is summarized, dividing its frequency by congestion level and with totals for each type (each type is represented by Type 1 through Type 5).

Table 6.5: Distribution of Type of Initial Solutions per Congestion Level

	Quantity					
	$q = 1$	$q = 2$	$q = 3$	$q = 4$	$q = 5$	Total
Type 1	0	2	0	0	0	2
Type 2	2	0	4	7	9	22
Type 3	4	3	5	9	6	27
Type 4	4	1	0	0	0	5
Type 5	6	10	7	0	0	23
Type 2 and 3	0	0	0	0	1	1

The time spent adding feasible schedules to the RMP is not substantial, considering five different types of solutions are added. The values are shown in Table 6.6 in average, considering all the congestion levels, for all pairs M-J of instances. The highest average time spent corresponds to the set of instances with 10 machines and 100 jobs in which almost 5.5 seconds ($1.1 second/type$) are needed to add the initial solutions. These values for the smaller instances are practically irrelevant.

Table 6.6: Time needed to add Initial Solutions (average in congestion levels)

M-J	Time (sec)	M-J	Time (sec)	M-J	Time (sec)	M-J	Time (sec)
2-20	0.013	4-30	0.032	6-40	0.116	8-60	0.677
2-30	0.022	4-40	0.083	6-50	0.254	8-80	1.816
2-40	0.051	4-50	0.174	6-70	0.878	10-70	1.440
2-50	0.118	4-60	0.367	8-40	0.148	10-100	5.408

All detailed results, with values from all initial solutions types and time spent running all initial heuristics for all instances are presented in Table C.2 of the Appendix.

6.4.2 Subproblems Resolution

In this work, three approaches are used to solve the SPs. Detailed information for each one of the methods is available in Subsection 5.2.2 of the previous Chapter.

Table 6.7: Linear Relaxation Values (Compact (C) and CG (D))

	$q=1$		$q=2$		$q=3$		$q=4$		$q=5$	
M-J	C	D	C	D	C	D	C	D	C	D
2-20	0.13	0.13	0.38	0.40	0.64	0.65	18.72	19.90	66.13	67.60
2-30	0.59	0.59	3.01	3.01	32.72	32.72	151.56	151.56	272.44	272.44
2-40	0.47	0.49	2.02	2.04	3.62	3.62	126.84	129.37	322.45	326.35
2-50	0.43	0.46	0.85	0.85	2.63	2.63	70.50	78.01	350.00	355.09
4-30	0.04	0.05	1.75	1.78	36.06	36.70	86.37	86.78	128.50	129.04
4-40	0.07	0.07	0.55	0.55	27.39	27.39	68.47	70.23	133.271	136.17
4-50	0.05	0.05	0.32	0.34	1.67	1.69	6.45	6.59	67.78	70.92
4-60	0.09	0.10	12.19	12.76	28.12	29.16	74.09	75.94	193.83	198.38
6-40	0.00	0.00	0.04	0.04	0.82	0.88	29.97	30.57	72.63	73.31
6-50	0.07	0.07	3.26	3.27	12.04	13.71	76.41	79.06	156.75	158.73
6-70	0.01	0.01	2.61	2.61	10.42	10.78	51.05	52.02	141.08	144.61
8-40	0.00	0.00	2.74	3.18	10.31	11.53	42.97	46.61	66.96	72.02
8-60	0.00	0.00	10.29	10.53	27.95	29.88	85.13	89.15	162.01	165.99
8-80	0.00	0.00	2.97	2.97	3.92	3.96	27.85	28.96	143.23	144.88
10-70	0.00	0.00	3.10	3.12	32.84	33.04	81.84	82.01	157.63	157.87
10-100	0.00	0.00	0.05	0.06	8.55	12.76	60.48	78.47	203.64	245.76

The values of the LR for comparison between the compact model and the decomposition model are shown in Table 6.7. The values presented correspond to the resolution of the SPs using the Global Heuristic, as it is has the better global results.

The lower bounds provided by the CG are generally better than the compact model, especially for the most congested instances and with higher number of machines. The instances are divided by congestion level, with the optimal values of the LR of the compact model (C) and CG from the decomposition model (D) being given for each instance.

The results from both heuristics are completely presented in Tables C.5 and C.6 in Appendix. A summarized version of these results is shown in Table 6.8 where E, I and G represent the exact SP resolution, Independent Heuristic and Global Heuristic respectively. These values were obtained considering the introduction of the initial solutions.

Table 6.8: Overview of decomposition implementations results

	$q=1$			$q=2$			$q=3$			$q=4$			$q=5$		
	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G
Nº of incumbent solutions	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
Nº of better incumbent sol. within E-I-G	0	0	15	0	1	2	0	1	0	0	0	0	0	0	0
Nº of improved initial solutions	0	4	15	0	1	2	0	1	0	0	0	0	0	0	0
Average time spent solving CG (sec)	583	138	271	631	238	396	624	527	632	557	481	538	494	442	505

Analysing the results overview it is possible to identify the Global Heuristic as the one providing a higher number of better incumbent solutions when comparing to the exact resolution and independent heuristic. This improvement in the number of better incumbent solutions has the counterpart of increasing the time needed to solve the CG, with the average time spent for instances with higher congestion levels, being close to the exact approach.

When solving the decomposition model using heuristics to solve the SPs, and without initial solutions, feasible solutions were not found for some instances, though this number of instances is much smaller than when solving the SPs exactly (see Table C.4 in Appendix).

The solution values without initial solutions for the exact resolution of the SPs are present in Table C.1 in the Appendix, as they are not fit for comparing with the heuristic values due to their infeasibility. Decomposition results for the heuristics without the initial solutions are also presented in the Appendix in Table C.3, with time and values of the solutions (Exact resolution, Independent Heuristic and Global Heuristic represented by MIP, I and G respectively).

The computational tests introduced implied the use of a searcher component to solve the integer problem. The MIP searcher was the chosen searcher, as it guaranteed the best performance. This will be discussed in the following subsection, where the computational tests used to analyse the searcher components are introduced.

6.4.3 Searchers

In previous subsections, the searcher component was already being tested through use of the MIP searcher. Besides MIP, the searcher VNS was also tested to analyse its performance when applied to our decomposition model.

Four different configurations of VNS were used regarding the configuration of the local

search component - VNS1 and VNS12 - and the constructor component.

The VNS was tested using a first descent strategy for local search, where the first neighbour that improves the current solution is selected (the alternative is the best descent strategy which selects the best neighbour). The different VNS searchers tested consist in using two different local search inputs for the type of neighbourhood being used. For VNS1 the neighbour solution has one modification and for VNS12 the neighbourhood of one solution is made of solutions with one and two modifications. Furthermore, for each searcher configuration two different constructors were also used: Higher Weights of CG (HW) and Incumbent (I) (see Subsection 5.3.2).

Information regarding the four tests are available in the Appendix (see Tables C.7 and C.8). In Table 6.9 an overview of the results is shown.

Table 6.9: Comparison of used searchers

	MIP	VNS1HW	VNS1I	VNS12HW	VNS12I
Number of better incumbent solutions	80	72	70	76	76
Number of improved initial solutions	43	37	33	39	40
Average time spent in Searcher (sec)	0.38	204.68	200.48	207.53	200.26

By comparing the MIP and both VNS searchers, MIP provides better results, particularly in the time spent to find an incumbent solution. The value of the incumbent solutions using MIP are generally better, as for all the 80 tested instances it assures the best incumbent solution value. The worst performance in number of better incumbent solutions is from VNS1. It must be noted that the subset of, for example, 72 and 70 better incumbent solutions in VNS1 are not necessarily included in the subset of 76 better incumbent solutions in VNS12. The same observations can be made when analysing if the solution value of each searcher improved the value of the best initial solution.

Between the different configurations of VNS, although the overview does not highlight it (as it is using average values), VNS1 provides faster search times than VNS12 when comparing the values for each instance. Furthermore, the value of the incumbent solutions are similar for either the constructors and the different neighbourhood type used for VNS.

6.4.4 Perturbators

The *perturbators* being used are Comb Prob and Comb Type0. For the other components, the CG is using the Global SP Heuristic and the search is conducted by the MIP searcher.

With *perturbator* Comb Prob being a stochastic component, three runs were made for each test instance, while for Comb Type0, a deterministic *perturbator*, only one run was executed.

In the appendix, in Tables C.9 and C.10, it is provided complete information of these computational tests. An overview of these results is available in Table 6.10.

Table 6.10: Overview of improvements using perturbators

Comb Type0		Comb Prob					
M-J-q	Imp. (%)	M-J-q	Imp. (%)	M-J-q	Imp. (%)	M-J-q	Imp. (%)
2-20-1	5.26	2-20-1	5.26	6-40-2	88.69	8-40-3	35.69
4-30-1	45.45	4-30-2	0.11	6-40-3	1.23	8-40-3	35.62
4-30-2	16.71	4-30-2	7.23	6-40-4	1.30	8-40-3	35.68
6-40-2	97.53	4-40-2	0.05	6-50-1	22.22	8-40-4	3.25
6-50-1	44.44	4-40-2	0.05	6-50-1	22.22	8-40-5	23.82
8-40-2	62.80	4-40-3	1.42	6-50-2	32.57	8-40-5	17.11
10-100-1	100.00	4-40-4	2.66	6-50-2	52.57	8-60-1	90.91
		4-50-2	9.84	6-50-3	0.56	8-60-2	0.29
		4-50-2	0.98	6-50-4	0.34	8-60-3	0.59
		4-60-2	0.74	8-40-2	68.38	8-60-5	0.78
		6-40-2	94.35	8-40-2	50.15	10-70-4	1.13
		6-40-2	94.35	8-40-2	74.83	10-70-4	5.36
Average	**53.17**					**Average**	**24.51**

Despite all 80 instances were tested, the summary shows only the subset of instances where an improvement was obtained compared to tests not using the *perturbators*. The instances where the incumbent solution value improved due to the *perturbations* are easily checked, by the number of total SearchCol iterations, as one of the stopping criteria is one iteration without improvement of the incumbent solution (besides the total time limit). Therefore, instances with a number of iterations of 2, for example, ran the *perturbator* once, solving the CG twice and not being able to improve the incumbent solution value from the first to its seconds iteration. All values of improvement are measured by comparing the value of the incumbent solution of the first CG with the last incumbent solution value.

By analysing the improvements obtained using perturbations, it's possible to verify the majority occurs in low congested instances. The Prob *perturbator* is the only one able to improve higher congestion level instances, improving also a larger number of instances. Considering also the best improvement for each instance using the Prob *perturbator* and comparing its level of improvements against the Comb Type 0 *perturbator*, the last one is able to provide bigger improvements although in a smaller number of instances (see Table 6.11).

Table 6.11: Comparison of improvements using perturbators

M-J-q	Type0 Imp. (%)	Prob Best Imp. (%)
2-20-1	5.26	5.26
4-30-1	45.45	0.00
4-30-2	16.71	0.11
4-40-2	0.00	0.05
4-40-3	0.00	1.42
4-40-4	0.00	2.66
4-50-2	0.00	9.84
4-60-2	0.00	0.74
6-40-2	97.53	94.35
6-40-3	0.00	1.23
6-40-4	0.00	1.30
6-50-1	44.44	22.22
6-50-2	0.00	52.57
6-50-3	0.00	0.56
6-50-4	0.00	0.34
8-40-2	62.80	74.83
8-40-3	0.00	35.69
8-40-4	0.00	3.25
8-40-5	0.00	23.82
8-60-1	0.00	90.91
8-60-2	0.00	0.29
8-60-3	0.00	0.59
8-60-5	0.00	0.78
10-70-4	0.00	5.36
10-100-1	100.00	0.00
Average	**14.89**	**17.13**

6.5 Comparison of the models

In Tables 6.12 and 6.13 the results from the computational tests using *perturbator Comb Type 0* (represented by 'Dec') are used to compare against the results of the compact model

(represented by 'Comp' and available in previous Section 6.3). For instances where a value is substituted for '*inf*', the corresponding model was not able to provide a feasible solution, thence the infeasible remark.

Table 6.12: Comparison of values between models

	$q=1$		$q=2$		$q=3$		$q=4$		$q=5$	
M-J	Comp	Dec	Comp	Dec	Comp	Dec	Comp	Dec	Comp	Dec
2-20	0.18	0.18	0.65	0.82	14.83	79.64	88.74	142.52	146.27	219.39
2-30	0.69	0.8	4.08	5.35	170.98	276.61	367.23	491.62	521.3	652.06
2-40	0.59	0.77	4.38	6.36	189.17	324.31	515.83	706.34	764.09	1088.14
2-50	0.56	1.16	1.36	2.89	193.16	449.1	755.22	1059.04	1246.43	1511.43
4-30	0.06	0.06	6.21	23.37	94.27	145.66	166.38	208.03	231.69	269.31
4-40	0.07	0.07	9.23	37.99	90.59	141.25	198.32	267.77	290.24	372.49
4-50	0.08	0.15	1.45	3.05	33.93	69.03	170.17	249.78	319.44	450.45
6-60	0.17	0.31	18.04	35.11	111.58	250.49	416.31	639.34	644.9	838.14
6-40	0	0	0.05	0.07	19.17	77.77	100.35	153.92	163.56	214.2
6-50	0.08	0.1	5.21	18.45	77.63	170.91	232.79	293.36	347.59	406.14
6-70	0.01	0.11	7.09	21.52	68.58	146.52	270.38	403.22	462.42	582.24
8-40	0	0	4.02	6	37.66	83.22	94.06	120.9	116.82	162.44
8-60	0	0	18.87	35.02	107.3	168.3	224.81	305.11	316.96	382.29
8-80	0	0	2.97	7.55	20.15	127.96	248.1	389.52	471.44	555.05
10-70	0	0	11	27.94	88.34	174.24	217.95	350.79	317.82	433.83
10-100	0	0	0.06	7.62	69.66	119.85	inf	558.28	inf	756.97
Average	0.16	0.23	5.92	14.94	86.69	175.30	-	396.22	-	555.91

Comparing solution values, the compact model offers a better performance, especially for instances of congestion level $q = 3$. On the other hand, for low congested instances, the decomposition approach using all implementations, matches for some instances, the optimal values of the compact model. The computational times have a different behaviour, as for higher congestion levels the compact model requires longer solving times than the decomposition approach, and for lower congestion levels the decomposition model spends a larger amount of time.

Although for most of the instances tested the compact model provides better results, for some instances, in particular the ones with more machines and jobs, and high congestion, SearchCol is able to provide a feasible solution while the compact model is not.

Table 6.13: Comparison of time spent between models (in seconds)

	$q=1$		$q=2$		$q=3$		$q=4$		$q=5$	
M-J	Comp	Dec	Comp	Dec	Comp	Dec	Comp	Dec	Comp	Dec
2-20	0.71	831.98	1.02	163.08	2.24	133.30	8.39	76.90	8.35	95.24
2-30	3.49	1813.77	6.35	1824.91	1674.40	1336.07	1799.99	305.65	1800.00	231.35
2-40	6.68	1836.25	6.83	1803.51	1800.00	1804.34	1800.00	1036.24	1800.00	617.80
2-50	9.99	1800.27	7.87	1808.10	1800.03	1806.30	1800.02	1808.49	1800.02	1806.72
4-30	1.10	181.71	1.73	581.69	134.03	218.43	82.86	128.74	140.42	127.79
4-40	2.22	70.50	10.88	874.97	1799.99	830.08	1349.75	560.17	1641.20	514.52
4-50	5.39	1809.31	10.82	1817.57	1437.82	1808.41	1800.08	1399.47	1800.06	1146.68
6-60	11.10	1809.31	21.09	1837.14	1800.03	1807.16	1800.01	1811.04	1800.02	1814.78
6-40	2.55	13.01	3.98	761.54	124.92	418.12	471.16	334.39	354.62	301.58
6-50	5.35	910.87	11.68	1146.37	1800.00	1338.28	1800.01	715.56	1800.01	659.86
6-70	15.11	1820.25	29.08	1831.15	1800.01	1823.46	1800.01	1824.11	1800.17	1813.94
8-40	3.12	8.72	4.43	461.91	48.59	367.78	124.17	205.13	29.39	204.23
8-60	9.47	428.37	36.76	1822.36	1639.79	1813.39	1800.06	1279.46	1800.08	990.73
8-80	28.48	73.95	31.54	1872.04	1370.18	1808.63	1800.08	1824.62	1800.25	1836.97
10-70	19.08	162.92	44.40	1816.83	1800.01	1834.90	1800.06	1829.84	1799.99	1650.89
10-100	76.66	569.69	88.75	1844.50	1800.12	1841.35	1800.37	1809.52	1800.83	1803.25
Average	12.53	883.81	19.83	1391.73	1302.01	1311.87	1364.81	1059.33	1373.46	976.02

Chapter 7

Conclusion

In this dissertation, the UPMSPjs was approached. A MIP model was proposed for the UPMSPjs, as well as a new approach to the problem based on a machine scheduling decomposition of the compact model using the Dantzig-Wolfe technique [Dantzig and Wolfe, 1960]. Furthermore, an application of the UPMSPjs to the SearchCol framework was developed, with problem specific algorithms implemented in SearchCol and validated through extensive computational tests.

The problem specific algorithms suggested in this work followed the succession of the global SearchCol algorithm. Several heuristics are presented for the UPMSPjs. First, a set of initial solutions was developed to provide the CG phase in SearchCol with efficient and feasible solutions that could accelerate the process and, at the same time, guarantee a good upper bound and a feasible final solution. The next step was to develop an efficient algorithm to solve the SPs. Two different approaches were used: one where the SPs were being solved one by one and another where the SPs were solved as a whole. Furthermore, the several featured general components of the Search and Perturbation phases were also tested, in order to improve final solution values for each tested instance. Finally, a three alternative algorithm was also suggested to be implemented for this last phase.

The performed computational tests showed that the UPMSPjs is a difficult problem, with its complexity rising exponentially with the number of jobs, machines, defined periods and also with the congestion level of each instance. The difficulty of the problem was observed in both models.

From the analysis of the computational tests, lower bounds provided by the LR of the

decomposition model are better than the ones provided by the compact model, although this happens at the cost of longer computational times (in average, the compact model's LR is solved in 1% of the time the CG needs to solve the LR). Also, in the LR of the decomposition model, lower bounds provided by the CG are not, in general, affected by the implementation of the heuristics to solve the SPs.

The implementation of the initial solutions provided feasible solutions for the problem, shortening the time spent in CG. Moreover, for the CG process, the implementation of problem specific algorithms improved the global algorithm in quality of solutions (using the Global Heuristic) and in computational times. The time spent solving the SPs was shorter using the Independent Heuristic, although its solutions were of poorer quality.

The testing of other SearchCol features indicate that for the type of decomposition developed in this work, the MIP searcher is the most efficient tool for the search phase and the featured *perturbators*, although improving a reasonable subset of instances, have margin for improvement. This improvement could potentially be obtained by implementing a specific perturbation algorithm. In this work, three alternatives were suggested regarding a *perturbator* for the UPMSPjs in SearchCol based in the characteristics of the problem and the developed model.

Comparing overall results, from both models for the set of chosen instances, the ones obtained from the compact model are satisfactory in efficiency and quality. For larger instances, both models find difficulties to solve the problem. The decomposition model requires a larger amount of time to solve the problems, when compared with the compact model. This happens mainly due to the time spent solving the CG, despite the improvement achieved using heuristics to solve the SPs. The solutions' values, when comparing both approaches, are better with the compact model, although specific implementations to the decomposition overall algorithm brought significant improvements, which can still go through further developments.

Within the scope of this work, it would be advisable to test the global and specific algorithms using a different set and type of instances. As the literature for the UPMSPjs using sequence independent setup times and job splitting is scarce, an instance generator would be needed, thence a first version of this generator is currently being developed to test new instances and suggested perturbation implementations.

Reckoning the needs in real-world situations of scheduling, the consideration and imple-

mentation of sequence dependent setup times in this model, would be the next reasonable task. By doing this, the model would not lose its generality, and the adaptation of the heuristics developed in this work would not be arduous, although its model implementation would require further analysis due to the time indexed formulation being used in this work. Equally important, the implementation of sequence dependent setup times would predictably raise the problem's complexity, requiring even larger computational times.

Finally, several components of SearchCol can still be tested. An important feature to be tested is the possibility of running the CG with a stopping criterion different from the one used in this work which would allow to diminish the time spent in CG. This could be achieved by defining an absolute or relative improvement of the primal solution as the CG stopping criterion. Testing these alternatives would enrich the continuity of this work and research for the UPMSPjs.

References

Akker, M. v. d., Hoogeveen, J., and Kempen, J. v. (2006). Parallel machine scheduling through column generation: Minimax objective functions. *Algorithms–ESA 2006*, pages 648–659.

Akker, M. v. d., Hoogeveen, J., and Velde, S. v. d. (1999). Parallel machine scheduling by column generation. *Operations Research*, 47(6):862–872.

Allahverdi, A., Gupta, J. N., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239.

Allahverdi, A., Ng, C., Cheng, T., and Kovalyov, M. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032.

Alvelos, F. (2012). *SearchCol++ User Manual Version 1.0*.

Alvelos, F., Sousa, A., and Santos, D. (2013). Combining column generation and metaheuristics. In Talbi, E.-G., editor, *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 285–334. Springer Berlin Heidelberg.

Anghinolfi, D. and Paolucci, M. (2007). Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. *Computers & Operations Research*, 34(11):3471–3490.

Armentano, V. and de Franca Filho, M. (2007). Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based grasp approach. *European Journal of Operational Research*, 183(1):100–114.

Azizoglu, M. and Webster, S. (2003). Scheduling parallel machines to minimize weighted flowtime with family set-up times. *International Journal of Production Research*, 41(6):1199–1215.

Barbosa, V., Respício, A., and Alvelos, F. (2013). A hybrid metaheuristic for the bus driver rostering problem. In *Proceedings of the 2nd International Conference on Operations Research and Enterprise Systems, 16-18 February 2013, Barcelona, Spain.*

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

Blum, C., Puchinger, J., Raidl, G. R., and Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: a survey. *Applied Soft Computing*, 11(6):4135–4151.

Boschetti, M., Maniezzo, V., and Roffilli, M. (2010). Decomposition techniques as metaheuristic frameworks. In *Matheuristics*, pages 135–158. Springer.

Chen, J.-F. (2009). Scheduling on unrelated parallel machines with sequence-and machine-dependent setup times and due-date constraints. *The International Journal of Advanced Manufacturing Technology*, 44(11):1204–1212.

Chen, J.-F. and Wu, T.-H. (2006). Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. *Omega*, 34(1):81 – 89.

Chen, Z.-L. and Powell, W. B. (1999a). A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. *European Journal of Operational Research*, 116(1):220–232.

Chen, Z.-L. and Powell, W. B. (1999b). Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing*, 11(1):78–94.

Cheng, T. E., Gupta, J. N., and Wang, G. (2000). A review of flowshop scheduling research with setup times. *Production and Operations Management*, 9(3):262–282.

Crauwels, H., Beullens, P., and Van Oudheusden, D. (2006). Parallel machine scheduling by family batching with sequence-independent set-up times. *International Journal of Operations Research*, 3(2):144–154.

Danna, E. and Le Pape, C. (2005). Branch-and-price heuristics: A case study on the vehicle routing problem with time windows. In *Column Generation*, pages 99–129. Springer.

Dantzig, G. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8:101–111.

Desaulniers, G., Desrosiers, J., and Solomon, M. (2005). *Column generation*, volume 5. Springer.

Desrosiers, J. and Lübbecke, M. (2005). A primer in column generation. In *Column generation*, pages 1–32. Springer.

Dunstall, S. and Wirth, A. (2005a). A comparison of branch-and-bound algorithms for a family scheduling problem with identical parallel machines. *European Journal of Operational Research*, 167(2):283–296.

Dunstall, S. and Wirth, A. (2005b). Heuristic methods for the identical parallel machine flowtime problem with set-up times. *Computers & Operations Research*, 32(9):2479–2491.

Fanjul-Peyro, L. and Ruiz, R. (2012). Scheduling unrelated parallel machines with optional machines and jobs selection. *Computers & Operations Research*, 39:1745–1753.

Gendreau, M. and Potvin, J.-Y. (2010). *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer US.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5(2):287–326.

Hansen, P., Mladenović, N., Brimberg, J., and Pérez, J. A. M. (2010). Variable neighborhood search. In *Handbook of Metaheuristics*, pages 61–86. Springer.

Hu, T. (1961). Parallel sequencing and assembly line. *Operations Research*, 9:841—-948.

ILOG (2010). *IBM ILOG CPLEX Optimization Studio V12.2*.

Jackson, S. M. (1955). *Scheduling a production line to minimize maximum tardiness. Management Science Research Project*. University of California, Los Angeles, USA.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:61–68.

Joo, C. and Kim, B. (2012). Parallel machine scheduling problem with ready times, due times and sequence-dependent setup times using meta-heuristic algorithms. *Engineering Optimization*, 44(9):1021–1034.

Kaplan, S. and Rabadi, G. (2011). Exact and heuristic algorithms for the aerial refueling parallel machine scheduling problem with due date-to-deadline window and ready times. *Computers & Industrial Engineering*.

Kedad-Sidhoum, S., Solis, Y. R., and Sourd, F. (2008). Lower bounds for the earliness–tardiness scheduling problem on parallel machines with distinct due dates. *European Journal of Operational Research*, 189(3):1305–1316.

Keha, A. B., Khowala, K., and Fowler, J. W. (2009). Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, 56(1):357–367.

Kim, D.-W., Kim, K.-H., Jang, W., and Frank Chen, F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 18(3):223–231.

Kim, Y., Shim, S., Kim, S., Choi, Y., and Yoon, H. (2004). Parallel machine scheduling considering a job-splitting property. *International Journal of Production Research*, 42(21):4531–4546.

Koulamas, C. (1997). Decomposition and hybrid simulated annealing heuristics for the parallel-machine total tardiness problem. *Naval Research Logistics (NRL)*, 44(1):109–125.

Lee, J.-H., Yu, J.-M., and Lee, D.-H. (2013). A tabu search algorithm for unrelated parallel machine scheduling with sequence- and machine-dependent setups: minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, pages 1–9.

Liaw, C.-F., Lin, Y.-K., Cheng, C.-Y., and Chen, M. (2003). Scheduling unrelated parallel machines to minimize total weighted tardiness. *Computers & Operations Research*, 30(12):1777–1789.

Lin, Y., Pfund, M., and Fowler, J. (2011). Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. *Computers & Operations Research*, 38:901–9016.

Little, J. D., Murty, K. G., Sweeney, D. W., and Karel, C. (1963). An algorithm for the traveling salesman problem. *Operations research*, 11(6):972–989.

Logendran, R., McDonell, B., and Smucker, B. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations Research*, 34(11):3420–3438.

Logendran, R. and Subur, F. (2004). Unrelated parallel machine scheduling with job splitting. *IIE Transactions*, 36(4):359–372.

Lopes, M. P. and Carvalho, J. d. (2007). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research*, 176(3):1508–1527.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science*, 6:1–12.

Nait, T. D., Yalaoui, F., Chu, C., and Amodeo, L. (2006). A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times. *International Journal of Production Economics*, 99(1):63–73.

Park, T., Lee, T., and Kim, C. O. (2012). Due-date scheduling on parallel machines with job splitting and sequence-dependent major/minor setup times. *The International Journal of Advanced Manufacturing Technology*, 59(1):325–333.

Pfund, M., Fowler, J. W., and Gupta, J. N. (2004). A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling problems. *Journal of the Chinese Institute of Industrial Engineers*, 21(3):230–241.

Pinedo, M. L. (2002). *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 2nd ed. 2002 edition.

Potts, C. N. and Kovalyov, M. Y. (2000). Scheduling with batching: a review. *European Journal of Operational Research*, 120(2):228–249.

Potts, C. N. and Wassenhove, L. N. V. (1992). Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity. *The Journal of the Operational Research Society*, 43(5):pp. 395–406.

Raidl, G. R. (2006). A unified view on hybrid metaheuristics. In Almeida, F., Aguilera, M., Blum, C., Moreno Vega, J. M., Perez, M., Roli, A., and Sampels, M., editors, *Hybrid Metaheuristics*. Springer.

Rocha, P., Ravetti, M., Mateus, G., and Pardalos, P. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, 35(4):1250–1264.

Rodriguez, F. J., Lozano, M., Blum, C., and García-Martínez, C. (2013). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, 40:1829–1841.

Sarıçiçek, . and Çelik, C. (2011). Two meta-heuristics for parallel machine scheduling with job splitting to minimize total tardiness. *Applied Mathematical Modelling*, 35(8):4117–4126.

Shim, S. and Kim, Y. (2008). A branch and bound algorithm for an identical parallel machine scheduling problem with a job splitting property. *Computers & Operations Research*, 35(3):863–875.

Shim, S.-O. and Kim, Y.-D. (2006). Minimizing total tardiness in an unrelated parallel-machine scheduling problem. *Journal of the Operational Research Society*, 58(3):346–354.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66.

Sourd, F. (2005). Earliness–tardiness scheduling with setup considerations. *Computers & Operations Research*, 32(7):1849–1865.

Talbi, E.-G. (2002). A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8:541–564.

Talbi, E.-G. (2009). *Metaheuristics*. John Wiley and Sons.

Talbi, E.-G. (2013). *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg.

Unlu, Y. and Mason, S. (2010). Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, 58(4):785–800.

Vallada, E. and Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622.

Wang, W.-L., Wang, H.-Y., Zhao, Y.-W., Zhang, L.-P., and Xu, X.-L. (2013). Parallel machine scheduling with splitting jobs by a hybrid differential evolution algorithm. *Computers & Operations Research*, 40(5):1196 – 1206.

Wilhelm, W. (2001). A technical review of column generation in integer programming. *Optimization and Engineering*, 2(2):159–200.

Xing, W. and Zhang, J. (2000). Parallel machine scheduling with splitting jobs. *Discrete Applied Mathematics*, 103(1):259–269.

Yalaoui, F. and Chu, C. (2003). An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times. *IIE Transactions*, 35(2):183–190.

Yang, W.-H. (1999). Survey of scheduling research involving setup times. *International Journal of Systems Science*, 30(2):143–155.

Zhu, Z. and Heady, R. (2000). Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. *Computers & Industrial Engineering*, 38(2):297–305.

Appendix A

Overview of Literature Review

Table A.1: Literature on PMS

Paper	α	γ	β	**Approach**
Akker et al. [1999]	P_m	$f(C_j)$	-	CG
Akker et al. [2006]	P_m	$Min\, X$	-	CG
Anghinolfi and Paolucci [2007]	P_m	$\sum T_j$	Sequence Dependent Setup; Ready Times	Hybrid Approach usin TS, SA and VNS
Armentano and de Franca Filho [2007]	Q_m	$\sum T_j$	Sequence Dependent Setup	GRASP
Azizoglu and Webster [2003]	P_m	$\sum F_j$	Family Setup Times	B&B Algorithms
Chen and Powell [1999a]	P_m	$\sum w_j E_j + \sum w_j T_j$	Due Dates	CG
Chen and Powell [1999b]	PMAC	X	Due Dates	CG
Chen and Wu [2006]	R_m	$\sum T_j$	Auxiliary Equipment (Setup and Processing Sequence Dependent)	Combines TA, TS and designed improvement procedures
Chen [2009]	R_m	$\sum T_j$	Sequence and machine dependent setup times and due-date constraints.	CombineSA, ATCS and designed improvement procedures
Crauwels et al. [2006]	P_m	X	Family Sequence Independent Setup;	Heuristic with job sequence construction as knapsack problem
Dunstall and Wirth [2005a]	P_m	$\sum w_j C_j$	Family Sequence Independent Setup;	Three different branching rules embedded on a B&B algorithm
Dunstall and Wirth [2005b]	P_m	$\sum w_j C_j$	Family Sequence Independent Setup;	Heuristics
Fanjul-Peyro and Ruiz [2012]	R_m	C_{max}	NAM; NAJ	2 MIP and 3 algorithms to combine with CPLEX

Continues to next page

Table A.1 – from previous page

Paper	α	γ	β	**Approach**
Joo and Kim [2012]	P_m	X	Sequence Dependent Setup; Due Dates; Ready Times	Two Genetic Algorithms and a Self-Evolution Algorithm
Kaplan and Rabadi [2011]	P_m	$\sum w_j T_j$	Ready Times; Deadline and Due Dates	MIP and Modified ATC
Kedad-Sidhoum et al. [2008]	P_m	$\sum E_j + \sum T_j$	Due dates	Lower and upper bounds
Kim et al. [2002]	R_m	$\sum T_{max\,i}$	Sequence-dependent setup times	Simulated Annealing
Kim et al. [2004]	P_m	$\sum T_j$	Job Splitting; Setup	Two-phase heuristic algorithm
Lee et al. [2013]	R_m	$\sum T_j$	Sequence Dependent Setup	TS algorithm
Liaw et al. [2003]	R_m	$\sum w_j T_j$	Due Dates	B&B
Lin et al. [2011]	R_m	$C_{max} + \sum w_j C_j + \sum w_j T_j$	-	Heuristics and GA
Logendran and Subur [2004]	R_m	$\sum w_j T_j$	Job Splitting	TS based heuristic
Logendran et al. [2007]	R_m	$\sum w_j T_j$	Sequence-dependent setups; dynamic release of jobs; dynamic availability of machines	Creation of 4 different initial solutions and TSS algorithms to improve
Nait et al. [2006]	P_m	C_{max}	Job splitting; Sequence-dependent Setup	LP and Improved Heuristic
Park et al. [2012]	P_m	$\sum T_j$	Job splitting; Sequence-dependent Setup (major/minor)	Heuristic considering job splitting and setup sequences
Lopes and Carvalho [2007]	R_m	$\sum w_j T_j$	Sequence-dependent Setup; Release date for jobs and machine	New B&P optimization algorithm
Rocha et al. [2008]	R_m	$C_{max} + \sum w_j T_j$	Sequence and Machine Dependent Setup Times, Due Dates, Weighted Jobs	Two MIP comparing to B&B algorithm with upper bound derived from GRASP
Rodriguez et al. [2013]	R_m	$\sum w_j C_j$	-	Iterated greedy algorithm
Sarıçiçek and Çelik [2011]	P_m	$\sum T_j$	Job Splitting; Sequence Independent Setup; Due Date	TS and Simulated Annealing
Shim and Kim [2006]	R_m	$\sum T_j$	Equal Due Dates	B&B
Shim and Kim [2008]	P_m	$\sum T_j$	Job Splitting; Equal Due Dates; Independent setup times	B&B
Sourd [2005]	1	$\sum E_j + \sum T_j$	Time-Indexed; Sequence Dependent Setup	B&B
Vallada and Ruiz [2011]	R_m	C_{max}	Job and Machine Sequence Dependent Setup	GA

Continues to next page

Table A.1 – from previous page

Paper	α	γ	β	**Approach**
Wang et al. [2013]	R_m; P_m	C_{max}	Job Splitting	Hybrid Differential Evolution
Xing and Zhang [2000]	P_m	C_{max}	Job Splitting; Sequence Independent Setup	Heuristic ML
Yalaoui and Chu [2003]	P_m	C_{max}	Job Splitting; Sequence Dependent Setup	Two-phase heuristic algorithm
Zhu and Heady [2000]	R_m	$\sum E_j + \sum T_j$	Sequence Dependent Setup; Due Dates	MIP

End of Table A.1

Appendix B

Developed Algorithms

Algorithm B.1: Type 1 of Initial Solutions

Input: A set I of machines i
Input: A set $J' = \{j'_1, j'_2, \ldots, j'_n\}$ of jobs j' with index number j ordered by a) increasing $d_{j'}$ b) non-increasing $w_{j'}$ c) increasing $r_{j'}$ d) increasing $j_{j'}$
Input: A set T of periods $t = \{0, ..., T_{max}\}$
Output: A set H' of scheduling plans h_i of type 1 for each machine i
for I **do**
  $marker_i = q_i$
end
for J' **do**
  for I **do**
    calculate $endperiod_i$ if setup and processing of j' in i starts at $marker_i$
  end
  $earliest = T_{max}$
  $chosen\,machine = 1$; **for** I **do**
    if $endperiod_i < T_{max}$ **then**
      $chosen\,machine = i$
      $earliest = endperiod_i$
    end
  end
  schedule j' in $chosen\,machine$
  $marker_{chosen\,machine} = earliest$
end
return H'

Algorithm B.2: Type 2 of Initial Solutions

Input: A set I of machines i
Input: A set $J' = \{j'_1, j'_2, \ldots, j'_n\}$ of jobs j' with index number j ordered by a) non-increasing $\bar{p_{ij'}}$ in i b) increasing $d_{j'}$ c) increasing $j_{j'}$
Input: A set T of periods $t = \{0, ..., T_{max}\}$
Output: A set H' of scheduling plans h_i of type 1 for each machine i
for J' **do**
  for I **do**
    calculate $ratio_i = \frac{available\_periods_i}{p_{ij'}}$
  end
  create subset I' of i with highest $ratio_i$
  if $size(I') \neq 1$ **then**
    $chosen\,machine = i \in I'$ with lowest $index_i$
  else if $size(I') == 1$ **then**
    $chosen\,machine = i \in I'$
  end
  for T **do**
    calculate $attractiveness_{j't}$
  end
  create set T' sorted by increasingly $attractiveness_{j't}$
  schedule j' in $chosen\,machine$ in first and available t from T';
end
return H'

Algorithm B.3: Type 3 of Initial Solutions

Input: A set I of machines i
Input: A set $J' = \{j'_1, j'_2, \ldots, j'_n\}$ of jobs j' with index number j ordered by a) non-increasing $\bar{p_{ij'}}$ in i b) increasing $d_{j'}$ c) increasing $j_{j'}$
Input: A set T of periods $t = \{0, ..., T_{max}\}$
Output: A set H' of scheduling plans h_i of type 1 for each machine i
for J' **do**
  for I **do**
    calculate $ratio_i = \frac{weight\_in\_OF\_available\_periods_i}{number\_available\_periods_i} \times p_{ij'}$
  end
  create subset I' of i with lowest $ratio_i$
  if $size(I') \neq 1$ **then**
    $chosen\,machine = i \in I'$ with lowest $index_i$
  else if $size(I') == 1$ **then**
    $chosen\,machine = i \in I'$
  end
  for T **do**
    calculate $attractiveness_{j't}$
  end
  create set T' sorted by increasingly $attractiveness_{j't}$
  schedule j' in $chosen\,machine$ in first and available t from T';
end
return H'

Algorithm B.4: Type 4 of Initial Solutions

Input: A set I of machines i
Input: A set $J' = \{j'_1, j'_2, \ldots, j'_n\}$ of jobs j' with index number j ordered by a) non-increasing $\bar{p_{ij'}}$ in i b) increasing $d_{j'}$ c) increasing $j_{j'}$
Input: A set T of periods $t = \{0, ..., T_{max}\}$
Output: A set H' of scheduling plans h_i of type 1 for each machine i

```
for J' do
    for I do
        calculate ratio_i = (weight_in_OF_available_periods_i / number_available_periods_i) × p_ij'
    end
    create subset I' of i with lowest ratio_i
    if size(I') ≠ 1 then
        chosen machine = i ∈ I' with lowest index_i
    else if size(I') == 1 then
        chosen machine = i ∈ I'
    end
    for T do
        calculate attractiveness_j't
    end
    create set T' sorted by increasingly attractiveness_j't
    while Σ_{i=1}^{m} Σ_{t=1}^{T_max} x_ij't / p_ij' < 1 do
        if first and available t from T' <= d'_j then
            schedule frac1p_ij' of j' in chosen machine in first and available t from T';
        else if first and available t from T' > d'_j then
            for I do
                calculate finishing_period_if_processing_ij'
            end
            chosen machine = i_min(finishing_period_if_processing_ij')
            schedule 1 / p_ij' of j' in chosen machine in first and available t from T;
        end
    end
end
return H'
```

Algorithm B.5: Type 5 of Initial Solutions

Input: A set I of machines i
Input: A set $J' = \{j'_1, j'_2, \ldots, j'_n\}$ of jobs j' with index number j ordered by a) increasing $d_{j'}$ b) non-increasing $\frac{w'_j \sum_i p_{ij'}}{m}$ c) increasing $j_{j'}$
Input: A set T of periods $t = \{0, ..., T_{max}\}$
Output: A set H' of scheduling plans h_i of type 1 for each machine i
for J' **do**
  for I **do**
    calculate $ratio_i = \frac{weight\_in\_OF\_available\_periods_i}{number\_available\_periods_i} \times p_{ij'}$
  end
  create subset I' of i with lowest $ratio_i$
  if $size(I') \neq 1$ **then**
    $chosen\,machine = i \in I'$ with lowest $index_i$
  else if $size(I') == 1$ **then**
    $chosen\,machine = i \in I'$
  end
  for T **do**
    calculate $attractiveness_{j't}$
  end
  create set T' sorted by increasingly $attractiveness_{j't}$
  schedule j' in $chosen\,machine$ in first and available t from T';
end
return H'

Algorithm B.6: Independent SP Heuristic

Input: Machine i corresponding to current SP
Input: A set $J' = \{j'_1, j'_2, \ldots, j'_n\}$ of jobs j' with index number j ordered by a) increasing of most negative 'cost': $cost_{x_{ijt}}$ b) increasing $\frac{d'_j}{w'_j(s_j + p_{ij})}$ c) increasing $j_{j'}$
Input: A set T of periods $t = \{0, ..., T_{max}\}$
Output: A scheduling plan h_i for machine i corresponding to SP being solved
while *available periods* **do**
  for J' **do**
    while $\sum_{t=1}^{T_{max}} \frac{x_{ij't}}{p_{ij'}} < 1$ & *available periods* & *no splitting needed for* j' **do**
      if *possible* **then**
        schedule a unit of pij' on available t with the most negative $cost_{x_{ijt}}$
      end
    end
  end
end
return H'

Algorithm B.7: Global SP Heuristic

Input: A set I of machines i
Input: A set $J' = \{j'_1, j'_2, \ldots, j'_n\}$ of jobs j' with index number j ordered by a) increasing of most negative 'cost': $cost_{x_{ijt}}$ b) increasing d'_j c) non-increasing w'_j d) increasing $j_{j'}$
Input: A set T of periods $t = \{0, ..., T_{max}\}$
Output: A set H'_i of scheduling plans h_i of type 1 for each machine i
for J' **do**
  sort $i \in I$ by increasing 'cost' of processing j' in available $t \rightarrow I'$
  for I' **do**
    sort $t \in T : \sum_{j'=1}^{n} x_{ij't} = 0 \wedge \sum_{j'=1}^{n} y_{ij't} = 0 \wedge t > r_{j'} \wedge t > q_i$ by increasing 'cost' $\rightarrow T'$
    $t' \rightarrow 0$
    while $\sum_{i=1}^{m} \sum_{t=1}^{T_{max}} \frac{x_{ij't}}{p_{ij'}} < 1$ & $t' \leq T'_{max}$ **do**
      if *'cost' of* $t' \leq 0$ **then**
        schedule a unit of pij' on available t with the most negative $cost_{x_{ijt}}$
        $t'++$
      else if *'cost' of* $t' > 0$ **then**
        $t'++$
      end
    end
  end
  `// processing will be allowed on periods with positive 'cost'`
  for I' **do**
    sort $t \in T : \sum_{j'=1}^{n} x_{ij't} = 0 \wedge \sum_{j'=1}^{n} y_{ij't} = 0 \wedge t > r_{j'} \wedge t > q_i$ by increasing 'cost' $\rightarrow T'$
    $t' \rightarrow 0$
    while $\sum_{i=1}^{m} \sum_{t=1}^{T_{max}} \frac{x_{ij't}}{p_{ij'}} < 1$ & $t' \leq T'_{max}$ **do**
      schedule a unit of pij' on available t with the most negative $cost_{x_{ijt}}$
      $t'++$
    end
  end
end
return H'

Appendix C

Extended Computational Results

The notation used in the following tables of results follow the same principle of Chapter 6:

Z_{LR} - Value of the optimal LR solution;

Z_{INC_feas} - Feasible value of the incumbent solution;

Z_{INC_infeas} - Infeasible value of the incumbent solution;

Z_{INC} - Value of the incumbent solution;

Load - Time spent loading the decomposition model;

In Table C.3:

MIP - Resolution of SP using exact method;

I - Resolution of SP using Independent Heuristic;

G - Resolution of SP using Global Heuristic;

Iteration in Tables C.9 and C.10 refer to SearchCol algorithm iterations.

Table C.1: Decomposition Results

	Value				Time (sec.)			
M-J	Z_{LR}	Z_{INC_feas}	Z_{INC_infeas}		Load	CG	Searcher	Total
2-20-1	0.13	0.14	3002.2		0.000	54.8	0.3	55.1
2-30-1	0.59	0.84	5004.5		0.015	379.3	1.0	380.3
2-40-1	0.49	0.57	9005.0		0.021	1653.6	4.6	1658.2
2-50-1	0.83	1.20	22018.7		0.031	1806.2	0.0	1806.2
4-30-1	0.05	0.07	5001.5		0.004	72.3	5.1	77.4
4-40-1	0.07	0.24	4002.2		0.009	219.8	17.7	237.5
4-50-1	0.05	0.12	7003.5		0.015	567.2	134.1	701.3
4-60-1	0.10	0.15	10005.0		0.029	1318.4	476.2	1794.6
6-40-1	0.00	1.67	1000.3		0.009	127.5	694.4	821.9
6-50-1	0.07	2.94	3002.2		0.031	359.2	924.4	1283.6
6-70-1	0.01	0.06	32024.0	*	0.048	-1.0	-1.0	-1.0
8-40-1	0.00	0.00	20016.7	*	0.015	-1.0	-1.0	-1.0
8-60-1	0.00	0.01	19011.3	*	0.037	-1.0	-1.0	-1.0
8-80-1	0.10	0.27	19009.5		0.091	1802.0	0.0	1802.0
10-70-1	0.00	0.99	56051.5	*	0.237	-1.0	-1.0	-1.0
10-100-1	61.17	0.00	100100.0		0.234	1807.6	0.0	1807.6
2-20-2	0.40	0.34	5005.0		0.000	21.1	0.2	21.3
2-30-2	3.01	0.86	9007.2		0.000	117.3	0.5	117.8
2-40-2	2.04	0.84	11009.5		0.016	451.2	1.1	452.3
2-50-2	0.85	0.96	15012.2		0.013	1093.9	8.9	1102.7
4-30-2	1.78	1.29	7005.7		0.000	57.4	0.7	58.1
4-40-2	0.55	0.21	6004.3		0.015	207.5	1.9	209.4
4-50-2	0.34	0.29	12008.3		0.020	544.6	24.0	568.5
4-60-2	12.76	13.02	11007.5		0.034	1246.3	127.4	1373.7
6-40-2	0.04	0.05	4002.2		0.000	161.8	99.4	261.2
6-50-2	3.27	2.12	7005.7		0.024	408.4	225.9	634.2
6-70-2	2.61	3.25	10007.8		0.063	1735.1	65.4	1800.5
8-40-2	3.18	3.04	2002.0		0.016	168.8	5.4	174.3

* Out of memory in search in the root

Continues to next page

Table C.1 – from previous page

	Value				Time (sec.)		
M-J-q	Z_{LR}	Z_{INC_feas}	Z_{INC_infeas}	Load	CG	Searcher	Total
8-60-2	10.53	4.09	9006.2	0.048	795.2	873.3	1668.5
8-80-2	3.27	5.49	24017.7	0.092	1803.5	0.0	1803.6
10-70-2	3.12	3.18	28023.5 *	0.088	-1.0	-1.0	-1.0
10-100-2	906.22	0.00	100100.0	0.170	1802.5	0.0	1802.5
2-20-3	0.65	0.43	9008.3	0.000	18.5	0.0	18.6
2-30-3	32.72	16.88	14013.3	0.003	89.0	0.1	89.1
2-40-3	3.62	1.46	17016.5	0.008	328.6	0.4	328.9
2-50-3	2.63	1.55	22020.8	0.032	920.6	2.3	922.9
4-30-3	36.70	18.35	12012.0	0.015	50.8	0.1	51.0
4-40-3	27.38	23.01	12011.5	0.008	182.8	0.3	183.1
4-50-3	1.69	0.54	15013.7	0.016	523.3	2.9	526.2
4-60-3	29.16	16.33	19017.3	0.031	1087.9	18.4	1106.3
6-40-3	0.88	0.30	10008.5	0.012	170.7	3.6	174.3
6-50-3	13.71	12.24	16015.5	0.023	399.4	1.1	400.5
6-70-3	10.78	6.18	16015.5	0.062	1782.3	18.0	1800.3
8-40-3	11.53	7.99	10009.5	0.015	151.9	0.2	152.1
8-60-3	29.88	17.26	20018.2	0.053	775.3	6.7	782.0
8-80-3	4.74	4.78	43042.3	0.094	1805.9	0.0	1805.9
10-70-3	33.04	26.29	22021.5	0.094	1421.5	97.3	1518.8
10-100-3	1799.54	0.00	100100.0	0.171	1805.7	0.0	1805.7
2-20-4	19.90	108.68	12011.7	0.000	13.4	0.1	13.5
2-30-4	151.56	69.76	19019.0	0.000	63.8	0.0	63.8
2-40-4	129.37	64.01	28028.0	0.008	251.4	0.1	251.5
2-50-4	78.01	42.84	37036.0	0.013	758.5	0.1	758.6
4-30-4	86.78	122.90	17017.0	0.000	44.3	0.1	44.4
4-40-4	70.23	29.30	23022.0	0.007	158.6	0.1	158.7
4-50-4	6.59	2.78	28027.5	0.021	396.5	0.2	396.8
4-60-4	75.94	56.32	32032.0	0.034	932.9	0.4	933.3
6-40-4	30.57	7.63	20020.0	0.011	127.2	0.2	127.4

* Out of memory in search in the root

Continues to next page

Table C.1 – from previous page

	Value			Time (sec.)			
M-J-q	Z_{LR}	Z_{INC_feas}	Z_{INC_infeas}	Load	CG	Searcher	Total
6-50-4	79.06	33.40	29028.5	0.018	324.7	0.2	324.9
6-70-4	52.02	38.13	33033.0	0.063	1516.6	2.3	1518.9
8-40-4	46.61	23.01	18018.0	0.000	128.5	0.2	128.6
8-60-4	89.15	42.29	32032.0	0.051	722.0	0.3	722.3
8-80-4	29.61	38.23	51051.0	0.093	1810.8	0.0	1810.8
10-70-4	82.01	390.52	33032.5	0.085	1185.5	0.2	1185.7
10-100-4	2225.60	0.00	100100.0	0.175	1810.2	0.0	1810.2
2-20-5	67.60	48.99	12012.0	0.000	14.6	0.0	14.6
2-30-5	272.44	98.10	22021.5	0.000	50.2	0.1	50.3
2-40-5	326.35	178.00	30029.5	0.000	229.3	0.0	229.4
2-50-5	355.09	221.18	39039.0	0.012	613.9	0.0	613.9
4-30-5	129.04	99.20	17017.0	0.003	38.2	0.0	38.2
4-40-5	136.17	64.07	25025.0	0.007	132.6	0.1	132.7
4-50-5	70.92	29.54	34034.0	0.016	367.5	0.1	367.7
4-60-5	198.38	111.78	43041.8	0.016	754.8	0.1	754.9
6-40-5	73.31	39.03	22022.0	0.016	125.0	0.1	125.1
6-50-5	158.73	87.81	32032.0	0.015	291.9	0.1	292.0
6-70-5	144.61	100.04	45045.0	0.060	1235.0	0.1	1235.2
8-40-5	72.02	67.54	19019.0	0.013	137.6	0.1	137.7
8-60-5	165.99	98.43	35035.0	0.031	629.4	0.1	629.6
8-80-5	145.23	174.18	65065.0	0.088	1816.2	0.0	1816.2
10-70-5	157.87	91.44	38038.0	0.063	1171.1	0.2	1171.3
10-100-5	2585.94	621.92	98098.0	0.178	1803.1	0.0	1803.1

* Out of memory in search in the root

End of Table C.1

Table C.2: Initial Solutions' Complete Values

M-J-q	Z_{Type1}	Z_{Type2}	Z_{Type3}	Z_{Type4}	Z_{Type5}	Time (sec.)
2-20-1	17.51	0.28	0.58	0.49	0.61	0.011

Continues to next page

Table C.2 – from previous page

M-J-q	Z_{Type1}	Z_{Type2}	Z_{Type3}	Z_{Type4}	Z_{Type5}	Time (sec.)
2-20-2	4.04	0.98	0.82	1.08	2.11	0.016
2-20-3	104.17	96.13	79.64	127.76	96.55	0.015
2-20-4	322.77	142.52	155.21	210.7	203.25	0.016
2-20-5	357.44	228.31	219.39	352.99	285.36	0.007
2-30-1	32.45	10.99	3.89	1.04	6.75	0.035
2-30-2	16.41	23.59	36.52	95.93	12.72	0.016
2-30-3	469.32	313.00	276.61	352.61	337.63	0.019
2-30-4	924.66	491.76	491.62	755.41	623.70	0.016
2-30-5	1336.50	652.06	685.47	805.96	884.07	0.024
2-40-1	56.42	5.05	1.25	123.92	1.35	0.078
2-40-2	13.21	49.74	16.98	64.31	15.14	0.031
2-40-3	807.87	324.31	405.43	825.4	531.32	0.045
2-40-4	1431.54	706.34	710.30	1030.3	1230.60	0.054
2-40-5	2062.17	1088.14	1088.14	1641.67	1574.11	0.047
2-50-1	107.7	1.16	1.59	1.26	1.6	0.181
2-50-2	17.46	4.64	6.94	2.89	8.87	0.068
2-50-3	980.39	449.10	458.28	1412.53	604.54	0.094
2-50-4	3070.00	1063.00	1059.04	1503.39	1617.70	0.117
2-50-5	2912.61	1511.46	1511.43	2204.16	2287.92	0.132
4-30-1	11.25	0.36	1.41	0.29	0.32	0.028
4-30-2	30.68	53.88	69.80	120.2	28.06	0.015
4-30-3	283.17	145.66	154.57	312.06	169.31	0.034
4-30-4	483.15	208.03	244.55	519.82	287.10	0.047
4-30-5	491.05	269.31	274.26	521.8	358.38	0.038
4-40-1	21.65	2.38	0.78	23.51	0.56	0.067
4-40-2	46.84	85.75	46.39	265.92	37.99	0.063
4-40-3	339.57	159.95	141.25	257.16	235.62	0.088
4-40-4	578.16	267.77	331.54	472.83	414.82	0.096
4-40-5	785.08	372.49	402.30	698.22	573.22	0.100
4-50-1	47.56	4.71	3.8	0.47	0.63	0.130
4-50-2	14.71	52.42	3.05	75.29	10.63	0.133

Continues to next page

Table C.2 – from previous page

M-J-q	Z_{Type1}	Z_{Type2}	Z_{Type3}	Z_{Type4}	Z_{Type5}	Time (sec.)
4-50-3	185.63	104.02	69.03	271.43	81.09	0.182
4-50-4	738.62	279.52	249.78	463.72	372.43	0.207
4-50-5	1058.33	450.45	469.48	862.03	635.58	0.218
4-60-1	60.41	27.5	4.49	659.87	16.66	0.240
4-60-2	37.00	145.75	35.11	402.39	42.99	0.382
4-60-3	519.78	354.67	313.95	937.14	250.49	0.367
4-60-4	1023.68	640.49	639.34	1086.38	755.39	0.405
4-60-5	1556.33	838.14	909.45	1630.09	1077.14	0.439
6-40-1	16.25	0.41	0.22	0.26	0.38	0.078
6-40-2	6.69	14.86	18.63	67.1	2.83	0.101
6-40-3	195.13	95.82	77.77	222.4	81.36	0.121
6-40-4	265.47	168.62	153.92	310.16	229.74	0.140
6-40-5	477.25	235.87	214.20	386.29	304.95	0.140
6-50-1	25.05	8.52	4.4	0.48	0.41	0.154
6-50-2	24.29	57.59	41.25	197.75	18.45	0.218
6-50-3	333.68	170.91	172.91	441.14	174.39	0.274
6-50-4	695.04	293.36	300.24	687.6	392.04	0.296
6-50-5	793.98	407.13	406.14	865.5	511.83	0.330
6-70-1	39.52	10.51	3.33	0.47	0.56	0.502
6-70-2	33.09	40.42	33.32	441.12	21.52	0.771
6-70-3	162.41	197.54	217.40	781.74	146.52	0.964
6-70-4	701.94	407.30	403.22	1165.73	428.67	1.029
6-70-5	894.96	582.24	626.83	1290.16	680.13	1.125
8-40-1	9.35	7.02	3.11	26.91	0.14	0.104
8-40-2	16.13	51.86	39.89	110.96	18.19	0.136
8-40-3	99.99	95.28	98.24	134.65	83.22	0.158
8-40-4	201.96	128.80	120.90	211.96	165.33	0.171
8-40-5	282.15	162.44	164.42	250.47	198.00	0.172
8-60-1	22.91	21.05	17.08	206.04	5.2	0.468
8-60-2	50.75	65.74	53.77	212.35	35.02	0.637
8-60-3	276.21	203.21	220.16	446.78	168.30	0.718

Continues to next page

Table C.2 – from previous page

M-J-q	Z_{Type1}	Z_{Type2}	Z_{Type3}	Z_{Type4}	Z_{Type5}	Time (sec.)
8-60-4	587.07	305.11	308.08	644.51	360.36	0.767
8-60-5	705.87	382.29	403.00	647.46	483.12	0.796
8-80-1	55.51	25.62	11.39	173.57	0.21	1.029
8-80-2	30.98	62.47	35.22	238.78	7.55	1.560
8-80-3	207.21	185.83	164.90	652.09	127.96	2.230
8-80-4	775.19	453.04	389.52	871.76	485.10	2.124
8-80-5	951.39	557.05	555.05	1066.69	805.86	2.137
10-70-1	28.7	9.52	11.19	45.2	0.28	0.979
10-70-2	44.58	83.87	81.10	285.67	27.94	1.343
10-70-3	244.59	252.23	206.62	521.93	174.24	1.564
10-70-4	560.36	350.79	358.64	814.95	396.00	1.627
10-70-5	722.70	433.83	448.56	742.54	512.82	1.687
10-100-1	55.01	0.82	0.39	0.77	0.41	3.420
10-100-2	17.31	18.98	21.73	402.5	7.62	5.010
10-100-3	159.60	258.86	248.34	956.27	119.85	5.849
10-100-4	967.23	611.53	558.28	1276.65	572.22	6.146
10-100-5	1228.61	778.86	756.97	1488.16	924.66	6.617

End of Table C.2

Table C.3: Results of the Heuristics to solve SP without Initial Solutions

	Time (sec.)									Value			
	CG			Searcher			Total			Z_{feas}		Z_{infeas}	
M-J-q	MIP	I	G	MIP	I	G	MIP	I	G	I	G	I	G
2-20-1	54.8	43.9	43.4	0.3	0.0	0.0	55.1	44.0	43.5	0.18	0.18	0.00	0.00
2-20-2	21.1	19.6	19.1	0.2	0.0	0.0	21.3	19.6	19.1	1.04	1.04	0.00	0.00
2-20-3	18.5	22.9	22.3	0.0	0.0	0.0	18.6	22.9	22.3	68.50	68.50	0.00	0.00
2-20-4	13.4	16.2	16.8	0.1	0.0	0.0	13.5	16.3	16.8	156.88	156.88	0.00	0.00
2-20-5	14.6	14.9	14.7	0.0	0.0	0.0	14.6	15.0	14.7	222.38	222.38	0.00	0.00
2-30-1	379.3	280.8	282.6	1.0	0.1	0.1	380.3	280.9	282.7	0.74	0.74	0.00	0.00
2-30-2	117.3	100.1	99.6	0.5	0.2	0.2	117.8	100.3	99.8	63.05	63.05	0.00	0.00
2-30-3	89.0	92.1	92.3	0.1	0.1	0.0	89.1	92.2	92.3	639.93	639.93	0.00	0.00
2-30-4	63.8	67.7	67.3	0.0	0.1	0.1	63.8	67.7	67.4	996.21	996.21	0.00	0.00
2-30-5	50.2	57.1	56.7	0.1	0.0	0.1	50.3	57.1	56.8	787.32	787.32	0.00	0.00
2-40-1	1653.6	1204.4	1207.9	4.6	0.2	0.2	1658.2	1204.5	1208.1	0.72	0.72	0.00	0.00
2-40-2	451.2	384.9	385.4	1.1	0.3	0.2	452.3	385.2	385.6	15.28	15.28	0.00	0.00

Continues to next page

Table C.3 – from previous page

	Time (sec.)									Value			
	CG			Searcher			Total			Z_{feas}		Z_{infeas}	
M-J-q	MIP	I	G	MIP	I	G	MIP	I	G	I	G	I	G
2-40-3	328.6	341.2	341.1	0.4	0.1	0.1	328.9	341.3	341.1	674.05	674.05	0.00	0.00
2-40-4	251.4	238.7	238.9	0.1	0.1	0.1	251.5	238.8	239.0	1107.45	1107.45	0.00	0.00
2-40-5	229.3	219.3	220.2	0.0	0.1	0.1	229.4	219.4	220.3	1483.06	1483.06	0.00	0.00
2-50-1	1806.2	1803.3	1801.4	0.0	0.0	0.0	1806.2	1803.3	1801.4	0.43	0.55	14012.17	10007.67
2-50-2	1093.9	951.6	953.2	8.9	3.4	3.5	1102.7	955.0	956.7	4.41	4.41	0.00	0.00
2-50-3	920.6	885.8	887.8	2.3	0.3	0.3	922.9	886.2	888.1	1600.86	1600.86	1000.67	1000.67
2-50-4	758.5	714.2	716.4	0.1	0.1	0.1	758.6	714.3	716.5	1409.92	1409.92	0.00	0.00
2-50-5	613.9	603.0	604.0	0.0	0.1	0.1	613.9	603.1	604.1	2446.65	2446.65	0.00	0.00
4-30-1	72.3	33.9	34.7	5.1	0.1	0.1	77.4	34.0	34.8	0.09	0.09	0.00	0.00
4-30-2	57.4	55.9	55.7	0.7	0.1	0.1	58.1	56.1	55.9	70.47	70.47	0.00	0.00
4-30-3	50.8	47.4	47.9	0.1	0.2	0.2	51.0	47.5	48.1	209.93	209.93	0.00	0.00
4-30-4	44.3	38.2	38.2	0.1	0.2	0.1	44.4	38.4	38.4	262.37	262.37	0.00	0.00
4-30-5	38.2	39.8	39.3	0.0	0.0	0.0	38.2	39.9	39.3	301.95	301.95	0.00	0.00
4-40-1	219.8	87.4	87.0	17.7	-1.0	-1.0	237.5	87.4	87.0	0.07	0.07	0.00	0.00
4-40-2	207.5	167.9	168.5	1.9	0.4	0.3	209.4	168.3	168.9	49.11	49.11	0.00	0.00
4-40-3	182.8	165.7	164.3	0.3	0.2	0.2	183.1	165.9	164.5	256.82	256.82	0.00	0.00
4-40-4	158.6	142.5	142.8	0.1	0.1	0.1	158.7	142.7	143.0	481.20	481.20	0.00	0.00
4-40-5	132.6	111.9	111.5	0.1	0.1	0.2	132.7	112.0	111.6	586.21	586.21	0.00	0.00
4-50-1	567.2	237.2	238.1	134.1	0.3	0.3	701.3	237.5	238.4	0.17	0.17	0.00	0.00
4-50-2	544.6	408.9	408.4	24.0	1.4	1.4	568.5	410.2	409.7	8.33	8.33	0.00	0.00
4-50-3	523.3	507.7	506.7	2.9	1.1	1.2	526.2	508.8	507.8	189.03	189.03	0.00	0.00
4-50-4	396.5	359.3	359.0	0.2	0.2	0.3	396.8	359.5	359.3	578.19	578.19	0.00	0.00
4-50-5	367.5	325.9	326.1	0.1	0.2	0.2	367.7	326.2	326.4	659.34	659.34	0.00	0.00
4-60-1	1318.4	711.6	712.0	476.2	0.6	0.6	1794.6	712.2	712.5	0.31	0.31	0.00	0.00
4-60-2	1246.3	1148.8	1151.8	127.4	6.5	6.6	1373.7	1155.4	1158.4	31.59	31.59	0.00	0.00
4-60-3	1087.9	1046.2	1045.7	18.4	1.6	1.6	1106.3	1047.9	1047.4	693.57	693.57	0.00	0.00
4-60-4	932.9	881.8	883.5	0.4	0.5	0.5	933.3	882.3	884.0	948.87	948.87	0.00	0.00
4-60-5	754.8	659.6	661.6	0.1	0.2	0.2	754.9	659.8	661.7	1176.12	1176.12	0.00	0.00
6-40-1	127.5	43.9	43.8	694.4	0.1	0.1	821.9	44.0	43.8	0.00	0.00	0.00	0.00
6-40-2	161.8	87.5	86.9	99.4	0.2	0.2	261.2	87.7	87.2	0.10	0.10	0.00	0.00
6-40-3	170.7	140.2	140.5	3.6	0.3	0.3	174.3	140.5	140.8	96.45	96.45	0.00	0.00
6-40-4	127.2	96.4	97.0	0.2	0.3	0.2	127.4	96.7	97.2	192.42	192.42	0.00	0.00
6-40-5	125.0	92.5	93.1	0.1	0.2	0.2	125.1	92.7	93.3	272.27	272.27	0.00	0.00
6-50-1	359.2	144.3	143.9	924.4	0.2	0.2	1283.6	144.5	144.1	0.15	0.15	0.00	0.00
6-50-2	408.4	242.5	242.0	225.9	0.7	0.7	634.2	243.2	242.7	10.88	10.88	0.00	0.00
6-50-3	399.4	348.9	348.4	1.1	0.5	0.5	400.5	349.4	348.9	361.50	361.50	0.00	0.00
6-50-4	324.7	275.3	275.3	0.2	0.5	0.5	324.9	275.7	275.8	485.10	485.10	0.00	0.00
6-50-5	291.9	222.2	223.9	0.1	0.4	0.4	292.0	222.6	224.3	590.04	590.04	0.00	0.00
6-70-1	1399.2	435.3	435.5	289.7	0.3	0.3	-1.0	435.7	435.8	0.05	0.05	0.00	0.00
6-70-2	1735.1	1199.3	1193.7	65.4	2.4	2.4	1800.5	1201.9	1196.1	15.24	15.24	0.00	0.00
6-70-3	1782.3	1387.7	1368.1	18.0	2.6	2.6	1800.3	1390.3	1370.7	258.51	258.51	0.00	0.00
6-70-4	1516.6	1175.9	1167.7	2.3	0.6	0.6	1518.9	1176.5	1168.4	554.40	554.40	0.00	0.00
6-70-5	1235.0	889.4	890.8	0.1	0.5	0.5	1235.2	890.0	891.4	721.71	721.71	0.00	0.00

Continues to next page

Table C.3 – from previous page

	Time (sec.)									Value			
	CG			Searcher			Total			Z_{feas}		Z_{infeas}	
M-J-q	MIP	I	G	MIP	I	G	MIP	I	G	I	G	I	G
8-40-1	140.7	40.7	39.8	261.4	-1.0	-1.0	-1.0	40.8	39.8	0.00	0.00	0.00	0.00
8-40-2	168.8	87.7	88.5	5.4	0.3	0.3	174.3	88.0	88.8	7.96	7.96	0.00	0.00
8-40-3	151.9	110.3	109.9	0.2	0.4	0.3	152.1	110.7	110.2	77.25	77.25	0.00	0.00
8-40-4	128.5	91.6	92.1	0.2	0.3	0.2	128.6	91.9	92.3	135.67	135.67	0.00	0.00
8-40-5	137.6	85.8	86.1	0.1	0.3	0.2	137.7	86.1	86.3	144.54	144.54	0.00	0.00
8-60-1	700.6	208.5	210.5	320.8	0.3	0.1	-1.0	208.8	210.7	0.04	0.04	0.00	0.00
8-60-2	795.2	479.1	482.4	873.3	1.5	1.5	1668.5	480.7	483.9	33.91	33.91	0.00	0.00
8-60-3	775.3	567.2	565.6	6.7	1.8	2.1	782.0	568.9	567.7	293.06	293.06	0.00	0.00
8-60-4	722.0	456.7	457.9	0.3	0.6	0.6	722.3	457.3	458.5	436.59	436.59	0.00	0.00
8-60-5	629.4	388.0	387.2	0.1	0.8	0.8	629.6	388.8	388.0	490.05	490.05	0.00	0.00
8-80-1	1802.0	294.0	295.2	0.0	0.2	0.2	1802.0	294.2	295.4	0.00	0.00	0.00	0.00
8-80-2	1803.5	755.1	753.0	0.0	2.0	1.9	1803.6	757.1	755.0	4.06	4.06	0.00	0.00
8-80-3	1805.9	1806.6	1804.8	0.0	0.0	0.0	1805.9	1806.7	1804.9	10.27	10.27	34031.83	34031.83
8-80-4	1810.8	1592.9	1552.3	0.0	2.0	1.8	1810.8	1594.9	1554.2	505.89	505.89	0.00	0.00
8-80-5	1816.2	1323.1	1309.4	0.0	1.4	1.3	1816.2	1324.5	1310.7	776.16	776.16	0.00	0.00
10-70-1	1211.0	275.1	268.0	252.3	0.2	0.2	-1.0	275.3	268.3	0.00	0.00	0.00	0.00
10-70-2	1361.0	901.1	898.2	321.4	3.7	3.9	-1.0	904.9	902.1	49.64	49.64	0.00	0.00
10-70-3	1421.5	867.9	825.4	97.3	1.0	0.9	1518.8	869.0	826.3	210.96	210.96	0.00	0.00
10-70-4	1185.5	715.2	674.6	0.2	1.7	1.6	1185.7	716.8	676.2	441.54	441.54	0.00	0.00
10-70-5	1171.1	692.9	653.7	0.2	1.4	1.3	1171.3	694.3	655.0	572.22	572.22	0.00	0.00
10-100-1	1807.6	888.9	838.0	0.0	0.3	0.2	1807.6	889.2	838.2	0.02	0.02	0.00	0.00
10-100-2	1802.5	1337.5	1307.6	0.0	4.3	3.5	1802.5	1341.8	1311.2	3.17	3.17	0.00	0.00
10-100-3	1805.7	1805.1	1814.2	0.0	0.0	0.0	1805.7	1805.2	1814.2	13.01	8.08	55054.00	54053.50
10-100-4	1810.2	1824.2	1808.3	0.0	0.0	0.0	1810.2	1824.2	1808.3	108.16	126.58	76075.00	72071.00
10-100-5	1803.1	1828.1	1816.7	0.0	0.0	0.0	1803.1	1828.2	1816.7	271.16	324.72	75075.00	76076.00

End of Table C.3

Table C.4: Results of the exact method to solve SP with Initial Solutions

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	$gap(\%)$	LR	SearchCol	CG	Search	Total
2-20	1	0.28	0.28	0.53	1	41	40.68	0.03	40.725
2-20	2	0.82	0.82	0.51	1	41	15.21	0.03	15.257
2-20	3	79.64	79.64	0.99	1	41	18.72	0.00	18.735
2-20	4	142.52	142.52	0.86	1	41	13.11	0.01	13.114
2-20	5	219.39	219.39	0.69	1	41	13.43	0.01	13.445

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached.

Continues to next page

Table C.4 – from previous page

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	$gap(\%)$	LR	SearchCol	CG	Search	Total
2-30	1	1.04	1.04	0.43	1	41	364.34	0.05	364.42
2-30	2	12.72	12.72	0.76	1	41	100.89	0.08	100.981
2-30	3	276.61	276.61	0.88	1	41	87.33	0.11	87.471
2-30	4	491.62	491.62	0.69	1	41	57.88	0.03	57.93
2-30	5	652.06	652.06	0.58	1	41	47.57	0.03	47.628
2-40	1	1.25	1.25	0.61	1	41	1270.04	0.15	1270.272
2-40	2	13.21	13.21	0.85	1	41	420.07	0.10	420.212
2-40	3	324.31	324.31	0.99	1	41	349.62	0.11	349.782
2-40	4	706.34	706.34	0.82	1	41	271.54	0.03	271.621
2-40	5	1088.14	1088.14	0.70	1	41	222.55	0.03	222.631
2-50	1	1.16	1.16	0.75	3	41	1803.58	0.00	1803.764
2-50	2	2.89	2.89	0.71	1	41	1002.37	0.11	1002.565
2-50	3	449.1	449.1	0.99	1	41	881.30	0.07	881.47
2-50	4	1059.04	1059.04	0.93	1	41	767.50	0.03	767.648
2-50	5	1511.43	1511.43	0.77	1	41	626.06	0.02	626.222
4-30	1	0.29	0.29	0.82	1	41	38.02	0.11	38.163
4-30	2	28.06	28.06	0.94	1	41	46.01	0.03	46.054
4-30	3	145.66	145.66	0.75	1	41	39.28	0.05	39.357
4-30	4	208.03	208.03	0.58	1	41	30.09	0.03	30.171
4-30	5	269.31	269.31	0.52	1	41	28.06	0.08	28.18
4-40	1	0.56	0.56	0.87	1	41	141.15	0.14	141.368
4-40	2	37.99	37.99	0.99	1	41	165.97	0.10	166.131
4-40	3	141.25	141.25	0.81	1	41	149.02	0.06	149.171
4-40	4	267.77	267.77	0.74	1	41	128.86	0.05	129.011
4-40	5	372.49	372.49	0.63	1	41	117.99	0.05	118.139
4-50	1	0.47	0.47	0.90	1	41	346.77	0.29	347.194
4-50	2	3.05	3.05	0.89	1	41	453.68	0.18	454.002

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached.

Continues to next page

Table C.4 – from previous page

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	$gap(\%)$	LR	SearchCol	CG	Search	Total
4-50	3	69.03	69.03	0.98	1	41	440.16	0.08	440.43
4-50	4	249.78	249.78	0.97	1	41	327.47	0.08	327.76
4-50	5	450.45	450.45	0.84	1	41	311.42	0.05	311.696
4-60	1	4.49	4.49	0.98	1	41	1340.81	2.27	1343.318
4-60	2	35.11	35.11	0.64	1	41	1156.78	0.58	1157.746
4-60	3	250.49	250.49	0.88	1	41	1041.32	0.11	1041.793
4-60	4	639.34	639.34	0.88	1	41	969.62	0.05	970.077
4-60	5	838.14	838.14	0.76	1	41	724.16	0.05	724.645
6-40	1	0.22	0.22	1.00	1	41	43.29	0.19	43.555
6-40	2	2.83	2.83	0.99	1	41	93.81	0.35	94.262
6-40	3	77.77	77.77	0.99	1	41	130.87	0.06	131.069
6-40	4	153.92	153.92	0.80	1	41	80.21	0.05	80.417
6-40	5	214.2	214.2	0.66	1	41	85.29	0.07	85.504
6-50	1	0.41	0.41	0.83	1	41	154.62	0.17	154.957
6-50	2	18.45	18.45	0.82	1	41	306.54	0.18	306.941
6-50	3	170.91	170.91	0.92	1	41	355.06	0.16	355.491
6-50	4	293.36	293.36	0.73	1	41	286.95	0.06	287.311
6-50	5	406.14	406.14	0.61	1	41	234.56	0.07	234.961
6-70	1	0.47	0.47	0.98	1	41	700.81	0.86	702.181
6-70	2	21.52	21.52	0.88	1	41	1340.10	0.35	1341.226
6-70	3	146.52	146.52	0.93	1	41	1317.26	0.22	1318.439
6-70	4	403.22	403.22	0.87	1	41	1244.63	0.16	1245.823
6-70	5	582.24	582.24	0.75	1	41	991.85	0.06	993.035
8-40	1	0.14	0.14	1.00	1	41	41.10	0.34	41.55
8-40	2	16.13	16.13	0.80	1	41	94.47	0.18	94.788
8-40	3	83.22	83.22	0.86	1	41	93.20	0.06	93.44
8-40	4	120.9	120.9	0.61	1	41	68.94	0.05	69.178

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached.

Continues to next page

Table C.4 – from previous page

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	*gap*(%)	LR	SearchCol	CG	Search	Total
8-40	5	162.44	162.44	0.56	1	41	89.20	0.06	89.436
8-60	1	5.2	5.2	1.00	1	41	402.32	23.94	426.737
8-60	2	35.02	35.02	0.70	1	41	537.28	0.09	538.014
8-60	3	168.3	168.3	0.82	1	41	573.48	0.14	574.342
8-60	4	305.11	305.11	0.71	1	41	491.82	0.11	492.699
8-60	5	382.29	382.29	0.57	1	41	387.56	0.06	388.416
8-80	1	0.21	0.21	1.00	1	41	722.69	0.78	724.502
8-80	2	7.55	7.55	0.61	1	41	1611.01	187.70	1800.273
8-80	3	127.96	127.96	0.97	3	41	1800.01	0.00	1802.248
8-80	4	389.52	389.52	0.93	1	41	1577.33	0.13	1579.578
8-80	5	555.05	555.05	0.74	1	41	1451.53	0.12	1453.79
10-70	1	0.28	0.28	1.00	1	41	437.79	0.88	439.661
10-70	2	27.94	27.94	0.89	1	41	930.38	0.17	931.903
10-70	3	174.24	174.24	0.81	1	41	910.90	0.10	912.566
10-70	4	350.79	350.79	0.77	1	41	802.25	0.11	803.989
10-70	5	433.83	433.83	0.64	1	41	746.21	0.13	748.031
10-100	1	0.39	0.39	1.00	1	41	1479.94	1.57	1484.944
10-100	2	7.62	7.62	1.18	3	41	1815.32	0.00	1820.336
10-100	3	119.85	119.85	1.06	3	41	1801.12	0.00	1806.97
10-100	4	558.28	558.28	1.45	3	41	1796.11	0.00	1802.263
10-100	5	756.97	756.97	1.18	3	41	1828.74	0.00	1835.372

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached.

End of Table C.4

Table C.5: Results of the Independent Heuristic to solve SP with Initial Solutions

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	$gap(\%)$	LR	SearchCol	CG	Search	Total
2-20	1	0.28	0.28	0.53	1	41	11.26	0.20	11.475
2-20	2	0.82	0.82	0.51	1	41	12.84	0.04	12.887
2-20	3	79.64	79.64	0.99	1	41	16.26	0.02	16.292
2-20	4	142.52	142.52	0.86	1	41	15.12	0.02	15.15
2-20	5	219.39	219.39	0.69	1	41	13.65	0.02	13.673
2-30	1	1.04	0.99	0.40	1	41	163.34	1.06	164.444
2-30	2	12.72	12.72	0.76	1	41	60.28	0.69	60.988
2-30	3	276.61	276.61	0.88	1	41	76.45	0.13	76.603
2-30	4	491.62	491.62	0.69	1	41	61.70	0.07	61.796
2-30	5	652.06	652.06	0.58	1	41	51.66	0.13	51.811
2-40	1	1.25	1.25	0.61	1	41	453.36	1.98	455.44
2-40	2	13.21	13.21	0.85	1	41	281.36	4.01	285.401
2-40	3	324.31	324.31	0.99	1	41	318.08	3.12	321.248
2-40	4	706.34	706.34	0.82	1	41	254.06	0.15	254.265
2-40	5	1088.14	1088.14	0.70	1	41	232.28	0.27	232.606
2-50	1	1.16	1.16	0.62	1	41	1160.33	5.24	1165.753
2-50	2	2.89	2.89	0.71	1	41	655.64	10.38	666.096
2-50	3	449.1	449.10	0.99	1	41	806.52	15.32	821.941
2-50	4	1059.04	1059.04	0.93	1	41	770.02	0.67	770.807
2-50	5	1511.43	1511.43	0.77	1	41	710.64	0.55	711.33
4-30	1	0.29	0.22	0.76	1	41	4.87	0.16	5.057
4-30	2	28.06	28.06	0.94	1	41	47.90	0.18	48.115
4-30	3	145.66	145.66	0.75	1	41	47.17	0.10	47.309
4-30	4	208.03	208.03	0.58	1	41	39.20	0.06	39.3
4-30	5	269.31	269.31	0.52	1	41	37.59	0.08	37.71
4-40	1	0.56	0.50	0.86	1	41	13.63	0.24	13.936
4-40	2	37.99	37.99	0.99	1	41	125.72	1.36	127.149

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached.

Continues to next page

Table C.5 – from previous page

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	*gap*(%)	LR	SearchCol	CG	Search	Total
4-40	3	141.25	141.25	0.81	1	41	153.03	0.17	153.282
4-40	4	267.77	267.77	0.74	1	41	126.48	0.13	126.718
4-40	5	372.49	372.49	0.63	1	41	120.34	0.23	120.68
4-50	1	0.47	0.47	0.90	1	41	90.45	0.83	91.415
4-50	2	3.05	3.05	0.89	1	41	204.93	3.54	208.598
4-50	3	69.03	69.03	0.98	1	41	300.71	3.25	304.138
4-50	4	249.78	249.78	0.97	1	41	309.68	0.76	310.649
4-50	5	450.45	450.45	0.84	1	41	293.36	0.29	293.875
4-60	1	4.49	2.82	0.96	1	41	158.31	3.02	161.576
4-60	2	35.11	35.11	0.64	1	41	855.78	12.31	868.369
4-60	3	250.49	250.49	0.88	1	41	890.48	13.35	904.194
4-60	4	639.34	639.34	0.88	1	41	725.86	1.76	728.031
4-60	5	838.14	838.14	0.76	1	41	612.55	0.83	613.818
6-40	1	0.22	0.22	1.00	1	41	5.49	0.20	5.771
6-40	2	2.83	2.83	0.99	1	41	25.85	1.14	27.084
6-40	3	77.77	77.77	0.99	1	41	88.87	0.80	89.797
6-40	4	153.92	153.92	0.80	1	41	92.22	0.24	92.591
6-40	5	214.2	214.20	0.66	1	41	81.39	0.20	81.722
6-50	1	0.41	0.41	0.83	1	41	14.78	0.58	15.522
6-50	2	18.45	18.45	0.82	1	41	139.52	3.00	142.744
6-50	3	170.91	170.91	0.92	1	41	281.36	1.03	282.668
6-50	4	293.36	293.36	0.73	1	41	226.10	0.22	226.628
6-50	5	406.14	406.14	0.61	1	41	193.84	0.24	194.405
6-70	1	0.47	0.47	0.98	1	41	45.12	1.75	47.367
6-70	2	21.52	21.52	0.88	1	41	340.07	20.28	361.113
6-70	3	146.52	146.52	0.93	1	41	1047.52	3.64	1052.112
6-70	4	403.22	403.22	0.87	1	41	933.15	0.85	935.066

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached.

Continues to next page

Table C.5 – from previous page

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	$gap(\%)$	LR	SearchCol	CG	Search	Total
6-70	5	582.24	582.24	0.75	1	41	719.77	0.67	721.538
8-40	1	0.14	0.14	1.00	1	41	3.63	0.21	3.938
8-40	2	16.13	7.42	0.57	1	41	37.94	0.46	38.541
8-40	3	83.22	77.34	0.85	1	41	80.01	0.53	80.705
8-40	4	120.9	120.90	0.61	1	41	83.19	0.35	83.708
8-40	5	162.44	162.44	0.56	1	41	75.51	0.38	76.078
8-60	1	5.2	5.20	1.00	1	41	11.47	2.66	14.59
8-60	2	35.02	35.02	0.70	1	41	414.82	7.44	422.893
8-60	3	168.3	168.30	0.82	1	41	433.76	0.41	434.881
8-60	4	305.11	305.11	0.71	1	41	386.10	0.37	387.239
8-60	5	382.29	382.29	0.57	1	41	326.20	0.22	327.195
8-80	1	0.21	0.21	1.00	1	41	19.61	0.39	21.037
8-80	2	7.55	7.55	0.61	1	41	109.39	14.70	125.662
8-80	3	127.96	127.96	0.97	1	41	1356.61	12.81	1371.343
8-80	4	389.52	389.52	0.93	1	41	1223.55	1.45	1227.143
8-80	5	555.05	555.05	0.74	1	41	1194.00	1.25	1197.441
10-70	1	0.28	0.28	1.00	1	41	15.56	1.47	18.02
10-70	2	27.94	27.94	0.89	1	41	442.47	26.71	470.527
10-70	3	174.24	174.24	0.81	1	41	738.44	2.18	742.198
10-70	4	350.79	350.79	0.77	1	41	622.40	1.19	625.229
10-70	5	433.83	433.83	0.64	1	41	571.07	1.85	574.611
10-100	1	0.39	0.39	1.00	1	41	39.24	0.65	43.349
10-100	2	7.62	7.62	0.99	1	41	46.00	129.56	180.58
10-100	3	119.85	119.85	0.93	3	41	1802.11	0.00	1808.044
10-100	4	558.28	558.28	0.90	3	41	1822.47	0.00	1828.846
10-100	5	756.97	756.97	0.74	3	41	1830.78	0.00	1837.392

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached.

End of Table C.5

Table C.6: Results of the Global Heuristic to solve SP with Initial Solutions

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	$gap(\%)$	LR	SearchCol	CG	Search	Total
2-20	1	0.28	0.19	0.31	1	41	33.753	0.016	33.796
2-20	2	0.82	0.82	0.51	1	41	15.871	0.019	15.893
2-20	3	79.64	79.64	0.99	1	41	21.069	0.01	21.089
2-20	4	142.52	142.52	0.86	1	41	16.982	0.006	16.998
2-20	5	219.39	219.39	0.69	1	41	14.75	0.006	14.766
2-30	1	1.04	0.8	0.26	1	41	320.251	0.057	320.351
2-30	2	12.72	5.35	0.44	1	41	82.608	0.078	82.708
2-30	3	276.61	276.61	0.88	1	41	93.453	0.104	93.581
2-30	4	491.62	491.62	0.69	1	41	59.882	0.016	59.935
2-30	5	652.06	652.06	0.58	1	41	47.79	0.037	47.862
2-40	1	1.25	0.77	0.37	1	41	957.14	0.343	957.568
2-40	2	13.21	6.36	0.68	1	41	343.206	0.297	343.54
2-40	3	324.31	324.31	0.99	1	41	332.761	0.057	332.866
2-40	4	706.34	706.34	0.82	1	41	272.185	0.047	272.291
2-40	5	1088.14	1088.14	0.70	1	41	226.548	0.039	226.649
2-50	1	1.16	1.16	0.69	3	41	1804.067	0	1804.252
2-50	2	2.89	2.89	0.71	1	41	911.303	0.158	911.534
2-50	3	449.1	449.1	0.99	1	41	905.898	0.078	906.074
2-50	4	1059.04	1059.04	0.93	1	41	750.74	0.035	750.888
2-50	5	1511.43	1511.43	0.77	1	41	663.392	0.032	663.556
4-30	1	0.29	0.11	0.52	1	41	22.502	0.105	22.645
4-30	2	28.06	28.06	0.94	1	41	42.001	0.052	42.085
4-30	3	145.66	145.66	0.75	1	41	44.336	0.031	44.401
4-30	4	208.03	208.03	0.58	1	41	33.609	0.047	33.691
4-30	5	269.31	269.31	0.52	1	41	33.108	0.037	33.186
4-40	1	0.56	0.07	0.00	1	51	59.476	-1	59.56

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached

51 - Optimal solution found by CG.

Continues to next page

Table C.6 – from previous page

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	*gap*(%)	LR	SearchCol	CG	Search	Total
4-40	2	37.99	37.99	0.99	1	41	146.619	0.219	146.9
4-40	3	141.25	141.25	0.81	1	41	171.719	0.061	171.862
4-40	4	267.77	267.77	0.74	1	41	134.754	0.047	134.902
4-40	5	372.49	372.49	0.63	1	41	116.531	0.042	116.677
4-50	1	0.47	0.15	0.70	1	41	186.106	0.297	186.533
4-50	2	3.05	3.05	0.89	1	41	356.436	0.739	357.319
4-50	3	69.03	69.03	0.98	1	41	432.136	0.078	432.399
4-50	4	249.78	249.78	0.97	1	41	322.659	0.144	323.015
4-50	5	450.45	450.45	0.84	1	41	325.065	0.048	325.341
4-60	1	4.49	0.31	0.68	1	41	421.938	0.66	422.847
4-60	2	35.11	35.11	0.64	1	41	952.208	1.029	953.519
4-60	3	250.49	250.49	0.88	1	41	1058.633	0.152	1059.148
4-60	4	639.34	639.34	0.88	1	41	862.726	0.062	863.22
4-60	5	838.14	838.14	0.76	1	41	732.034	0.045	732.521
6-40	1	0.22	0	0.00	1	51	7.981	-1	8.057
6-40	2	2.83	2.83	0.99	1	41	41.671	0.379	42.155
6-40	3	77.77	77.77	0.99	1	41	120.284	0.25	120.66
6-40	4	153.92	153.92	0.80	1	41	89.58	0.123	89.839
6-40	5	214.2	214.2	0.66	1	41	91.224	0.109	91.469
6-50	1	0.41	0.18	0.61	1	41	41.443	0.177	41.785
6-50	2	18.45	18.45	0.82	1	41	190.313	0.764	191.322
6-50	3	170.91	170.91	0.92	1	41	342.539	0.111	342.92
6-50	4	293.36	293.36	0.73	1	41	260.44	0.063	260.806
6-50	5	406.14	406.14	0.61	1	41	250.612	0.065	250.994
6-70	1	0.47	0.11	0.91	1	41	100.022	0.432	100.965
6-70	2	21.52	21.52	0.88	1	41	1052.654	2.395	1055.83

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached

51 - Optimal solution found by CG.

Continues to next page

Table C.6 – from previous page

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	*gap*(%)	LR	SearchCol	CG	Search	Total
6-70	3	146.52	146.52	0.93	1	41	1513.903	0.17	1515.266
6-70	4	403.22	403.22	0.87	1	41	1121.839	0.118	1123.313
6-70	5	582.24	582.24	0.75	1	41	956.397	0.095	957.596
8-40	1	0.14	0	0.00	1	51	8.948	-1	9.267
8-40	2	16.13	16.13	0.80	1	41	64.22	0.238	64.599
8-40	3	83.22	83.22	0.86	1	41	97.379	0.249	97.787
8-40	4	120.9	120.9	0.61	1	41	71.999	0.08	72.256
8-40	5	162.44	162.44	0.56	1	41	68.101	0.172	68.45
8-60	1	5.2	0.11	1.00	1	41	68.861	0.307	69.658
8-60	2	35.02	35.02	0.70	1	41	356.696	0.752	358.079
8-60	3	168.3	168.3	0.82	1	41	576.101	0.102	576.921
8-60	4	305.11	305.11	0.71	1	41	451.061	0.219	452.049
8-60	5	382.29	382.29	0.57	1	41	421.704	0.143	422.669
8-80	1	0.21	0	0.00	1	51	66.032	-1	67.061
8-80	2	7.55	7.55	0.61	1	41	670.158	2.991	674.716
8-80	3	127.96	127.96	0.97	3	41	1804.188	0	1806.405
8-80	4	389.52	389.52	0.93	1	41	1609.171	0.156	1611.504
8-80	5	555.05	555.05	0.74	1	41	1624.479	0.222	1627.63
10-70	1	0.28	0.04	1.00	1	41	79.077	0.198	81.289
10-70	2	27.94	27.94	0.89	1	41	722.707	0.592	724.845
10-70	3	174.24	174.24	0.81	1	41	807.636	0.333	809.566
10-70	4	350.79	350.79	0.77	1	41	740.424	0.717	742.801
10-70	5	433.83	433.83	0.64	1	41	715.813	0.312	717.77
10-100	1	0.39	0.16	1.00	1	41	161.042	0.341	165.319
10-100	2	7.62	7.62	0.99	1	41	393.964	10.389	409.413
10-100	3	119.85	119.85	1.02	3	41	1797.56	0	1803.608

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached

51 - Optimal solution found by CG.

Continues to next page

Table C.6 – from previous page

		Value			Status		Time (sec.)		
M-J	q	$Z_{INITIAL}$	Z_{INC}	$gap(\%)$	LR	SearchCol	CG	Search	Total
10-100	4	558.28	558.28	1.01	3	41	1817.869	0	1827.12
10-100	5	756.97	756.97	1.01	3	41	1798.896	0	1807.153

Status LR: 1 - Optimal LR value (all artificials=0) 3 - Unbounded.

Status SearchCol: 41 - Maximum number of iterations reached

51 - Optimal solution found by CG.

End of Table C.6

Table C.7: Results using VNS1 metaheuristic search

	HigherWeightsCG			Incumbent		
M-J-q	Z_{INC}	Search Time (sec.)	Total Time (sec.)	Z_{INC}	Search Time (sec.)	Total Time (sec.)
2-20-1	0.23	0	32.183	0.23	0.003	50.939
2-20-2	0.82	0	15.241	0.82	0.002	38.97
2-20-3	79.64	0	19.812	79.64	0.02	55.461
2-20-4	142.52	0	15.896	142.52	0.002	33.892
2-20-5	219.39	0.002	14.014	219.39	0.001	28.877
2-30-1	1.04	0.015	312.006	1.04	0.009	383.139
2-30-2	5.35	0.008	80.664	12.72	0.006	114.397
2-30-3	276.61	0	89.669	276.61	0.004	126.541
2-30-4	491.62	0.004	58.702	491.62	0.003	81.975
2-30-5	652.06	0	46.209	652.06	0.002	64.829
2-40-1	1.25	0.016	936.443	1.25	0.035	1039.832
2-40-2	6.36	0.016	339.63	6.36	0.019	416.644
2-40-3	324.31	0.016	330.109	324.31	0.009	387.338
2-40-4	706.34	0.008	273.89	706.34	0.007	315.031
2-40-5	1088.14	0.006	228.476	1088.14	0.006	262.176
2-50-1	1.16	0	1802.816	1.16	0	1804.181
2-50-2	2.89	0.02	932.004	2.89	0.018	1024.731
2-50-3	449.1	0.017	916.088	449.1	0.014	1001.054
2-50-4	1059.04	0.011	762.119	1059.04	0.01	827.645
2-50-5	1511.43	0.01	669.143	1511.43	0.009	716.648
4-30-1	0.25	0.011	22.46	0.29	0.004	32.828
4-30-2	28.06	0.007	41.791	28.06	0.005	62.294
4-30-3	145.66	0.005	44.473	145.66	0.005	65.068
4-30-4	208.03	0.004	33.722	208.03	0.004	49.421
4-30-5	269.31	0.005	33.957	269.31	0.004	49.481
4-40-1	* 0.07	0	59.554	* 0.07	0	72.3
4-40-2	37.99	0.019	147.252	37.99	0.013	180.191
4-40-3	141.25	0.01	173.21	141.25	0.009	208.7

** Optimal solution found by CG.*

Continues to next page

Table C.7 – from previous page

	HigherWeightsCG			Incumbent		
M-J-q	Z_{INC}	Search Time (sec.)	Total Time (sec.)	Z_{INC}	Search Time (sec.)	Total Time (sec.)
4-40-4	267.77	0.009	136.355	267.77	0.008	161.769
4-40-5	372.49	0.007	117.265	372.49	0.007	139.788
4-50-1	0.47	0.019	187.408	0.47	0.016	212.208
4-50-2	3.05	0.019	359.236	3.05	0.014	396.605
4-50-3	69.03	0.015	436.346	69.03	0.014	480.728
4-50-4	249.78	0.011	325.747	249.78	0.01	359.451
4-50-5	450.45	0.011	327.799	450.45	0.009	361.269
4-60-1	4.49	0.035	426.389	4.49	0.029	455.516
4-60-2	35.11	0.044	966.264	35.11	0.039	1023.652
4-60-3	250.49	0.027	1073.235	250.49	0.242	1142.633
4-60-4	639.34	0.024	872.007	639.34	0.019	937.653
4-60-5	838.14	0.016	741.938	838.14	0.015	801.415
6-40-1	* 0	0	7.983	* 0	0	11.788
6-40-2	2.83	0.017	41.64	2.83	0.009	57.72
6-40-3	77.77	0.014	120.936	77.77	0.009	153.481
6-40-4	153.92	0.008	89.761	153.92	0.008	111.143
6-40-5	214.2	0.008	92.559	214.2	0.007	113.98
6-50-1	0.18	0.011	41.881	0.41	0.008	49.05
6-50-2	18.45	0.027	191.099	18.45	0.019	217.363
6-50-3	170.91	0.033	344.585	170.91	0.015	388.61
6-50-4	293.36	0.015	263.036	293.36	0.01	294.134
6-50-5	406.14	0.015	255.964	406.14	0.012	283.582
6-70-1	0.11	0.045	101.929	0.47	0.016	113.3
6-70-2	21.52	0.074	1070.914	21.52	0.036	1118.034
6-70-3	146.52	0.031	1460.501	146.52	0.406	1538.874
6-70-4	403.22	0.024	1124.38	403.22	0.586	1206.135
6-70-5	582.24	0.023	936.86	582.24	0.024	996.526
8-40-1	* 0	0	5.305	* 0	0	8.513
8-40-2	16.13	1736.581	1800.004	16.13	1721.206	1800.004
8-40-3	83.22	1705.463	1800.003	83.22	1683.54	1800.035
8-40-4	120.9	1728.208	1800.003	120.9	1708.89	1800.004
8-40-5	162.44	1732.044	1800.004	162.44	1713.368	1800.003
8-60-1	0.11	1730.617	1800.006	0.11	1722.517	1800.007
8-60-2	35.02	1441.085	1800.003	35.02	1405.723	1800.004
8-60-3	168.3	1235.278	1800.005	168.3	1193.931	1800.003
8-60-4	305.11	1345.919	1800.005	305.11	1310.897	1800.004
8-60-5	382.29	1376.747	1800.004	382.29	1344.984	1800.005
8-80-1	* 0	0	68.035	* 0	0	70.768
8-80-2	7.55	1135.625	1800.004	7.55	1108.068	1800.006
8-80-3	127.96	0	1810.226	127.96	0	1812.187
8-80-4	389.52	173.014	1800.003	389.52	122.704	1800.006
8-80-5	555.05	213.985	1800.004	555.05	198.528	1800.005
10-70-1	0.28	0.046	81.956	0.28	0.02	85.166

** Optimal solution found by CG.*

Continues to next page

Table C.7 – from previous page

	HigherWeightsCG			Incumbent		
M-J-q	Z_{INC}	Search Time (sec.)	Total Time (sec.)	Z_{INC}	Search Time (sec.)	Total Time (sec.)
10-70-2	27.94	0.107	731.165	27.94	0.054	767.305
10-70-3	174.24	0.078	816.151	174.24	0.052	860.373
10-70-4	350.79	0.061	748.891	350.79	0.04	789.176
10-70-5	433.83	0.081	718.651	433.83	0.041	759.167
10-100-1	0.39	0.04	156.408	0.39	0.033	156.903
10-100-2	7.62	0.144	392.344	7.62	0.051	400.706
10-100-3	119.85	0	1836.191	119.85	0	1832.511
10-100-4	558.28	0	1809.232	558.28	0	1800.355
10-100-5	756.97	0	1807.435	756.97	0	1816.690

** Optimal solution found by CG.*

End of Table C.7

Table C.8: Results using VNS12 metaheuristic search

		HigherWeightsCG			Incumbent		
M-J-q		Z_{INC}	Search Time (sec.)	Total Time (sec.)	Z_{INC}	Search Time (sec.)	Total Time (sec.)
2-20-1		0.19	0.15	33.786	0.19	0.152	55.601
2-20-2		0.82	0.101	15.916	0.82	0.093	42.212
2-20-3		79.64	0.144	21.323	79.64	0.132	44.742
2-20-4		142.52	0.086	16.956	142.52	0.089	35.478
2-20-5		219.39	0.062	14.486	219.39	0.103	29.665
2-30-1		0.8	2.215	325.718	0.8	2.298	395.003
2-30-2		5.35	0.932	83.983	5.35	0.913	115.756
2-30-3		276.61	0.483	94.026	276.61	0.444	126.015
2-30-4		491.62	0.205	60.476	491.62	0.172	82.167
2-30-5		652.06	0.14	47.797	652.06	0.119	64.855
2-40-1		0.77	2.537	962.56	0.77	2.526	1044.945
2-40-2		6.36	2.83	349.587	6.36	2.814	408.815
2-40-3		324.31	1.364	339.959	324.31	1.222	386.763
2-40-4		706.34	0.823	279.947	706.34	0.702	315.992
2-40-5		1088.14	0.501	233.249	1088.14	0.453	263.026
2-50-1		1.16	0	1809.198	1.16	0	1800.128
2-50-2		2.89	4.46	936.664	2.89	4.023	1003.064
2-50-3		449.1	2.905	922.753	449.1	2.702	1006.311
2-50-4		1059.04	1.604	764.607	1059.04	1.497	822.128
2-50-5		1511.43	1.188	671.475	1511.43	1.087	712.448
4-30-1		0.11	0.237	22.893	0.14	0.405	33.941
4-30-2		28.06	0.496	42.467	28.06	0.474	62.815
4-30-3		145.66	0.401	45.223	145.66	0.345	65.456
4-30-4		208.03	0.228	34.296	208.03	0.19	49.749
4-30-5		269.31	0.227	34.434	269.31	0.202	49.667
4-40-1	*	0.07	0	59.786	0.07	0	71.907
4-40-2		37.99	1.992	149.57	37.99	1.519	180.606

** Optimal solution found by CG.*

Continues to next page

Table C.8 – from previous page

		HigherWeightsCG			Incumbent	
M-J-q	Z_{INC}	Search Time (sec.)	Total Time (sec.)	Z_{INC}	Search Time (sec.)	Total Time (sec.)
4-40-3	141.25	1.291	175.009	141.25	1.171	208.257
4-40-4	267.77	0.696	137.034	267.77	0.645	162.354
4-40-5	372.49	0.556	118.399	372.49	0.495	138.94
4-50-1	0.15	1.729	189.967	0.47	0.992	213.509
4-50-2	3.05	2.605	361.431	3.05	2.371	408.198
4-50-3	69.03	2.765	439.575	69.03	2.303	482.444
4-50-4	249.78	1.344	328.931	249.78	1.233	359.084
4-50-5	450.45	1.342	330.125	450.45	1.224	359.163
4-60-1	0.31	3.971	432.161	0.33	4.756	461.768
4-60-2	35.11	8.397	973.747	35.11	6.476	1050.655
4-60-3	250.49	5.952	1080.873	250.49	5.32	1170.42
4-60-4	639.34	3.786	878.575	639.34	3.42	922.536
4-60-5	838.14	2.357	745.271	838.14	2.144	780.222
6-40-1 *	0	0	7.929	0	0	11.399
6-40-2	2.83	0.988	42.977	2.83	0.499	54.976
6-40-3	77.77	1.273	122.658	77.77	1.126	153.318
6-40-4	153.92	0.643	90.793	153.92	0.57	110.921
6-40-5	214.2	0.624	93.189	214.2	0.567	112.93
6-50-1	0.18	0.326	42.036	0.18	0.594	48.975
6-50-2	18.45	3.661	194.835	18.45	1.833	215.776
6-50-3	170.91	3.367	347.889	170.91	2.436	390.122
6-50-4	293.36	1.369	263.863	293.36	1.264	292.421
6-50-5	406.14	1.6	255.337	406.14	1.336	284.015
6-70-1	0.47	1.282	103.259	0.47	1.001	112.427
6-70-2	21.52	12.627	1081.849	21.52	6.439	1113.695
6-70-3	146.52	7.982	1470.448	146.52	7.555	1520.516
6-70-4	403.22	4.636	1156.85	403.22	4.317	1181.672
6-70-5	582.24	3.052	949.802	582.24	2.773	994.988
8-40-1 *	0	0	5.421	0	0	8.078
8-40-2	16.13	1735.693	1800.03	16.13	1720.441	79.56
8-40-3	83.22	1705.519	1800.213	83.22	1680.751	119.261
8-40-4	120.9	1728.207	1800.054	120.9	1710.268	89.828
8-40-5	162.44	1731.846	1800.065	162.44	1713.271	86.788
8-60-1	0.11	1730.303	1800.006	0.11	1718.893	1800.032
8-60-2	35.02	1436.814	1800.031	35.02	1400.09	400.108
8-60-3	168.3	1230.275	1800.306	168.3	1188.289	611.985
8-60-4	305.11	1343.572	1800.078	305.11	1309.972	490.026
8-60-5	382.29	1376.59	1800.209	382.29	1337.516	462.631
8-80-1 *	0	0	67.725	0	0	82.907
8-80-2	7.55	1134.61	1800.239	7.55	1114.572	685.541
8-80-3	127.96	0	1812.713	127.96	0	1826.428
8-80-4	389.52	172.74	1800.37	389.52	95.387	1704.574
8-80-5	555.05	235.959	1800.295	555.05	109.371	1690.53

** Optimal solution found by CG.*

Continues to next page

Table C.8 – from previous page

	HigherWeightsCG			Incumbent		
M-J-q	Z_{INC}	Search Time (sec.)	Total Time (sec.)	Z_{INC}	Search Time (sec.)	Total Time (sec.)
10-70-1	0.28	1.469	81.088	0.28	0.357	113.715
10-70-2	27.94	8.895	740.963	27.94	5.024	763.415
10-70-3	174.24	14.73	834.475	174.24	6.536	850.976
10-70-4	350.79	8.02	758.389	350.79	7.16	789.65
10-70-5	433.83	5.903	728.761	433.83	5.919	760.881
10-100-1	0.16	1.377	160.444	0.16	1.186	204.21
10-100-2	7.62	9.725	405.4	7.62	5.17	421.21
10-100-3	119.85	0	1804.571	119.85	0	1813.039
10-100-4	558.28	0	1814.835	558.28	0	1819.538
10-100-5	756.97	0	1811.977	756.97	0	1821.268

** Optimal solution found by CG.*

End of Table C.8

Table C.9: Results using Perturbator Comb Prob

M-J-q	Z_{INC}	Iteration(s)	Time$_{INC}$ (sec.)	Time$_{TOTAL}$ (sec.)	Status
2-20-1	0.19	2	32.901	154.946	42
2-20-1	0.19	2	34.277	40.494	42
2-20-1	0.18	3	1274.719	1426.952	42
2-20-2	0.82	2	16.198	84.745	42
2-20-2	0.82	2	16.965	121.38	42
2-20-2	0.82	2	67.417	593.48	42
2-20-3	79.64	2	21.567	191.279	42
2-20-3	79.64	2	22.411	161.334	42
2-20-3	79.64	2	83.265	743.334	42
2-20-4	142.52	2	18.67	315.508	42
2-20-4	142.52	2	19.311	383.016	42
2-20-4	142.52	2	64.662	826.914	42
2-20-5	219.39	2	17.584	303.092	42
2-20-5	219.39	2	17.928	270.368	42
2-20-5	219.39	2	57.785	1164.61	42
2-30-1	0.8	2	333.515	1803.365	42
2-30-1	0.8	2	333.888	1815.836	42
2-30-1	0.8	2	477.132	1815.113	42
2-30-2	5.35	2	92.257	1811.928	42
2-30-2	5.35	2	91.851	1806.316	42
2-30-2	5.35	2	182.433	1805.571	42
2-30-3	276.61	2	107.189	1802.256	42
2-30-3	276.61	2	107.543	1802.692	42
2-30-3	276.61	2	201.218	1802.781	42
2-30-4	491.62	2	72.354	1802.945	42

Status:

42 - Maximum number of iterations without improvement reached.

51 - Optimal solution found by CG.

Continues to next page

Table C.9 – from previous page

M-J-q	Z_{INC}	Iteration(s)	Time_{INC} (sec.)	Time_{TOTAL} (sec.)	Status
2-30-4	491.62	2	72.801	1801.739	42
2-30-4	491.62	2	134.559	1802.367	42
2-30-5	652.06	2	59.674	1802.656	42
2-30-5	652.06	2	59.791	1802.069	42
2-30-5	652.06	2	107.931	1802.276	42
2-40-1	0.77	2	1006.173	1844.496	42
2-40-1	0.77	2	1017.973	1832.97	42
2-40-1	0.77	2	1269.272	1817.784	42
2-40-2	6.36	2	389.995	1804.495	42
2-40-2	6.36	2	397.761	1803.837	42
2-40-2	6.36	2	537.903	1804.951	42
2-40-3	324.31	2	382.934	1803.963	42
2-40-3	324.31	2	382.044	1806.284	42
2-40-3	324.31	2	516.157	1805.803	42
2-40-4	706.34	2	314.575	1804.809	42
2-40-4	706.34	2	328.354	1804.99	42
2-40-4	706.34	2	417.172	1804.988	42
2-40-5	1088.14	2	263.383	1804.941	42
2-40-5	1088.14	2	264.083	1803.724	42
2-40-5	1088.14	2	346.003	1805.454	42
2-50-1	1.16	2	1804.986	1805.061	42
2-50-1	1.16	2	1803.314	1803.448	42
2-50-1	1.16	2	1806.331	1806.562	42
2-50-2	2.89	2	1007.883	1804.522	42
2-50-2	2.89	2	1008.422	1805.244	42
2-50-2	2.89	2	1006.134	1804.132	42
2-50-3	449.1	2	991.431	1805.632	42
2-50-3	449.1	2	989.613	1804.134	42
2-50-3	449.1	2	993.134	1806.134	42
2-50-4	1059.04	2	820.849	1804.801	42
2-50-4	1059.04	2	819.914	1804.132	42
2-50-4	1059.04	2	820.013	1803.53	42
2-50-5	1511.43	2	722.645	1805.443	42
2-50-5	1511.43	2	722.244	1804.511	42
2-50-5	1511.43	2	724.615	1806.051	42
4-30-1	0.11	2	34.838	42.217	42
4-30-1	0.11	2	33.52	46.492	42
4-30-1	0.11	2	60.41	365.669	42
4-30-2	28.03	3	156.649	271.375	42
4-30-2	28.06	2	62.021	109.384	42
4-30-2	26.03	4	550.491	924.576	42
4-30-3	145.66	2	69.519	329.978	42

Status:

42 - Maximum number of iterations without improvement reached.

51 - Optimal solution found by CG.

Continues to next page

Table C.9 – from previous page

M-J-q	Z_{INC}	Iteration(s)	Time_{INC} (sec.)	Time_{TOTAL} (sec.)	Status
4-30-3	145.66	2	65.268	143.795	42
4-30-3	145.66	2	120.981	321.007	42
4-30-4	208.03	2	53.167	215.131	42
4-30-4	208.03	2	49.534	180.769	42
4-30-4	208.03	2	91.82	444.899	42
4-30-5	269.31	2	53.787	221.935	42
4-30-5	269.31	2	49.645	272.121	42
4-30-5	269.31	2	91.842	510.361	42
4-40-1	0.07	1	75.836	75.84	51
4-40-1	0.07	1	75.426	75.43	51
4-40-1	0.07	1	106.95	106.954	51
4-40-2	37.97	3	1185.918	1823.695	42
4-40-2	37.97	3	1381.226	1811.099	42
4-40-2	37.99	2	271.259	1811.187	42
4-40-3	139.25	3	1650.011	1803.572	42
4-40-3	141.25	2	213.179	765.551	42
4-40-3	141.25	2	308.177	1152.819	42
4-40-4	260.66	3	1406.868	1805.605	42
4-40-4	267.77	2	165.939	1319.236	42
4-40-4	267.77	2	239.345	1808.047	42
4-40-5	372.49	2	152.104	1301.319	42
4-40-5	372.49	2	144.283	1215.41	42
4-40-5	372.49	2	207.944	1561.745	42
4-50-1	0.15	2	225.681	1807.916	42
4-50-1	0.15	2	213.042	275.712	42
4-50-1	0.15	2	275.255	400.49	42
4-50-2	3.05	2	426.869	1809.113	42
4-50-2	2.75	4	1800.811	1806.18	42
4-50-2	3.02	3	1804.433	1816.051	42
4-50-3	69.03	2	519.45	1806.039	42
4-50-3	69.03	2	493.958	1810.112	42
4-50-3	69.03	2	610.4	1816.967	42
4-50-4	249.78	2	383.963	1807.799	42
4-50-4	249.78	2	371.695	1805.108	42
4-50-4	249.78	2	459.987	1805.985	42
4-50-5	450.45	2	387.28	1810.289	42
4-50-5	450.45	2	377.167	1808.364	42
4-50-5	450.45	2	482.967	1809.5	42
4-60-1	0.31	2	484.48	1814.034	42
4-60-1	0.31	2	491.04	1804.441	42
4-60-1	0.31	2	471.04	1813.543	42
4-60-2	34.85	3	1802.544	1824.363	42

Status:

42 - Maximum number of iterations without improvement reached.

51 - Optimal solution found by CG.

Continues to next page

Table C.9 – from previous page

M-J-q	Z_{INC}	Iteration(s)	Time_{INC} (sec.)	Time_{TOTAL} (sec.)	Status
4-60-2	35.11	2	1025.32	1819.431	42
4-60-2	35.11	2	1013.87	1809.413	42
4-60-3	250.49	2	1215.761	1828.098	42
4-60-3	250.49	2	1194.031	1823.012	42
4-60-3	250.49	2	1121.011	1812.726	42
4-60-4	639.34	2	967.283	1808.024	42
4-60-4	639.34	2	945.037	1809.562	42
4-60-4	639.34	2	935.944	1804.701	42
4-60-5	838.14	2	820.903	1814.257	42
4-60-5	838.14	2	812.003	1819.807	42
4-60-5	838.14	2	809.993	1809.631	42
6-40-1	0	1	13.878	13.882	51
6-40-1	0	1	11.927	11.93	51
6-40-1	0	1	21.788	21.791	51
6-40-2	0.16	4	109.338	123.764	42
6-40-2	0.16	3	71.911	446.434	42
6-40-2	0.32	3	361.894	426.513	42
6-40-3	76.81	3	308.595	381.306	42
6-40-3	77.77	2	160.922	282.243	42
6-40-3	77.77	2	239.079	388.83	42
6-40-4	151.92	3	293.242	629.423	42
6-40-4	153.92	2	119.341	330.34	42
6-40-4	153.92	2	174.447	619.535	42
6-40-5	214.2	2	134.256	306.608	42
6-40-5	214.2	2	122.722	276.925	42
6-40-5	214.2	2	180.014	477.072	42
6-50-1	0.18	2	55.127	127.622	42
6-50-1	0.14	4	1779.439	1808.461	42
6-50-1	0.14	4	1217.533	1369.601	42
6-50-2	12.44	3	1370.013	1810.719	42
6-50-2	8.75	5	1315.668	1752.166	42
6-50-2	18.45	2	176.212	870.439	42
6-50-3	170.91	2	427.404	979.289	42
6-50-3	170.91	2	407.793	1159.784	42
6-50-3	169.96	3	1216.473	1807.986	42
6-50-4	292.37	3	950.079	1579.845	42
6-50-4	293.36	2	310.517	1094.26	42
6-50-4	293.36	2	247.368	927.784	42
6-50-5	406.14	2	315.549	1650.063	42
6-50-5	406.14	2	300.971	1054.217	42
6-50-5	406.14	2	239.883	859.946	42
6-70-1	0.11	2	135.437	392.564	42

Status:

42 - Maximum number of iterations without improvement reached.

51 - Optimal solution found by CG.

Continues to next page

Table C.9 – from previous page

M-J-q	Z_{INC}	Iteration(s)	Time_{INC} (sec.)	Time_{TOTAL} (sec.)	Status
6-70-1	0.11	2	125.311	384.957	42
6-70-1	0.11	2	117.144	380.389	42
6-70-2	21.52	2	1177.246	1828.129	42
6-70-2	21.52	2	1150.145	1823.547	42
6-70-2	21.52	2	1141.415	1818.311	42
6-70-3	146.52	2	1587.793	1822.837	42
6-70-3	146.52	2	1575.229	1813.052	42
6-70-3	146.52	2	1571.018	1810.589	42
6-70-4	403.22	2	1268.029	1818.445	42
6-70-4	403.22	2	1245.414	1818.894	42
6-70-4	403.22	2	1203.103	1807.971	42
6-70-5	582.24	2	1033.403	1820.148	42
6-70-5	582.24	2	1020.874	1811.376	42
6-70-5	582.24	2	1004.857	1806.246	42
8-40-1	0	1	10.27	10.273	51
8-40-1	0	1	8.548	8.551	51
8-40-1	0	1	4.474	4.477	51
8-40-2	5.1	4	261.611	295.9	42
8-40-2	8.04	5	403.75	449.734	42
8-40-2	4.06	6	664.527	672.297	42
8-40-3	53.52	6	797.785	882.386	42
8-40-3	53.58	7	768.582	888.832	42
8-40-3	53.53	5	323.464	393.008	42
8-40-4	120.9	2	109.643	225.506	42
8-40-4	120.9	2	100.583	218.303	42
8-40-4	116.97	3	177.269	259.224	42
8-40-5	123.75	10	1386.058	1520.535	42
8-40-5	134.64	5	571.772	764.278	42
8-40-5	162.44	2	61.088	131.634	42
8-60-1	0.11	2	90.206	185.858	42
8-60-1	0.11	2	83.429	134.15	42
8-60-1	0.01	3	223.26	230.308	42
8-60-2	35.02	2	427.798	597.19	42
8-60-2	34.92	3	884.539	1834.603	42
8-60-2	35.02	2	347.793	1823.448	42
8-60-3	167.31	3	1803.589	1814.426	42
8-60-3	168.3	2	633.972	1816.618	42
8-60-3	168.3	2	550.118	1543.773	42
8-60-4	305.11	2	534.837	1822.741	42
8-60-4	305.11	2	515.762	1830.438	42
8-60-4	305.11	2	442.751	1762.304	42
8-60-5	379.32	3	1542.8	1816.277	42

Status:

42 - Maximum number of iterations without improvement reached.

51 - Optimal solution found by CG.

Continues to next page

Table C.9 – from previous page

M-J-q	Z_{INC}	Iteration(s)	Time$_{INC}$ (sec.)	Time$_{TOTAL}$ (sec.)	Status
8-60-5	382.29	2	486.738	1667.209	42
8-60-5	382.29	2	414.006	1575.414	42
8-80-1	0	1	75.889	75.892	51
8-80-1	0	1	74.244	74.250	51
8-80-1	0	1	75.246	75.254	51
8-80-2	7.55	2	719.437	1509.499	42
8-80-2	7.55	2	708.132	1501.931	42
8-80-2	7.55	2	713.256	1504.875	42
8-80-3	127.96	2	1810.549	1810.809	42
8-80-3	127.96	2	1810.549	1810.549	42
8-80-3	127.96	2	1810.549	1810.549	42
8-80-4	389.52	2	1749.469	1836.931	42
8-80-4	389.52	2	1720.455	1823.039	42
8-80-4	389.52	2	1705.104	1822.242	42
8-80-5	555.05	2	1702.991	1823.737	42
8-80-5	555.05	2	1699.410	1815.126	42
8-80-5	555.05	2	1687.313	1809.143	42
10-70-1	0.04	2	92.44	106.866	42
10-70-1	0.04	2	90.114	104.461	42
10-70-1	0.04	2	77.614	90.266	42
10-70-2	27.94	2	836.715	1818.46	42
10-70-2	27.94	2	794.359	1494.546	42
10-70-2	27.94	2	716.736	1826.394	42
10-70-3	174.24	2	920.666	1834.958	42
10-70-3	174.24	2	897.385	1822.247	42
10-70-3	174.24	2	803.817	1832.976	42
10-70-4	346.83	3	1730.931	1821.775	42
10-70-4	331.98	3	1801.478	1816.762	42
10-70-4	350.79	2	735.514	1815.702	42
10-70-5	433.83	2	837.458	1826.147	42
10-70-5	433.83	2	790.932	1821.153	42
10-70-5	433.83	2	709.562	1594.35	42
10-100-1	0.16	2	230.158	1847.911	42
10-100-1	0.16	2	220.121	1840.144	42
10-100-1	0.16	2	222.431	1838.133	42
10-100-2	7.62	2	376.956	1862.864	42
10-100-2	7.62	2	401.114	1860.131	42
10-100-2	7.62	2	378.345	1844.674	42
10-100-3	119.85	2	1833.886	1833.898	42
10-100-3	119.85	2	1820.091	1820.221	42
10-100-3	119.85	2	1821.01	1821.049	42
10-100-4	558.28	2	1806.076	1806.09	42

Status:

42 - Maximum number of iterations without improvement reached.

51 - Optimal solution found by CG.

Continues to next page

Table C.9 – from previous page

M-J-q	Z_{INC}	Iteration(s)	Time_{INC} (sec.)	Time_{TOTAL} (sec.)	Status
10-100-4	558.28	2	1805.529	1805.54	42
10-100-4	558.28	2	1810.129	1810.14	42
10-100-5	756.97	2	1834.799	1834.813	42
10-100-5	756.97	2	1834.799	1834.813	42
10-100-5	756.97	2	1834.799	1834.813	42

Status:

42 - Maximum number of iterations without improvement reached.

51 - Optimal solution found by CG.

End of Table C.9

Table C.10: Results using Perturbator Comb Type0

M-J-q	Z_{INC}	Iteration(s)	Time_{INC} (sec.)	Time_{TOTAL} (sec.)	Status
2-20-1	0.18	3	539.595	831.977	42
2-20-2	0.82	2	17.496	163.077	42
2-20-3	79.64	2	23.561	133.3	42
2-20-4	142.52	2	20.135	76.899	42
2-20-5	219.39	2	16.852	95.237	42
2-30-1	0.8	2	330.352	1813.771	42
2-30-2	5.35	2	90.251	1824.911	42
2-30-3	276.61	2	104.865	1336.065	42
2-30-4	491.62	2	69.954	305.648	42
2-30-5	652.06	2	55.753	231.346	42
2-40-1	0.77	2	981.711	1836.246	42
2-40-2	6.36	2	372.182	1803.511	42
2-40-3	324.31	2	365.695	1804.335	42
2-40-4	706.34	2	300.862	1036.238	42
2-40-5	1088.14	2	251.565	617.802	42
2-50-1	1.16	2	1800.214	1800.27	42
2-50-2	2.89	2	975.266	1808.101	42
2-50-3	449.1	2	960.702	1806.297	42
2-50-4	1059.04	2	797.575	1808.49	42
2-50-5	1511.43	2	701.305	1806.716	42
4-30-1	0.06	3	168.479	181.709	42
4-30-2	23.37	3	350.577	581.693	42
4-30-3	145.66	2	61.798	218.427	42
4-30-4	208.03	2	47.015	128.742	42
4-30-5	269.31	2	47.324	127.787	42
4-40-1	0.07	1	70.493	70.497	51
4-40-2	37.99	2	175.824	874.969	42
4-40-3	141.25	2	205.227	830.083	42
4-40-4	267.77	2	160.194	560.174	42

Status:

42 - Maximum number of iterations without improvement reached.

51 - Optimal solution found by CG.

Continues to next page

Table C.10 – from previous page

M-J-q	Z_{INC}	Iteration(s)	Time_{INC} (sec.)	Time_{TOTAL} (sec.)	Status
4-40-5	372.49	2	138.495	514.517	42
4-50-1	0.15	2	207.097	1809.309	42
4-50-2	3.05	2	396.489	1817.567	42
4-50-3	69.03	2	484.318	1808.409	42
4-50-4	249.78	2	364.943	1399.466	42
4-50-5	450.45	2	374.384	1146.679	42
4-60-1	0.31	2	462.954	1809.31	42
4-60-2	35.11	2	1033.522	1837.144	42
4-60-3	250.49	2	1147.223	1807.157	42
4-60-4	639.34	2	936.097	1811.04	42
4-60-5	838.14	2	807.328	1814.779	42
6-40-1	0	1	13.004	13.007	51
6-40-2	0.07	4	529.893	761.536	42
6-40-3	77.77	2	158.931	418.121	42
6-40-4	153.92	2	117.854	334.393	42
6-40-5	214.2	2	121.225	301.581	42
6-50-1	0.1	5	793.816	910.873	42
6-50-2	18.45	2	226.682	1146.366	42
6-50-3	170.91	2	400.76	1338.275	42
6-50-4	293.36	2	308.765	715.558	42
6-50-5	406.14	2	297.096	659.858	42
6-70-1	0.11	2	116.098	1820.248	42
6-70-2	21.52	2	1138.253	1831.154	42
6-70-3	146.52	2	1571.129	1823.46	42
6-70-4	403.22	2	1200.107	1824.112	42
6-70-5	582.24	2	1008.142	1813.942	42
8-40-1	0	1	8.719	8.721	51
8-40-2	6	4	348.702	461.91	42
8-40-3	83.22	2	128.868	367.78	42
8-40-4	120.9	2	100.419	205.125	42
8-40-5	162.44	2	96.97	204.234	42
8-60-1	0	2	428.312	428.374	51
8-60-2	35.02	2	410.255	1822.364	42
8-60-3	168.3	2	647.766	1813.392	42
8-60-4	305.11	2	513.014	1279.462	42
8-60-5	382.29	2	476.405	990.731	42
8-80-1	0	1	73.951	73.952	51
8-80-2	7.55	2	703.88	1872.039	42
8-80-3	127.96	2	1808.333	1808.634	42
8-80-4	389.52	2	1710.197	1824.621	42
8-80-5	555.05	2	1685.337	1836.973	42
10-70-1	0	2	162.905	162.92	51

Status:

42 - Maximum number of iterations without improvement reached.

51 - Optimal solution found by CG.

Continues to next page

Table C.10 – from previous page

M-J-q	Z_{INC}	Iteration(s)	Time_{INC} (sec.)	Time_{TOTAL} (sec.)	Status
10-70-2	27.94	2	795.65	1816.828	42
10-70-3	174.24	2	890.937	1834.9	42
10-70-4	350.79	2	820.885	1829.836	42
10-70-5	433.83	2	790.184	1650.888	42
10-100-1	0	3	530.68	569.693	42
10-100-2	7.62	2	409.84	1844.499	42
10-100-3	119.85	2	1839.429	1841.347	42
10-100-4	558.28	2	1809.489	1809.518	42
10-100-5	756.97	2	1803.233	1803.247	42

Status:

42 - Maximum number of iterations without improvement reached.

51 - Optimal solution found by CG.

End of Table C.10