# Tools for Traffic Engineering on IP Networks

Tiago Sá
CCTC
University of Minho
Campus de Gualtar, Braga, Portugal
Email: tiagosa@di.uminho.pt

Miguel Rocha
CCTC
University of Minho
Campus de Gualtar, Braga, Portugal
Email: mrocha@di.uminho.pt

Pedro Sousa
CCTC
University of Minho
Campus de Gualtar, Braga, Portugal
Email: pns@di.uminho.pt

***Abstract*—In this work, an user friendly software application is proposed, built on top of a network optimization framework, aiming to make traffic engineering an easier task for IP network administrators. This framework was developed in the Center of Computer Science and Technology (CCTC) of the University of Minho and allows the improvement of quality of service levels in TCP/IP based networks, by configuring the routing weights of link-state protocols, such as OSPF. This goal is achieved mainly using Evolutionary Algorithms as the optimization engines, while networks are represented using graph-based mathematical models. These methods allow the optimization of distinct cost functions, using penalties that take into account several measures of network performance such as network congestion and average end-to-end delays. The main goal of this work is to create a structured graphical user interface to support the optimization framework, enabling the user to simulate the effects of different OSPF settings, to obtain highly optimized configurations and to compare different weight setting optimization methods.**

***Index Terms*—Traffic Engineering, Routing protocols, Network management, Evolutionary Algorithms, Open-source Software.**

## I. Introduction

During the last years, various kinds of applications have been integrated over IP converged networks, increasing the requirements on the ability to provision adequate service levels. In order to address this issue, researchers came up with many different Quality of Service (QoS) solutions and traffic control mechanisms, mostly based on traffic prioritization and selective resource reservation [1].

There is no single solution to provide this kind of performance and, in general, achieving reasonable service quality requires several components to work seamlessly.

Besides the QoS mechanisms, there are other factors which play a crucial role on the networking performance, like the way data routing is controlled across a given domain. This work focuses on the Open Shortest Path First (OSPF) intra-domain routing protocol, extremely popular due to its simplicity and ease of implementation [9] [10]. In order to perform its job, the administrator sets specific weights to every link in the network, which are then used to compute the best paths from each source to each destination, using the well-known Dijkstra algorithm, resulting on the nodes' routing tables [3]. This weight setting process has a major impact on the networking performance, although in practice simple methods and heuristics are commonly used, like setting the weights inversely proportional to the link capacity. However, this often leads to sub-optimal network resource utilization.

Another approach was taken by Fortz et al. [2], where OSPF wight setting is implemented using traffic engineering, assuming that the administrator has access to a matrix representing traffic demands between each pair of nodes in the network. These authors face this task as an optimization problem, by defining a cost function that measures the network congestion.

In previous work, the authors proposed a new approach [4] [5], also accommodating delay based constraints, that are crucial to implement QoS aware networking services. In this work, optimization algorithms were used to calculate link-state routing weights that optimize traffic congestion, while simultaneously complying with specific delay requirements, providing a multi-constrained QoS aware optimization framework, proved to clearly outperform the common OSPF weight setting heuristics.

Although the published results are competitive, this traffic engineering mechanism that efficiently performs the weight setting task is still emerging within the research community, and there is the need to put it in practice, in real case scenarios. The main goal of this work is, therefore, to develop an user-friendly software application, to be used by any network administrator that allows to apply the developed optimization methods in real environments, like Internet Service Providers (ISPs) or large-scale networking domains.

In order to fill this gap, an application was developed to simplify the use of the existing framework, by hiding the complexity of the optimization tasks, making the administration job easier and more efficient. This application provides a Graphical User Interface (GUI) to improve the interaction with the user, eliminating any requirement for programming skills. Several operations were defined to better allow manipulating data structures, executing optimization algorithms, retrieving the results of simulation and optimization tasks, etc. Data can also be displayed in different forms, in different views, adapted for improved efficiency. This tool was developed in a structured way, in different conceptual layers, divided into modular components which can be easily extended with new functionalities.

The paper proceeds with a description of the optimization framework used, including the methods used in network representation and in OPSF weight setting. Then, the software application is described, including the requirements and functionalities, as well as implementation details. The paper finishes with conclusions and further work.

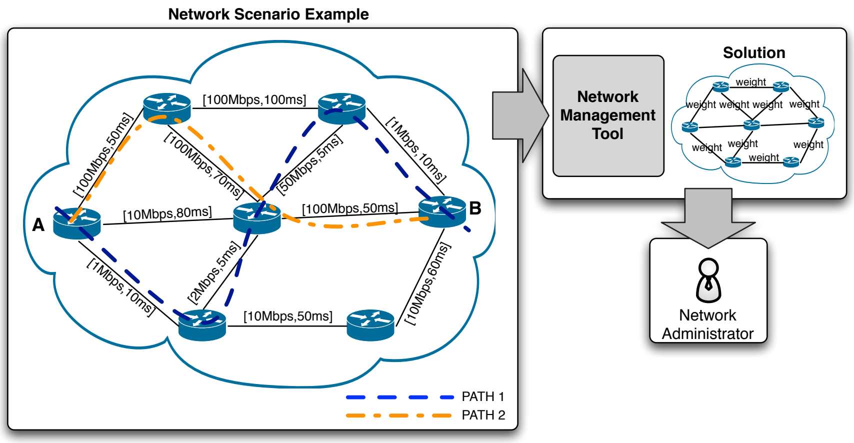


Fig. 1. Example of a network scenario.

## II. Methods for OSPF Weight setting

This section presents an overview of the mathematical problem that gives ground to the optimization task, which can be tackled using different optimization algorithms.

### *A. Problem Definition*

The developed application is supported by the optimization framework presented by the authors in [5], which aims to provide network administrators with efficient OSPF link configurations, taking into account the users demands, the network topology and corresponding characteristics of a given network domain.

This work assumes that client demands are mapped into a matrix[1] summarizing, for each source/destination router pair, a given amount of bandwidth and a target end-to-end delay required to be supported by the network domain.

As an illustrative example, consider the network scenario included in Fig.1, involving an individual demand between two network nodes (A and B). If the demand is mainly expressed in terms of a delay target, then the administrator, in the absence of other traffic, should be able to compute OSPF weights that would result in a data path with the minimum end-to-end delay between the network nodes (PATH 1). In opposition, if no delay requirements are imposed, and the only constraint between A and B is a given bandwidth requirement, then the optimization methods would try to minimize the network congestion and assign OSPF weights to force a data path inducing the lowest level of losses in the traffic (PATH 2). These two distinct optimization aims would result in two distinct sets of OSPF weights.

Additionally, if one considers that a given demand has simultaneously bandwidth and delay constraints, then it is expected that the OSPF weight setting process would try to find a data path representing a trade-off between both. The example in Fig.1 is extremely straightforward, due to the fact that one simple demand was considered in the traffic traversing the network domain. Taking into account the fact that each router pair of a given ISP may have specific multi-constrained QoS requirements (i.e. congestion vs. delay demands), it is easy to understand how complex can the problem get, with the need of obtaining OSPF settings able to optimize multiple parameters of a given network domain.

The general routing problem, that underpins this work, represents routers and transmission links by a set of nodes and a set of arcs, respectively, in a directed graph [8]. Each arc has a specific bandwidth capacity and an average propagation delay, both intrinsic in the network topology [2].

Additionally, a demand matrix is available, where each element represents the traffic demand between each pair of nodes, allowing to calculate the total load on each arc. This value is used to define a congestion measure for each link, resulting on a penalty function that becomes more expensive and exponentially penalizes high values of congestion.

The framework was enriched with the inclusion of delay requirements for each pair of routers in the network. These are modeled as a matrix that, for each pair of nodes, gives the delay target for traffic between the origin and destination. Again, a cost function was developed to evaluate the delay compliance for each scenario. This, in turn, allowed the definition of a delay minimization cost function.

In OSPF, all arcs are associated with an integer weight. Every node uses these weights as an input to the Dijkstra algorithm [3] to calculate the shortest paths to all other nodes in the network. All the traffic from a given source to a destination travels along the shortest path, except when two or more paths have the same length[3]. In that case, traffic is evenly divided among the arcs in these paths (load balancing) [13].

[1]There are several techniques on how to obtain traffic demand matrices [6] [7] which provide estimations regarding the overall QoS requirements within a given network domain.

[2]Note that it was considered that, in the scenarios where this work would be applicable, the delay in each path is dominated by the component given by propagation delays in its arcs and that queuing delays can be neglected.

[3]This feature can be fine-tuned in the routing protocol.

The bi-objective optimization problem addressed in our framework aims to find the set of OSPF weights that simultaneously minimizes the cost functions associated with network congestion and average end-to-end delays in a network domain. Details on the mathematical definitions of this functions can be found in [5].

### B. *Algorithms for OSPF*

Different methods and heuristics can be used in order to solve the optimization problem described in Section II-A. Our framework supports different types of optimization algorithms. Part of those have been implemented in the current version of the software application proposed here.

As mentioned, the base optimization framework resorts to the use of Evolutionary Algorithms (EAs) in order to improve the performance of a given network domain [4]. In the developed EA, each individual encodes a solution as a vector of integer values, where each value (gene) corresponds to the weight of a link in the network. Therefore, the size of the individual equals the number of links in the network. The individuals in the initial population are randomly generated, with link weights taken from an uniform distribution. In order to create new solutions, several reproduction operators were used.

The overall structure of the EA is given by:

1) Generate and evaluate the initial population($P_0$).
2) While the termination criteria is not met:
   a) Select from $P_t$ individuals for reproduction.
   b) Apply the reproduction operators to breed the offspring and evaluate them.
   c) Insert the offspring into the next population ($P_{t+1}$).
   d) Select the survivors from $P_t$ to be kept in $P_{t+1}$.

The selection procedure is done by converting the fitness value into a linear ranking in the population, and then applying a roulette wheel scheme. The default population size of 100 individuals was considered. When using this EA, the user specifies a parameter ($\alpha$) that defines the importance that is confered to each objective (congestion and delays).

Also, two multi-objective EAs (SPEA2 and NSGAII) were implemented. Its natural multi-objective orientation makes them the most adequate algorithms for the job, since no further parameters are defined by the user.

The Differential Evolution (DE) method differs from the EA essentially in the reproduction operators. DE generates trial individuals by calculating vector differences between other randomly selected members of the population. Since OSPF weights are integer, it is necessary to round the values used in the DE before the evaluation. It is important to notice that in the DE all individuals go through the previous reproduction step.

A number of *heuristic methods* were implemented to assess the order of magnitude of the improvements obtained by the proposed methods when compared with the traditional weight setting heuristics:

- InvCap – sets each link weight to a value inversely proportional to its capacity.

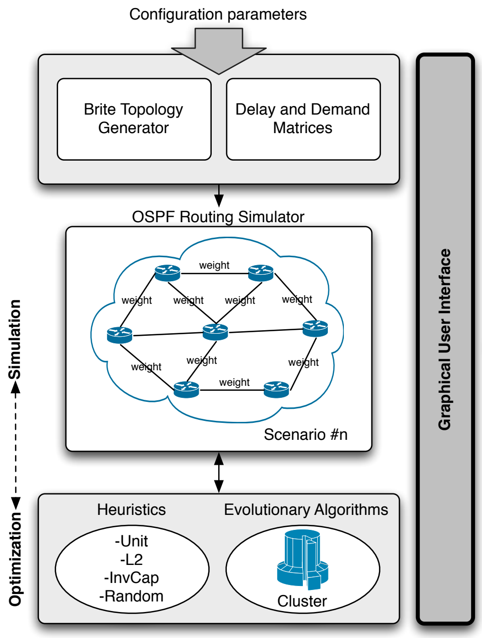


Fig. 2. Platform for performance evaluation.

- L2 – sets each link weight to a value proportional to its Euclidean distance.
- Random – a number of randomly generated solutions are analyzed and the best is selected.
- Unit - sets every link weight to one.

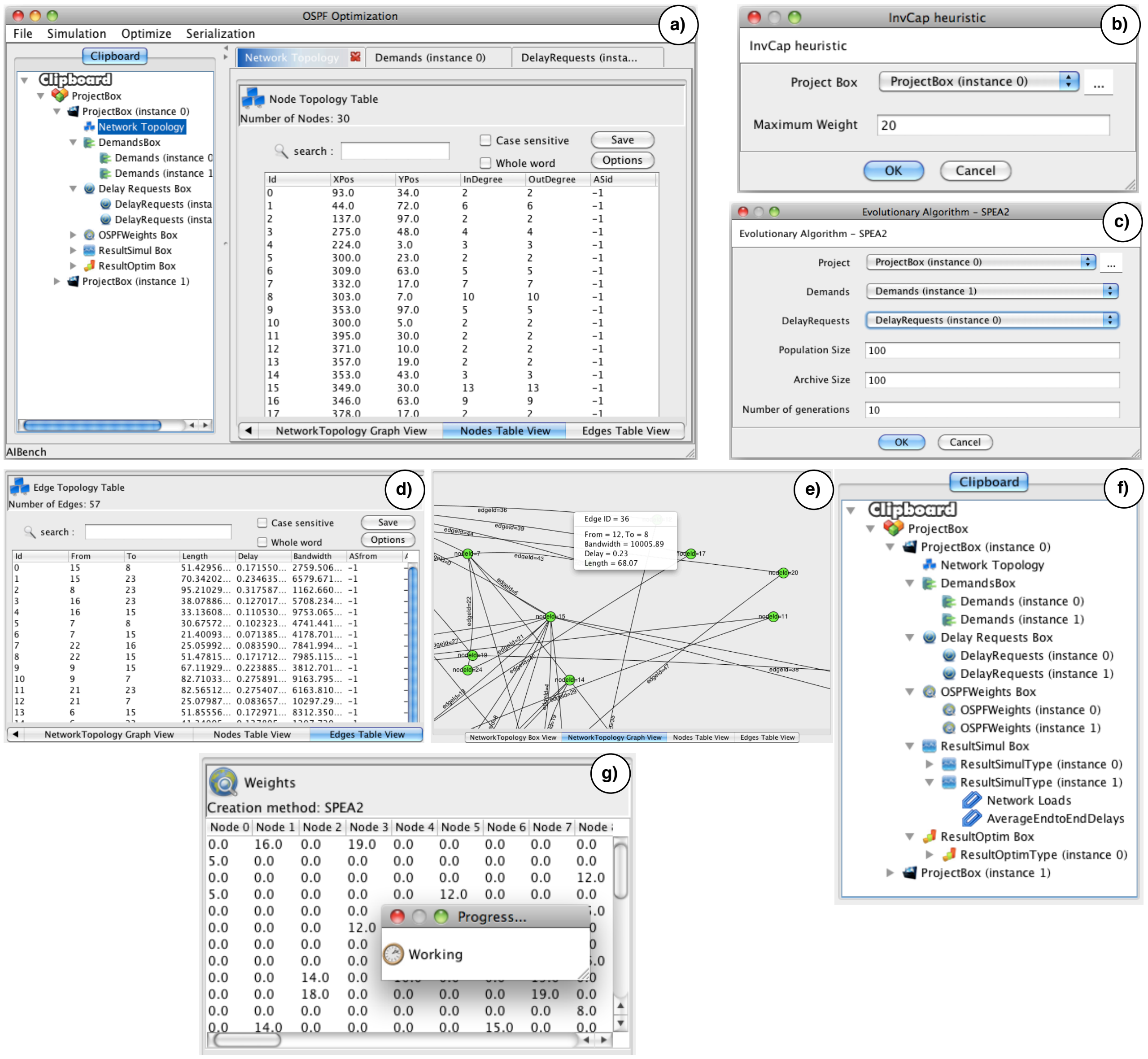
An extended performance analysis of the model is presented in [4] and [5], for a large set of distinct QoS constrained scenarios. Fig.2 presents the experimental platform that was implemented and used in that work for benchmarking. The main components are: a topology generator, a traffic demand generator, an OSPF simulator, a set of optimization heuristics and a module implementing the proposed EA. As emphasized in the figure, the developed application acts as a bridge between the platform and the user.

## III. Software

The developed tools allow the creation of Wide Area Network (WAN) models, setting OSPF weights for each connection and calculating how traffic is routed on the network, for given arrays of point-to-point requirements. This allows to calculate measures of network performance in terms of QoS, such as congestion or average end-to-end delays. An important component is the implementation of optimization algorithms, whose aim is to set the value of the OSPF weights on each connection, in order to improve the network performance, for specific objective functions, involving some QoS metrics.

### A. *Requirements and Functionalities*

The presented problem imposed a set of requirements, which were taken into consideration, as guidelines for the implementation task. One of the main goals of the developed application was to provide an easy way to make use of the



Clipboard
- ProjectBox
  - ProjectBox (instance 0)
    - Network Topology
    - DemandsBox
      - Demands (instance 0)
      - Demands (instance 1)
    - Delay Requests Box
      - DelayRequests (instance 0)
      - DelayRequests (instance 1)
    - OSPFWeights Box
      - OSPFWeights (instance 0)
      - OSPFWeights (instance 1)
    - ResultSimul Box
      - ResultSimulType (instance 0)
      - ResultSimulType (instance 1)
        - Network Loads
        - AverageEndtoEndDelays
    - ResultOptim Box
      - ResultOptimType (instance 0)
  - ProjectBox (instance 1)

Node Topology Table
Number of Nodes: 30

search : Case sensitive Whole word Save Options

| Id | XPos | YPos | InDegree | OutDegree | ASid |
|---|---|---|---|---|---|
| 0 | 93.0 | 34.0 | 2 | 2 | -1 |
| 1 | 44.0 | 72.0 | 6 | 6 | -1 |
| 2 | 137.0 | 97.0 | 2 | 2 | -1 |
| 3 | 275.0 | 48.0 | 4 | 4 | -1 |
| 4 | 224.0 | 3.0 | 3 | 3 | -1 |
| 5 | 300.0 | 23.0 | 2 | 2 | -1 |
| 6 | 309.0 | 63.0 | 5 | 5 | -1 |
| 7 | 332.0 | 17.0 | 7 | 7 | -1 |
| 8 | 303.0 | 7.0 | 10 | 10 | -1 |
| 9 | 353.0 | 97.0 | 5 | 5 | -1 |
| 10 | 300.0 | 5.0 | 2 | 2 | -1 |
| 11 | 395.0 | 30.0 | 2 | 2 | -1 |
| 12 | 371.0 | 10.0 | 2 | 2 | -1 |
| 13 | 357.0 | 19.0 | 2 | 2 | -1 |
| 14 | 353.0 | 43.0 | 3 | 3 | -1 |
| 15 | 349.0 | 30.0 | 13 | 13 | -1 |
| 16 | 346.0 | 63.0 | 9 | 9 | -1 |
| 17 | 378.0 | 17.0 | 2 | 2 | -1 |

InvCap heuristic

| | |
|---|---|
| Project Box | ProjectBox (instance 0) |
| Maximum Weight | 20 |

Evolutionary Algorithm – SPEA2

| | |
|---|---|
| Project | ProjectBox (instance 0) |
| Demands | Demands (instance 1) |
| DelayRequests | DelayRequests (instance 0) |
| Population Size | 100 |
| Archive Size | 100 |
| Number of generations | 10 |

Edge Topology Table
Number of Edges: 57

search : Case sensitive Whole word Save Options

| Id | From | To | Length | Delay | Bandwidth | ASfrom |
|---|---|---|---|---|---|---|
| 0 | 15 | 8 | 51.42956... | 0.171550... | 2759.506... | -1 |
| 1 | 15 | 23 | 70.34202... | 0.234635... | 6579.671... | -1 |
| 2 | 8 | 23 | 95.21029... | 0.317587... | 1162.660... | -1 |
| 3 | 16 | 23 | 38.07886... | 0.127017... | 5708.234... | -1 |
| 4 | 16 | 15 | 33.13608... | 0.110530... | 9753.065... | -1 |
| 5 | 7 | 8 | 30.67572... | 0.102323... | 4741.441... | -1 |
| 6 | 7 | 15 | 21.40093... | 0.071385... | 4178.701... | -1 |
| 7 | 22 | 16 | 25.05992... | 0.083590... | 7841.994... | -1 |
| 8 | 22 | 15 | 51.47815... | 0.171712... | 7985.115... | -1 |
| 9 | 9 | 15 | 67.11929... | 0.223885... | 3812.701... | -1 |
| 10 | 9 | 7 | 82.71033... | 0.275891... | 9163.795... | -1 |
| 11 | 21 | 23 | 82.56512... | 0.275407... | 6163.810... | -1 |
| 12 | 21 | 7 | 25.07987... | 0.083657... | 10297.29... | -1 |
| 13 | 6 | 15 | 51.85556... | 0.172971... | 8312.350... | -1 |

Edge ID = 36
From = 12, To = 8
Bandwidth = 10005.89
Delay = 0.23
Length = 68.07

Weights
Creation method: SPEA2

| Node 0 | Node 1 | Node 2 | Node 3 | Node 4 | Node 5 | Node 6 | Node 7 | Node 8 |
|---|---|---|---|---|---|---|---|---|
| 0.0 | 16.0 | 0.0 | 19.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 12.0 |
| 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 12.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| 0.0 | 0.0 | 0.0 | 12.0 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| 0.0 | 0.0 | 0.0 | 0.0 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| 0.0 | 0.0 | 0.0 | 0.0 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| 0.0 | 0.0 | 14.0 | 0.0 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| 0.0 | 0.0 | 18.0 | 0.0 | 0.0 | 0.0 | 0.0 | 19.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 8.0 |
| 0.0 | 14.0 | 0.0 | 0.0 | 0.0 | 0.0 | 15.0 | 0.0 | 0.0 |

Progress...
Working

Fig. 3. Screenshots of the application: a) Main application window; b) Weight generation heuristic input dialog; c) Evolutionary Algorithm input dialog; d) Edge topology view; e) Another topology view; f) Clipboard displaying the main datatypes; g) Weights table resulting from optimization.

existing optimization framework for efficient OSPF weight setting. This structured application has the responsibility of hiding the complexity of the problem from the user, possibly a network administrator without major programming skills, by creating an abstraction layer between the user and the system.

Another major requirement is modularity. As described in the previous sections, there are different methods and algorithms which can be applied on the optimization job. Some of those have been implemented, many others can be developed and easily explored and plugged into the application, provided that those new functions meet the specified Application Programing Interface (API). To achieve the solution for the problem, the user has to handle different types of data, such as network topology, demands and delay requests, among others.

Software functionalities or available actions are represented as *operations*. When an operation is called, its interface is launched and the input data objects are selected. After being triggered, an operation typically creates an instance of an output datatype. The required application features were mapped in operations, divided in different groups, easily accessible in the graphical interface. Those are listed below:

- File
  - New project from files - Creates a project from text files specifying the *nodes* and *links* details.
  - Random Demands/Delay Requests - Generates a *Demands/DelayRequests* instance, based on a scale parameter. Allows to generate data for benchmark-

ing.

  – Load/Save - Load or Save data files (Demands, DelayRequests, OSPFWeights).

- Simulation
  – Weight Generation - Creates OSPFWeights based on the network topology, using different heuristics (InvCap, Unit, L2, Random).
  – Simulate Scenario - Computes resulting Loads and Delays, based on the topology and selected Demands, DelayRequests and OSPFWeights.
- Optimization - Different optimization algorithms compute OSPFWeights, based on the selected parameters.
- Serialization - Load or Save objects using Serialization.

### B. Implementation

Both the optimization framework and the application are fully implemented in the Java language, which is being increasingly used by the scientific community in the area and has the advantage of being platform independent.

The application is entirely built on top of AIBench [11], a software development framework that was born as a collaborative project between researchers from the University of Vigo and the University of Minho. AIBench is a lightweight, non-intrusive, MVC-based Java application framework that eases the connection, execution and integration of operations with well defined input/output, completely fitting on the optimization problem being addressed. The platform was particularly conceived to facilitate the development of a wide range of research applications based on general input–processing–output cycles, where the framework acts as the glue between each executed task.

Building applications over AIBench brings important advantages to both the developers and the users, given its design principles and architecture. AIBench based applications tend to follow the Model-View-Controller (MVC) design pattern. This leads to units of work with high coherence that can easily be combined and reused. Furthermore, it is plug-in based: applications are developed adding components, called plug-ins, each containing a set of AIBench objects. This allows reusing and integrating functionality of past and future developments based on AIBench.

In order to provide the basis for supporting rapid application development, AIBench manages three key concepts that are present in every AIBench application: operations, data-types and views. The developer only needs to concentrate on how to divide and structure the problem-specific code into objects of these three entities. The framework will carry out the rest of the work to generate a completely runnable final application. These tasks include:

- Producing a GUI under which the user is allowed to select and execute the implemented functionality.
- Automatically retrieving the user parameters of a given operation whenever it is needed.
- Running operations, gathering the results and keeping them available for further use.
- Displaying the results through custom (or default) views.
- Keeping track of all executed operations together with the information needed to repeat the same (or modified) workflow in the future.

Software development has taken as a first premise to build a tool aimed at network administrators and not at computational or programming experts. Thus, the primary goal in the development process was to provide good usability for the final user.

As previously stated, every AIBench application is divided into three kinds of components: operations, implementing the algorithms and data processing routines; data-types, storing relevant problem-related information; views, rendering data-types obtained from executed operations. Based on these concepts, a user-friendly GUI was developed. The layout of the components can be observed in Fig.3a).

The clipboard, Fig.3f), keeps all data objects created within the application, in a logical hierarchy, grouped by their datatypes. The root of this tree is the *ProjectBox* container, that keeps a list of instances, representing different problems.

The components of a project are graphically shown in the form of explicit hierarchical containers, namely:

- The *Network Topology* includes information about nodes, edges, capacities, and all the network details;
- The *Demands Box* and *Delay Requests Box* hold one or more instances of *Demands* or *DelayRequests*, respectively.
- *OSPFWeights*, hold sets of OSPF weights, one per each link in the network. These can be loaded from files or generated by the implemented operations, are grouped in the OSPFWeights box.
- Both the *ResultSimul* and *ResultOptim* aggregate the resulting information of the operations.

When an object in the clipboard is double-clicked, the views corresponding to its datatype will be launched on the right side of the working area (if more than one view is available, those are accessible in different tabs). Examples of two views of the network topology are shown in Figures 3d) and 3e).

All the available operations are easily accessible, either through the menu in the top or by right clicking the item in the clipboard area. Snapshots of simulation and optimization operation input dialogs are shown in Fig.3b) and Fig.3c), respectively.

As previously mentioned, operation outputs are grouped together in the respective *ResultBox*. Fig.3g) shows an example of the optimization algorithm *SPEA2*, in this case the resulting weights table.

All operations are, at the maximum possible level, default-oriented, thus hiding behind scenes their complexity (e.g. definition of non-obvious parameters). Nevertheless, they allow more advanced users to fine-tune the parameters available to a given operation.

The optimization part of the application makes use of JECoLi, an open-source Java-based library for the implementation of metaheuristic optimization algorithms with a focus on Genetic and Evolutionary Computation based methods [16]. JECoLi has been/is being used in several research projects that share similar optimization needs, ranging application fields from Bioinformatics to Data Mining.

The graphical presentation of the network topology (Fig.3e) was produced using Jung [12], a software library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network.

### C. Availability

The software is made available, together with other resources, in the home page accessible at http://darwin.di.uminho.pt/netopt/. Readers have access to the source code and different releases of the application, which is still under development. Thanks to the platform independence of Java, the only software requirement is Java JRE 1.6.

### D. Case Study and documentation

The developed application benefits from the well-defined structure of AIBench. The distinction of the components between operations, data-types and views makes utilization easier and more efficient. Yet, there is a flexible workflow that shall be followed to reach these results. Logically, the user starts by loading the initial datatypes (like network topology and demands) before triggering the operations. Finally, the results can be displayed in different ways or saved to files, in order to improve users' understanding of the information.

Given the space restrictions of this manuscript a full case study is detailed in the software documentation given in the project web site. This also includes a set of How To's that detail how the major operations can be achieved with the application.

## IV. Conclusion

This paper describes an user-friendly application, built on top of an existing optimization framework, that allows the improvement of QoS levels on IP networks. The presented way of optimizing traffic flow, using OSPF weights, is an important tool, with no need to modify the basic network model. It can be an easy way of guaranteeing adequate levels of QoS, avoiding the typical additional mechanisms and inherent complexity (e.g. MPLS). From the results observed in the tests, and the referenced bibliography, the EAs have proven to be, in general, capable of a very good performance. There is a significant gain when compared to common heuristics offered by routing protocols.

One of the interesting conclusions of this work is that AIBench guided the application development through a layered programming. The MVC philosophy, defended by this framework, forced a modular code, which clearly improves the development efficiency. The resulting application is scalable, which means that new components, such as datatypes or algorithms, can be easily plugged into the existing application, without any significant modification.

The supporting framework is currently under rapid development and the most recent functionalities may be integrated in the future. The class-based [15] and multicast [14] optimization mechanisms are two examples of new developments, which need integration. J. Pinho *et al.* have been working [17] on the parallelization of JEColi, the Java library that implements the EAs. This interesting feature will certainly improve the efficiency of the application.

## Acknowledgments

This work was supported by the grant UMINHO/BII/061/2009 and by the project PTDC/EIA/64541/2006, both funded by Fundacao para a Ciencia e Tecnologia (FCT). The authors would like to thank to Paulo Maia and Paulo Vilaça who kindly helped with their technical knwoledge.

## References

[1] Z. Wang. *Internet QoS: Architectures and Mechanisms for Quality of Service*. Morgan Kaufmann Publishers, 2001.

[2] B. Fortz and M. Thorup. Internet Traffic Engineering by Optimizing OSPF Weights. In *Proceedings of IEEE INFOCOM*, pages 519–528, 2000.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(269-271), 1959.

[4] P. Sousa, M. Rocha, M. Rio, and P.Cortez, Efficient ospf weight allocation for intra-domain qos optimization in *6th IEEE International Workshop on IP OPerations and Management, IPOM 2006, LNCS 4268, pages 37-48*, G. Parr, D. Malone, and M. O. Foghlú, Eds. Springer-Verlag, 2006.

[5] M. Rocha, P. Sousa, P. Cortez and M. Rio, Quality of Service constrained routing optimization using Evolutionary Computation, *Applied Soft Computing*, 11(1), pages 356-364, Elsevier (Jan. 2011)

[6] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. *Computer Communication Review*, 32(4):161–176, 2002.

[7] A. Davy, D. Botvich, and B. Jennings. An efficient process for estimation of network demand for qos-aware ip networking planning. In G. Parr, D. Malone, and M. Foghlú, editors, $6^{th}$ *IEEE International Workshop on IP OPerations and Management, IPOM 2006, LNCS 4268*, pages 120–131. Springer-Verlag, 2006.

[8] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows*. Prentice Hall, 1993.

[9] J. Moy. RFC 2328: OSPF version 2, April 1998.

[10] T.M. ThomasII. *OSPF Network Design Solutions*. Cisco Press, 1998.

[11] D. Glez-Peña and M. Reboiro-Jato and P. Maia and M. Rocha and F. Díaz. and F. Fdez-Riverola. AIBench: A rapid application development framework for translational research in biomedicine, *Computer Methods and Programs in Biomedicine*, 98(2), pages 191-203, May 2010.

[12] Jung Java Library http://jung.sourceforge.net/

[13] J. Moy. *OSPF, Anatomy of an Internet Routing Protocol*. Addison Wesley, 1998.

[14] P. Sousa, M. Rocha, P. Cortez and M. Rio. Multiconstrained Optimization of Networks with Multicast and Unicast Traffic. *Management of Converged Multimedia Networks and Services*, Springer, LNCS 5274, pages 139-150, 2008.

[15] P. Sousa, M. Rocha, M. Rio and P. Cortez. Class-Based OSPF Traffic Engineering Inspired on Evolutionary Computation, *Wired/Wireless Internet Communication*, pp. 141-152, Springer-Verlag , 2007.

[16] P. Evangelista, P. Maia and M. Rocha, Implementing Metaheuristic Optimization Algorithms with JECoLi, *Intelligent Systems Design and Applications(ISDA 2009)*, pages 505-510, 2009

[17] J. Pinho, M. Rocha and J. L. Sobral. Pluggable Parallelization of Evolutionary Algorithms Applied to the Optimization of Biological Processes *18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pp.395-402, 2010