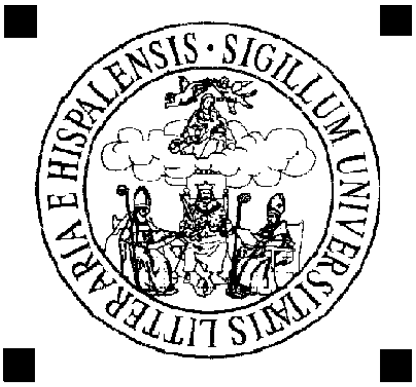
UNIVERSITY OF SEVILLE



Department of Applied Mathematics I

ANT COLONY ALGORITHMS

FOR

MULTIPLE OBJECTIVE COMBINATORIAL OPTIMIZATION

APPLICATIONS TO THE MINIMUM SPANNING TREES PROBLEMS

Pedro Jorge Sequeira Cardoso

December 2006

UNIVERSITY OF SEVILLE

Department of Applied Mathematics I

Ant Colony Algorithms

for

Multiple Objective Combinatorial Optimization

Applications to the Minimum Spanning Trees Problems

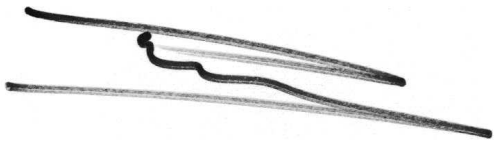
A dissertation presented

by

Pedro Jorge Sequeira Cardoso

in partial fulfilment of the requirements for the degree of

Doctor by the University of Seville



Doctor Mário Carlos Machado Jesus
(Ph.D. in Computer Science,
Professor in the Department of
Civil Engineering from
University of Algarve)

Doctor Alberto Márquez Pérez
(Ph.D. in Mathematics,
Professor in the Department of
Applied Mathematics I from
University of Seville)

Seville, December 2006.

ABSTRACT

Pedro Jorge Sequeira Cardoso, Ph.D.

University of Seville, 2006

The study of meta-heuristic solutions based on the Ant Colony Optimization (ACO) paradigm for the Multiple Objective Minimum Spanning Trees and related combinatorial problems is the main concern of this investigation.

In the commonly accepted complexity scale for problems, the Multiple Objective Minimum Spanning Trees is rated as an $\mathcal{NP}$-complete problem. Furthermore, as in the generality of the multiple objective optimization problem, the solution of the Minimum Spanning Trees case is a set of trade-off solutions in the sense that to improve one of the objectives it is necessary to worse at least one of the others, which is a major concern in a practical point of view.

In the first part of the investigation, a theoretical analysis of the problem is made to complement the known results. This analysis corroborates the fact that in practice the use of exact methods to solve the Multiple Objective Minimum Spanning Trees problems is only applied in specific circumstances. This implies that to solve the problem an approximation method must be considered as an alternative. In particular, two methods based on the ACO paradigm are proposed: the Multiple Objective Network optimization based on an ACO (`MONACO`) and the ϵ-Depth ANT Explorer (ϵ-`DANTE`). The `MONACO` method uses a set of pheromone trails and specific heuristics to approximate the Pareto set. The ϵ-`DANTE` method is an improvement of the `MONACO` method that uses a depth search procedure, based on the best performing solutions, to deeply exploit the search space.

The proposed methods are tested with selected multiple objective problems, im-

proving the results previously obtained by other authors. To test the MONACO and ϵ-DANTE algorithms over the Multiple Objective Minimum Spanning Trees problem is proposed a library/repository of multiple objective network problems established over a systematized set of networks generators. The results obtained with MONACO and ϵ-DANTE are then compared with the results obtained with a Brute Force and the Weighted Sum method.

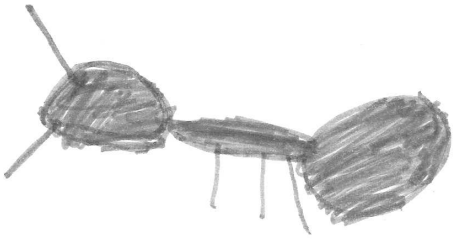
Biographical Sketch

1996	Graduate in Mathematics, specialization in Computer Science, by the University of Coimbra.
1997-98	Assistant Professor at the Portuguese Catholic University.
1998-...	Assistant Professor at University of Algarve.
1999	MSc in Computational Mathematics by the University of Minho.
2001	Registration on a Discrete Mathematics PhD program at University of Seville.
2003	Qualification of *Suficiencia Investigadora* by the University of Seville.

Ao meu filho João Pedro,

à minha filha Margarida,

e à minha esposa Cláudia.



MARGARIDA
04-06-2006

Acknowledgements

I must say it: "Yeeees! I'm writing the acknowledgements!". Why? Well, although it might be one of he first things you see of this thesis, it was one of the last things that I wrote. Besides, it gives me the opportunity to acknowledge a group of people to whom I'm deeply grateful because without them I would certainly never reached this moment.

Special thanks are due to my advisors, Professor Mário Jesus and Professor Álberto Márquez, for their friendship, invaluable help and guidance, incentive, permanent readiness, careful reading of the manuscript, and consequent constructive analysis.

À minha esposa Cláudia pela sua paciência e apoio, e pela graça de me ter dado duas crianças maravilhosas, que sempre me conheceram com estudante de Doutoramento, às quais prometo: "Margarida e João Pedro, o pai vai deixar de estar resmungão quando acorda! Bom, pelo menos tão frequentemente...". Aos meus pais João Cardoso e Maria da Conceição Cardoso pelo seu contínuo amor, incentivo e apoio. Em geral, aos meus familiares e amigos pelo sempre presente "Já está?" (Agora vamos poder falar sobre o tempo).

To the School of Technology of the University of Algarve, in particular to the Department of Electrical Engineering, the Department of Civil Engineering, and the

Centro de Simulação e Cálculo (CsC) for the resources made available.

To my working colleagues for their incentive. In particular, to my office colleagues Jânio e Ana Bela for talking to me even when I was miles away. To Pedro Guerreiro for providing me help in the Linux configurations and for being a good friend.

Appreciations are also due to Horst Hamacher, Andrzej Jaszkiewicz, Joshua Knowles, Ray Koopman, Francesco Maffioli, and Oliver Schütze for so kindly replying to my questions and providing me assistance in different subjects.

Finally, I would like to thank the support provided by the ORI project of Spanish Ministry of Science and Technology.

Contents

List of Tables

List of Figures

List of symbols

$\preceq$	...	Weakly dominates
$\prec$	...	Dominates
$\prec\prec$	...	Strictly dominates
$\sim$	...	Incomparable
CEG	...	Complete Edges Generator
$ClEG$	...	Clusters Edges Generator
$ClNG$	...	Clusters Nodes Generator
$C_T(e)$	...	Cycle induced by adding e to tree T
CWG	...	Concave Weights Generator
$\rho - CWG$	...	ρ-Correlated Weights Generator
DEG	...	Delaunay Edges Generator

ϵ-DANTE	...	ϵ Depth ANT Explorer
$\mathcal{E}$	...	Set of edges
$\mathcal{E}_T$	...	Set of edges of network T
$\mathcal{E}\|_T$	...	Set of edges defined in $\mathcal{N}$ by $\mathcal{V}_T$
$\mathcal{E}_u$	...	Set of edges with origin u
$\mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)$	...	Set of the edges defined between $\mathcal{V}_1$ and $\mathcal{V}_2$
$\widehat{\mathcal{E}}(\mathcal{V}_1, \mathcal{V}_2)$	...	Set of the non-nominated edges defined between $\mathcal{V}_1$ and $\mathcal{V}_2$
e_{uv}	...	Edge of $\mathcal{E}$ defined by nodes u and v of $\mathcal{V}$
GEG	...	Grid Edges Generator
GNG	...	Grid Nodes Generator
HEG	...	Hexagonal Edges Generator
$k - CEG$	...	k-Convex Edges Generator
MONACO	...	Multiple Objective Network optimization based on an ACO
m	...	Number of objectives
M	...	The set $\{1, 2, \ldots, m\}$
$\mathcal{N}$	...	Network ($\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z})$)
$\mathcal{N}\|_T$	...	Sub-network induce by T, $(\mathcal{V}_T, \mathcal{E}\|_T, \mathcal{Z})$

NNG	...	Normal Nodes Generator
$\mathcal{P}^*$	...	Pareto Set
$\mathcal{P}$	...	Approximation set
$\mathcal{Q}$	...	Approximation set
$\mathcal{R}$	...	Approximation set
RWG	...	Random Weights Generator
$\mathcal{S}$	...	Feasible set
TEG	...	Triangle Edges Generator
TNG	...	Triangular Nodes Generator
UNG	...	Uniform Nodes Generator
$\mathcal{V}$	...	Set of nodes
$\mathcal{V}_T$	...	Set of nodes of subnetwork T
VEG	...	Voronoi Edges Generator
$\mathcal{W}$	...	Weight function ($\mathcal{W} : \mathcal{S} \rightarrow \mathbb{R}^m$)
w_i	...	i-Weight component of $\mathcal{W}$
X	...	Solution ($X \in \mathcal{S}$)
Y	...	Solution ($Y \in \mathcal{S}$)

$\mathcal{Z}$	...	Edge weight function ($\mathcal{Z} : \mathcal{E} \rightarrow \mathbb{R}$)
z_i	...	i-Weight component of $\mathcal{Z}$

Remember tonight...

for it is the beginning of always.

Dante Alighieri

1

Introduction

Contents

1.1 Overview

In the last decades, the technological advances of the computational capacities in conjunction with the decreasing prices of the informatics apparatus placed the humanity into a new era of scientific knowledge. Those progresses, combined with the networks of knowledge, like the Internet or the media, allowed a much faster dissemination of the accomplished technical developments.

However, newer and pushing limits are being imposed to those networks in areas

like security, capacity/velocity, delays, reliability and accuracy.

Behind the viewable face of this Computer Science revolution, investigators all over the world are using the available computational capacities to develop new methods, most of them based in large numerical computations, to solve the challenging proposed problems. Those methods can be decomposed in several sub-fields, each one embracing important areas of investigation, like

- **Design** – Process to originate and develop a project associated to some concept. For instance, the several existing Computer-Aided Design (CAD) tools help the developers in the project phase of complex systems, allowing at the same time to have a perspective of the developed plans.

- **Visualization** – Techniques to create images, diagrams, or animations that are used to support decisions. For example, the use of virtual reality together with seismic studies to find the best places to drill oil and gas wells [Hodgson, 2002].

- **Simulations** – Methods used to model real-life situations on a computer. For instance, the use of flight simulators or the traffic simulation (that permits the preview of the consequences of changes introduced to circulation plans or unexpected occurrences) are examples of applications that reduce the costs and risks associated with these activities.

- **Optimization** – Common to most of the engineering processes, uses computational methods to make the most of the available resources.

All this cases are nowadays a common working procedure. However, the last point will be precisely the main problem to be discussed in this thesis.

1.2 Optimization Problems

In the *The American Heritage Dictionary of the English Language* [AHD, 2000] optimization is defined as

> *"The procedure or procedures used to make a system or design as effective or functional as possible, especially the mathematical techniques involved."*

This definition characterizes optimization as a set of actions applicable to any practice or object to be improved. In fact, optimization arises in the most distinct and demanding areas of knowledge like all fields of engineering, logistics, economics, medicine and biology.

The inherent complexity in the behaviour of the above mentioned areas makes extremely difficult to adequately model them. Furthermore, in most of the modelled problems parameters, like the number of variables, dynamics, noises, restrictions, number of objectives, time, or computational capacity limitations, can make the search for a simple solution a very demanding task. However, finding a good solution is usually a much more challenging or even an impossible assignment.

For example, suppose that our objective is to connect the 100 biggest cities of the Iberian Peninsula, using the shortest possible quantity of a certain optical fibre cable – the classical Spanning Tree Problem [Borůvka, 1926; Nešetřil, 1997]. Obtaining a solution is an easy task (after selecting the 100 biggest cities!). To compute it, we can simply put 100 papers with the names of the selected cities in some ballot box and start picking papers. Then, the cities are connected following the order in which they were selected. Now, we just have to hope that this was a good choice out of the 10^{196} possibilities.

Certainly the previous solution is improved if a wiser criterion is used in the choice of the connections between the cities. For example, the process can start by computing and sorting the distance between each pair of cities. Then a tree is built through the addition of the cheapest connections that keeps the feasibility, that is, it does not introduces any cycle. Comparing both solutions, the last one has a higher computational overhead but certainly decreases the necessary cable length. In fact, the sketched method describes an algorithm to obtain an optimum solution for the proposed problem (see for example [Jungnickel, 1999]). For the 100 cities, and even for a higher number of cities, the procedure is certainly executable by hand in a large limited time window.

If some restrictions are included the problem can become much harder to solve. Suppose that no city can be connected to more than three neighbour cities. A feasible solution is easily attainable using the same procedure described above with the ballot box. Like before this is a very rudimentary process and the solution is not expected to be a good one.

Obviously, these circumstances also allows the use of the procedure that successively adds the cheapest connections that maintain the feasibility, that is, it does not introduces any cycle and no city is directly connected to more than three neighbour cities. However, probably it will not be the best possible solution. It is well known that the class of problems described (Minimum k-Degree Spanning Tree Problem) is intricate for hand computation and requires the use of complex strategies to achieve the optimum solution in feasible time, even for relatively small problems [Ausiello *et al.*, 1999, p. 393].

Therefore, in an optimization problem, the best possible solution has different interpretations according to the bearable costs associated with its computation. Conditions, like time and computational capacities, restrict the aptitude to reach global or even

local optimums.

Imagine another example. Now, the objective is to choose a kindergarten for your children. Several objectives are to be fulfilled, like the

- Quality of the installations;
- Quality of the interior equipments;
- Quality of the playground equipments;
- Quality of the open green area;
- Experience of the educators;
- Easiness of parking for living and picking up the children;
- Distance to your home or your workplace;
- Working period;
- Monthly fee;
- Existing extra-curricular activities (swimming classes, informatics classes, foreign languages classes, ...); and
- Etc.

Those objectives are, in most cases, antagonistic. For example, if you want a place with extra-curricular activities you will probably have a higher fee.

For the sake of clarity, suppose that we have five possible schools and we are only considering three objectives, that are all somehow quantifiable from 1 (good) to 3 (bad), resumed in Table 1.1. From Figure 1.1, the radar chart for the values of Table 1.1, we can conclude that there is not a school that is better than the others in all the objectives. Like in most of the practical problems this circumstances make necessary

	Objective 1	Objective 2	Objective 3
School A	1	3	1
School B	2	2	2
School C	1	1	2
School D	3	2	3
School E	2	2	1

Table 1.1: School classification example.

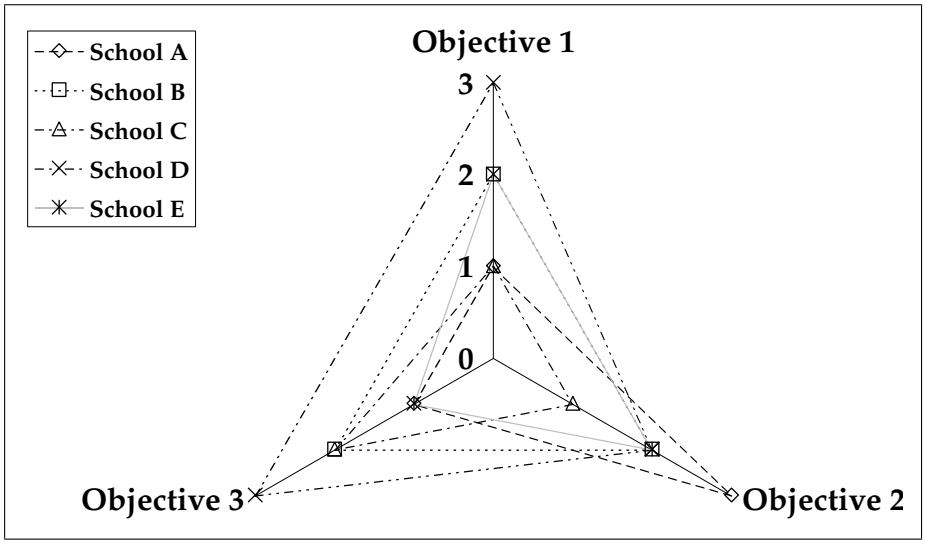


Figure 1.1: Radar chart of the schools classification example.

to decide between trade-off solutions. In fact, although we can discard Schools B and D since Scholl C is better or equal in all objectives; Schools A, C and E are not better than any of the others since

- C is worse than A and E in objective 3 (being better in objective 2); and
- A is better than E for objective 1 but worse in objective 2.

Before this kind solutions, where a set of trade-off answers is proposed, the choice will end up to be made by a decision maker, which in this case are the parents.

The last example can be considered simple, since only five feasible solutions and three objectives were considered. Returning to our initial problem of connecting the 100 biggest cities of the Iberian Peninsula with an optical fibre network (using a spanning tree), the dimensions and the intractability of the problem is much higher. Factor like distances, existing infrastructures, maximum/minimum nodes degree, network reliability and other installation and maintenance costs produce a large number of compromise solutions.

The last two proposed examples are classified as multiple objective optimization problems. In fact, realistic optimization problems require the simultaneous optimization of more than one objective, which brings new questions:

- How to balance all the options?
- What are the priorities?
- What to do in situations with equal priority?

Those and other questions have to be answered by the decision maker that, according to the conjecture, has to decide which solution to choose among all the trade-off proposed ones. Nevertheless, we are particularly interested in those solutions that are not worse than any of the others in all considered objectives. Those best solutions are called Pareto solutions and the set of all Pareto solutions is called Pareto or efficient set. This set contains the best trade-off performing options in the sense that to improve an objective it is necessary to worse at least one of the others.

Consequently, an optimization problem has one basic component, called objective or weight function

$$\mathcal{W} : \mathcal{F} \to \mathcal{O},$$

where $\mathcal{F}$ is the feasible set of solutions (with the restrictions included) and $\mathcal{O}$ is the objective space. $\mathcal{O}$ is usually a subset of $\mathbb{R}^m$, where m are the number of considered

objectives. The solution of the problem are the elements of $\mathcal{F}$ that have optimum values (maximum or minimum) in $\mathcal{O}$, according to some predefined order relation [Deb, 2001].

Optimization problem can be classified in two main classes: continuous and discrete. In the continuous case, $\mathcal{F}$ is a continuous set (like intervals). In the discrete case, which include the combinatorial problems, the variables are limited to assume only discrete values ($\mathcal{F}$ is a discrete set).

In the remaining document, unless stated otherwise, it will be considered only multiple objective combinatorial optimization problems, mainly directed to network optimization cases.

1.3 Heuristics and Meta-Heuristics

The use of heuristics is strictly connected to the computation of approximations to optimal solutions. In [Barr *et al.*, 1995] an heuristic is defined as

> "An heuristic method (also called an approximation algorithm, an inexact procedure, or, simply, a heuristic) is a well-defined set of steps for quickly identifying a high-quality solution for a given problem (...)"

In fact, solving problems using heuristics has always been a part of the human way, like the process described to construct the 3-degree spanning tree between the 100 cities in the previous section, where the solution is not necessarily the optimum.

Since the endings of the 1960′s decade, a set of new heuristic optimization trends are being developed. Those methods are in their majority based in stochastic (pseudo-random) processes that delineate general optimization procedures, usually designated as meta-heuristics.

More specifically, the (stochastic) meta-heuristics methods are characterized for being non-deterministic procedures. These optimization processes are general algo-

rithmic frameworks, defined using non-problem specific information, which allow their application to large classes of problems with relatively few modifications.

Frequently, the understanding of those meta-heuristics is helped by the use of metaphors inspired by the surrounding world. The observation of successfully adapted organism in particular environments or the observation of physical processes has the ability to suggest new methodologies or algorithms.

Some examples of those nature inspired meta-heuristics are the

- Ant Colony Optimization – uses the ants' colonies behaviour as a metaphor to describe a process based in a population of agents and layers of numerical values (pheromone trails) to guide in the construction of the solutions. This communication process is called stigmergy since it is done by the change of the surrounding environment [Dorigo & Stutzle, 2004; Dorigo *et al.*, 1999].

- Genetic Algorithms – the main procedure is a simulation of the biological evolutionary Darwinian theories. Those algorithms are probably the best studied among the meta-heuristic methods, with several variants proposed. In general, the procedure is constituted by a global search technique that successively improves the known best performing solution (known as individuals) through the use of three basic operators: mutation, selection and crossover. Those operators allow the inheritance of the information from iteration to iteration (through individuals), avoid the stagnation of the procedure in some local optimum and discard the less promising solutions [Vose, 1999].

- Simulated Annealing – based in the idea of the metallurgic annealing, the process evolves from solution to solution using a local search strategy. A parameter called temperature controls the probability of moving to worse solutions, which avoids getting stuck in some local optimum. As the method advances, the temperature

parameter disallows more often moves to worse solutions [Kirkpatrick *et al.*, 1983].

- Particle Swarm Optimization – these methods reproduce the behaviour of the flocks of birds or fish schooling. A set of agents (particles) move through the search space following the best performing solution [Eberhart & Kennedy, 1995a; Eberhart *et al.*, 2001].

- Stochastic Diffusion Search – this meta-heuristic is another population based search method that uses a set of simple agents to execute simple computations steps. The information obtained by each agent is communicated directly to other agents, instead of the stigmergetic communication used in the Ant Colony Optimization methods. As a result this diffusion mechanism implies the appearance of clusters of agents that keep the information of regions where high quality solutions are located [Bishop *et al.*, 2002; De Meyer, 2003].

Methods like Ant Colony Optimization, Particle Swarm Optimization, or Stochastic Diffusion Search, are all based in populations of simple agents that somehow have the ability to communicate among them. They are generally categorized in an overall meta-heuristic class designated as Swarm Intelligence.

Other heuristics like the Hill-Climbing, the Greedy Randomized Adaptive Search Procedure [Ribeiro & Resende, 2001], or the Tabu Search [Glover & Laguna, 1997], are also local search methods that successively try to improve the known solutions by movements in their vicinity. Generally, the procedure ends when a local-optimum is reached or certain amount of iterations is executed.

Some of these methods, like the Genetic Algorithms or the Ant Colony Optimization, have their convergence in probability to the global optimums proved, whenever some basic conditions are verified.

Above all, the widespread utilization of these methods in the most distinct academic and industrial applications is justified by the exceptional results that are being obtained, even when compared with the classical methods, for some of the most demanding optimization problems.

In Section 2.3 the essential methods used in this thesis will get a more detailed review.

1.4 Contributions

The main contributions of this thesis are the

- **Establishment of a set of theoretical results for the Multiple Objective Minimum Spanning Trees problem.** A part of these results is the generalizations of properties known for the single objective problem. Furthermore, it is presented an additional set of definitions and results that are specifically related to the multiple objective case. The analysis of the results, allows to present conclusions related to the difficulty of defining algorithms capable of computing the Pareto set or, in a more pragmatic mode, good approximation sets [Cardoso *et al.*, 2005b].

- **Establishment of a library/repository of network problems.** This contribution can be divided in two main parts [Cardoso *et al.*, 2006b]:

  - Systematization of the network generating process. One of the main objectives is to define a set of generating algorithms, capable of translating, as much as possible, real world situations. This is done by the combination of several possible network topologies and edges weights generators.

  - Associated to the problem is presented a set of network problems and their Pareto and/or approximation sets. Those instances will allow a faster clas-

sification of the performance in future algorithms, since it is avoided the endeavours associated to the settlement of representative libraries of problems to be tested.

- **Formulation of the MONACO – Multiple Objective Network optimization based on an ACO – algorithm.** This method is a Swarm Algorithm, which, as the name suggests, is particularly adaptable to the optimization of network problems with multiple objectives. The process relies on a set of pheromone trails, one for each objective, which are used to lead the stochastic search procedure to the more promising solutions. Tests made with an implementation of MONACO for three multiple objective optimization problems (the simulation of a network of flows, the Travelling Salesman Person and the Minimum Spanning Trees problem) show that it can be easily adapted to different classes of multiple objective problems, maintaining its efficiency and accuracy [Cardoso *et al.*, 2003a, b, 2004, 2005a]. Other authors adapted and compared MONACO with two Genetic Algorithms and other Ant Colony methods for the Multiple Objective Travelling Salesman Person problem [García-Martínez *et al.*, 2004a], concluding of its potential quality.

- **Development of the ϵ-DANTE – Depth ANT Explorer – algorithm.** This method is also a Swarm Algorithm that uses the same framework of MONACO, that is, it uses several pheromone trails to direct the stochastic search procedure. However, selected solutions are further explored through the employment of a depth search process, which is also based in the same stochastic pheromone mechanism, producing a hybrid algorithm.

Additional contributions are

- **A survey and discussion of the Multiple Objective Minimum Spanning**

Trees problem. This analysis is based on the bibliography examination of some of the more relevant documents, with special interest in the algorithmic solutions, the computational complexity and the theoretical results.

- **The improvement of results using ϵ-DANTE algorithm.** An implementation of the ϵ-DANTE algorithm is applied to three multiple objective problems: the Travelling Salesman Person problem, the k-Degree Minimum Spanning Trees problem and the Minimum Spanning Trees problem. The obtained results improve the results that were previously collected by other authors for the first and second of those problems. Relatively to the Minimum Spanning Trees case, the method improves, in general, the results obtained with some deterministic methods, which were used to approximate the Pareto set.

- **The proposal of a new Angle-Pheromone Update strategy associated to the ϵ-DANTE algorithm.** This strategy avoids noisy pheromone trails, which are common to the use, in the pheromone updating formula, of all the computed solutions or all the elements in the approximation set. More specifically, are used only subsets of the approximation set which, for example, in the bi-objective case corresponds to the elements that lie in a subangle of the angle defined by the origin and the extreme solutions weights. These updating formulas are generalized for problems with $m \geq 2$ objectives.

- **The use of a new strategy to observe the time evolution of the optimization process.** This method can use most of the performance metrics to sketch a time evolution graphics, which allows to estimate the speed of convergence toward a reference set and the required time to achieve an acceptable approximation set, for some defined computational environment.

Some extra contributions, which are not detailed presented here for the sake of simplicity and clearness, were accomplished during the research time associated to the development of the work presented in this thesis. For example, in [Cardoso & Jesus, 2004] it is planned one possible utilization of the developed methods, which includes the simulation of networks of flows in a multi-user framework.

1.5 Outline of the Thesis

This thesis is divided in seven chapters, which are organized as follows. In this first chapter was made a brief introduction to the multiple objective optimization problems, difficulties, trends and an outline of the main contributions of this thesis.

The second chapter contains basic concepts necessary to understand the following chapters. More precisely, it is made a survey of the basic definitions of the multiple objective optimizations problems, of the classical optimization techniques and the more recent stochastic trends in the optimization field. The chapter ends with the analysis of different algorithm performance metrics and the examination of some algorithmic properties related to the spanning trees problems. The aim of this chapter is to make this thesis self contained, which implies that the reader familiarized with the described concepts may skip into the next chapter.

The third chapter analyses the multiple objective minimum spanning trees problem, which includes a state of the art survey and the presentation of a set of theoretical results based in the edges efficiency relations. A discussion about the need to propose algorithmic alternatives to the existing ones ends the chapter.

In Chapter 4 it is presented a framework for a set of network generators. It includes a systematization of the process used to build distinct network problems. The process includes three generating steps associated to the nodes, the edges and the weights. Several sub-generators are proposed for each of those steps, which upon combination

allow to define various optimization difficulties that become intrinsically associated to the networks topologies and edges weights. Some Pareto and approximation sets are sketched in the final section, allowing to recognize interesting characteristics, like fronts with large concave regions or with gaps.

The fifth chapter is devoted to the presentation of two meta-heuristics: `MONACO` and ϵ-`DANTE` algorithms. First, the `MONACO` procedure is presented along with a set of results obtained for three multiple objective optimization problems: the simulation of a network of flows, the Travelling Salesman Person and the Minimum Spanning Trees. In the second part of this chapter, it is described the ϵ-`DANTE` procedure followed by presentation of the results that were obtained for a set of instances of the Multiple Objective Travelling Salesman Person and the Multiple Objective k-Degree Minimum Spanning Trees problems. It is demonstrated that in most cases the achieved solutions are, at least, comparable to previous known results. This chapter presents also the definition of the Angle-Pheromone Updating strategy, which is based in a subset of the elements of the approximation set that is in construction.

A detailed analysis of the results obtained with ϵ-`DANTE` for an extensive set of Multiple Objective Minimum Spanning Trees problems is made in Chapter 6. This analysis includes the use of several statistics associated to the performance metrics results.

Conclusions about the developed work and considerations about future research and possible trends are presented in the seventh and last chapter.

Maybe then I can finish my book.

Calvin & Hobbes

2

Preliminaries

Contents

2.1 Overview

Beyond the inherent complexity associated to contingencies like the dimensions, the topologies and their dynamics, real world networks conception and optimization problems are extremely intricate due to the multiple objectives to be fulfilled. Features like productivity, reliability, longevity, efficiency, or quality, are strategic aspects to be considered in the optimization of networks. For instance

- In a telecommunications network, where the solutions of many problems are based on spanning tree optimizations, exist various aspects that compete among each other. Among these factors are the maintenance and communication costs, the communication delays, the reliability, the bandwidth, and the profitability [Pinto *et al.*, 2005];

- In traffic networks, if you want to plan a car trip, yours choices might be pondered by distance, expected time, tolls and, others not so much quantifiable objectives, like possible scenic views, roads quality and accident risks. Solutions for this problem can, in some circumstances, be computed as the solution of the Travelling Salesman Person;

- In the planning of an irrigation network, we would like to optimize the necessary canals excavations and construction works, the canals/pipes length, the canals/pipes dimensions, the number of irrigated points. For instance, solutions for this problem might be computed using Steiner trees [Jesus, 2000; Jungnickel, 1999].

The obligation of accomplishing several conflicting simultaneous objectives, usually implies the non-existence of one single solution that can be considered as the optimum, since improving one objective will worse at least one of the others. As a consequence, a set of solutions that represents the best compromise between the conflicting objectives is called the efficient set. The final verdict of picking one of those optimal trade-off answers is left to a decision maker that has to rank them according to some criteria.

On the other hand, the computation of the efficient set of networks can be impractical, or even impossible, due to several circumstances, like the absence of efficient algorithms or to an exponential number of solutions [Hamacher & Ruhe, 1994]. This implies that a representative collection of efficient or quasi-efficient solutions can eventually be acceptable.

As already surveyed in the first chapter, there are two main classes of algorithm to solve an optimization problem: exact or approximation methods.

The generality of the exact methods are deterministic procedures. Meaning that under the same initial conditions they do precisely the same steps until the optimality criterion is verified. Nevertheless, problems may arise when the exact methods, whenever they exist, are not applicable for circumstances like their high algorithmic complexity (see Section 2.3.1) [Ausiello *et al.*, 1999; Bovet & Crescenzi, 1994].

As an alternative, the last decades extensive use of non-deterministic methods or meta-heuristics, one of the main concerns of this study, confirmed their potentialities by the establishment of optimum or quasi-optimum solutions in some of the most

demanding practical and academic optimization problems [Andersson, 2001; Bagchi, 1999; Bernal-Agustin, 1998; Deb, 2001].

This appealing area originates the development of a large number of methods, along with their variants, which implies that to achieve an empirical investigation of the quality of the different algorithms, it is necessary to be in the possession of performance metrics, capable of, under certain conditions, conclude of the possible overperformance of one algorithm over another.

Therefore, this chapter will start by discussing the basic concepts of the multiple objective problems: the definition of the problem and its optimality conditions. Next, the algorithmic theory is surveyed followed by a review of some of the more relevant deterministic and approximation methods for the combinatorial optimization field. This chapter ends with the analysis of a set of performance metrics that are used to estimate the quality of the obtained approximations and a review of the (single objective) spanning tree problem.

2.2 Multiple Objective Optimization

2.2.1 Problem Definition

The solution of a multiple objective optimization problem, as in any optimization problem, has associated two sets: the search space and the objective space. The search space or feasible set contains all the possible answers for an optimization problem. On the other hand, the objective space elements, associated through a function to the elements of the search space, allow the ranking of the solutions.

The function that estimates the weight of the feasible solutions is called objective function. In the multiple objective case, this function is a weight vector function defined by m components to be optimized. Mathematically, this function can be defined as

follows. In an optimization problem an **objective function**, $\mathcal{W}$, is defined as

$$\begin{array}{rccl} \mathcal{W}: & \mathcal{S} & \rightarrow & \mathcal{O} \\ & X & \mapsto & \mathcal{W}(X) = (w_1(X), w_2(X), \ldots, w_m(X)), \end{array} \tag{2.1}$$

where w_i are the components of $\mathcal{W}$, $\mathcal{S}$ is the **feasible set** or **search space**, $\mathcal{O} \subset \mathbb{R}^m$ is the **objective space** and X is a solution. For example, the components of $\mathcal{W}$ can be defined as

$$w_i(X) = \sum_{x \in X} z_i(x), i = 1, 2, \ldots, m \tag{2.2}$$

or

$$w_i(X) = \max_{x \in X} z_i(x), i = 1, 2, \ldots, m, \tag{2.3}$$

where x is an element of a feasible solution X and $z_i(x)$ is the i-weight contribution of x to the overall weight. The objective functions defined in (2.2) and (2.3) are usually denominated as **Sum Objective function** and **Bottleneck Objective function**, respectively.

Next is presented a more specific example.

Example 2.1: If the network N, in Figure 2.1, is the solution of some optimization problem and the weights of the edges, $(z_1(e), z_2(e))$, are given in Table 2.1, then the objective function can be defined as the

- Sum objective function

$$\begin{aligned} \mathcal{W}(N) &= (w_1(N), w_2(N)) \\ &= \left(\sum_{e \in N} z_1(e), \sum_{e \in N} z_2(e) \right) = (32, 36); \end{aligned} \tag{2.4}$$

- Bottleneck objective function

$$\mathcal{W}(N) = (w_1(N), w_2(N)) \tag{2.5}$$

$$= \left(\max_{e \in N} z_1(e), \max_{e \in N} z_2(e) \right) = (7, 5). \tag{2.6}$$

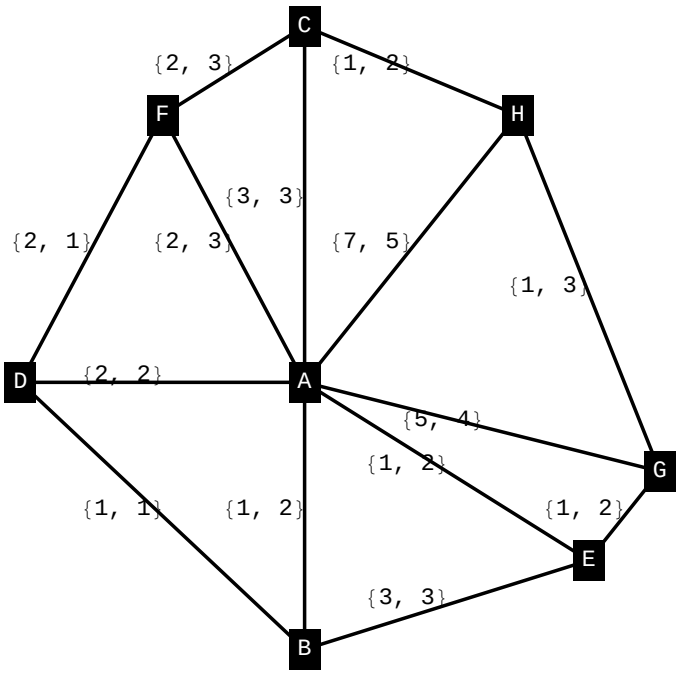


Figure 2.1: Network example.

Given the objective function, it is now possible to define a multiple objective optimization problem, which can be stated as

$$\begin{aligned} &\text{optimize} \quad \mathcal{W}(s), \\ &s \in \mathcal{S} \end{aligned} \tag{2.7}$$

where optimize means that we either want to minimize or maximize each of the m objectives, $w_i (i = 1, 2, \ldots, m)$.

In this thesis, unless stated otherwise, it will always be considered that all objectives are to be minimized, denoted as

$$\begin{aligned} &\min^{+} \quad \mathcal{W}(s). \\ &s \in \mathcal{S} \end{aligned} \tag{2.8}$$

Note that if some of the objectives were to be maximized, we could always use the duality principle that states the equivalence of the original problem to one where all objectives are to be minimized [Deb, 2001].

	A	B	C	D	E	F	G	H
A		(1,2)	(3,3)	(2,2)	(1,2)	(2,3)	(5,4)	(7,5)
B	(1,2)	–	–	(1,1)	(3,3)	–	–	–
C	(3,3)	–	–	–	–	(2,3)	–	(1,2)
D	(2,2)	(1,1)	–	–	–	(2,1)	–	–
E	(1,2)	(3,3)	–	–	–	–	(1,2)	–
F	(2,3)	–	(2,3)	(2,1)	–	–	–	–
G	(5,4)	–	–	–	(1,2)	–	–	(1,3)
H	(7,5)	–	(1,2)	–	–	–	(1,3)	–

Table 2.1: Weights of the edges of network in Figure 2.1.

Using the previous definitions, the optimization problem $\min^{+}_{N\in\mathcal{N}} \mathcal{W}(N)$ can be stated as

$$\min_{X\in\mathcal{S}}{}^{+} \left(\sum_{x\in X} z_1(x), \sum_{x\in X} z_2(x), \dots , \sum_{x\in X} z_m(x) \right) \tag{2.9}$$

for the sum objective function and as

$$\min_{X\in\mathcal{S}}{}^{+} \left(\max_{x\in X} z_1(x), \max_{x\in X} z_2(x), \dots , \max_{x\in X} z_m(x) \right) \tag{2.10}$$

for the bottleneck objective function.

The problem defined in (2.10) is known as the max-ordering problem (often also known as the minimax problem) and has applications in the game theory. During the remaining of this document we will consider the optimization problem defined in (2.9).

Considering the previous example with network N, in Figure 2.1, and the shortest path problem between nodes A and H. There are several admissible solutions, like $A-H$, $A-C-H$, or $A-G-H$. The weights of those paths are $\mathcal{W}(A-H) = (7,5)$, $\mathcal{W}(A-C-H) = (4,5)$, and $\mathcal{W}(A-G-H) = (6,7)$, meaning that $A-C-H$ is obviously better than the other two.

On the other hand, if we think of paths between B and E, examples of feasible solutions are $B - E$ and $B - A - E$. In this case, the weights are $\mathcal{W}(B - E) = (3, 3)$ and $\mathcal{W}(B - A - E) = (2, 4)$. Therefore, unless some objective preference is settled, none of them is better than the other.

In fact, in the multiple objective optimization situations like the previous one, where it is not possible to say that one solution over performs another, are common. This requires the definition of an order relation to allow the partial ranking of the solutions, as the one that will be introduced in the next section.

2.2.2 Dominance and Pareto Optimality

An optimization process requires the definition of an order relation capable of comparing two solution X and Y. This comparison has three possible results: X is better than Y, Y is better than X, or none of the solutions is better than the other.

In the first two cases the solution that is better than the other is, usually, referred as the dominating solution. The last case includes two distinct situations: when $\mathcal{W}(X)$ is equal to $\mathcal{W}(Y)$ and when $\mathcal{W}(X)$ is different from $\mathcal{W}(Y)$.

To compare two solutions the more commonly used relations in the multiple objective optimization area is due to Vilfredo Pareto. These relations assume that no objective is more important than any of the others and are defined as follows [Knowles, 2002; Zitzler *et al.*, 2003]. Given two solutions X and Y of the feasible set $\mathcal{S}$ it is said that

- X **weakly dominates** Y, $X \preceq Y$, if

$$\forall_{i \in \{1,2,\ldots,m\}} : w_i(X) \leq w_i(Y); \tag{2.11}$$

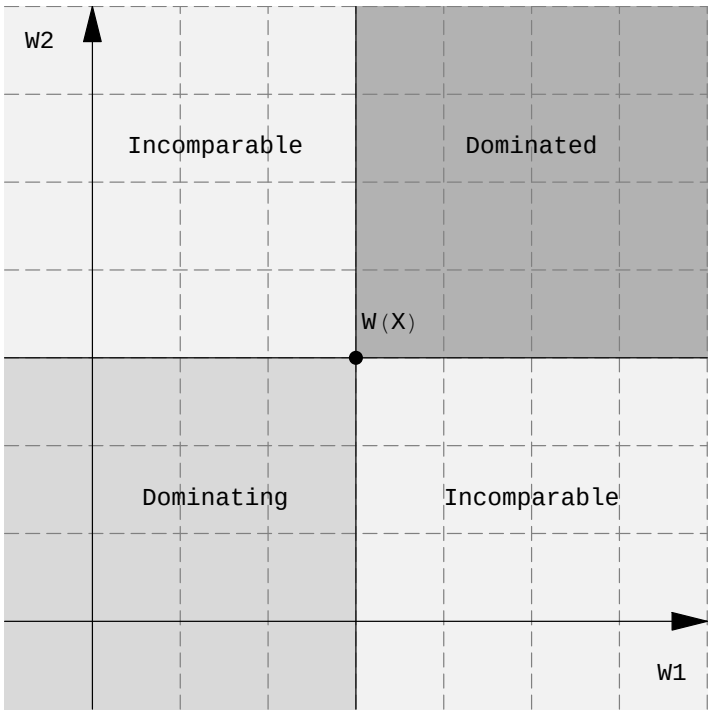


Figure 2.2: Dominated, dominating and incomparable regions in the objective space.

- X **dominates** Y, $X \prec Y$, if

$$\begin{cases} \forall_{i\in\{1,2,\dots,m\}} : w_i(X) \leq w_i(Y) \\ \exists_{j\in\{1,2,\dots,m\}} : w_j(X) < w_j(Y); \end{cases} \tag{2.12}$$

- X **strictly dominates** Y, $X \prec\prec Y$, If

$$\forall_{i\in\{1,2,\dots,m\}} : w_i(X) < w_i(Y); \tag{2.13}$$

- X is **incomparable** to Y, $X \sim Y$, If

$$X \not\preceq Y \text{ and } Y \not\preceq X. \tag{2.14}$$

Those definitions state that solution X weakly dominate solution Y if X is not worse than Y in all objectives; X dominates Y if X is no worse than Y in all objectives and is strictly better in at least one of those objectives; and if X is strictly better than Y in all objectives then X strictly dominates Y. It is also considered the case where

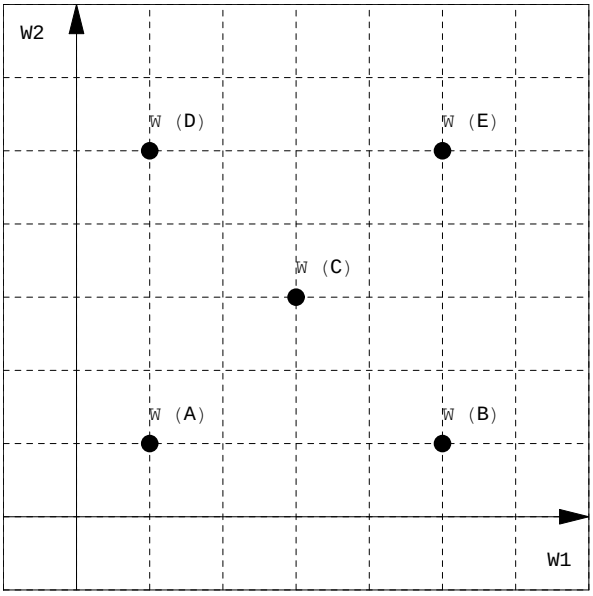


Figure 2.3: Graphical representations of the weights vector of five solutions.

none of the solutions over performs the other, designated as incomparable solutions. Figure 2.2 represents the regions in the objective space that dominates, are dominated, or are incomparable to a solution X.

From the definition it is also obvious that if X strictly dominates Y then X dominates Y, and if X dominates Y then X weakly dominates Y, that is,

$$X \prec\prec Y \Rightarrow X \prec Y \Rightarrow X \preceq Y. \tag{2.15}$$

In Figure 2.3 we can see the dominance relations between five solutions A, B, C, D, and E. In this case $A \prec\prec C$, $A \prec\prec E$, $C \prec\prec E$, $A \prec B$, $A \prec D$, $B \prec E$, and $D \prec E$.

Table 2.2 resumes the properties that the binary dominance relation satisfy (in Appendix A are the recalled the basic binary relations classes). The $\preceq$ relation is a (weak) partial order since it is reflexive, antisymmetric and transitive. Being irreflexive and transitive, the other two relations ($\prec$ and $\prec\prec$) are strict partial orders.

According to the above dominance definitions, it is now possible to set an optimality condition where the solutions of the problem are the solutions that are not dominated by any other feasible solution as follows. A solution X is **Pareto (optimal)** or

	$\preceq$	$\prec$	$\prec\prec$
Reflexive	✓	×	×
Irreflexive	×	✓	✓
Co-reflexive	×	×	×
Symmetric	×	×	×
Antisymmetric	✓	×	×
Transitive	✓	✓	✓
Total	×	×	×
Trichotomous	×	×	×
Observations	Weak partial order	Strict partial order	Strict partial order

Table 2.2: Binary relations satisfied by the dominance relations.

efficente if it is not dominated by any other solutions of $\mathcal{S}$, that is, $\forall_{Y \in \mathcal{S}-\{X\}} : Y \not\prec X$. The set of all Pareto solutions is called **Pareto set** or **efficient set**.

To better understand the relations between the identities of the above definition, Figure 2.4 sketches in dark grey the objective space associated to some problem and identifies the dominated region, the Pareto set and the Pareto front.

As we will see in the next sections, the outcome of an approximation algorithm does not necessarily return the exact Pareto set. Therefore, it is usual to define the resulting set as follows. A set $\mathcal{Q}$ of solutions of some optimization problem is called an **approximation set** if no element of $\mathcal{Q}$ is weakly dominated by any other of the elements of $\mathcal{Q}$. This definition indicates that all dominated solutions can be discarded as approximations to the problem in study.

Furthermore, the concept of **well-distributed efficient solutions** [Hamacher & Ruhe, 1994], can be generalized to the approximation set case as follows. Let $\mathcal{P} = \{X_1, X_2, \ldots, X_N\}$ be a lexicographically ordered approximation set. $\mathcal{P}$ is a **well-distributed approximation set** if the Euclidean distance between two lexicographic

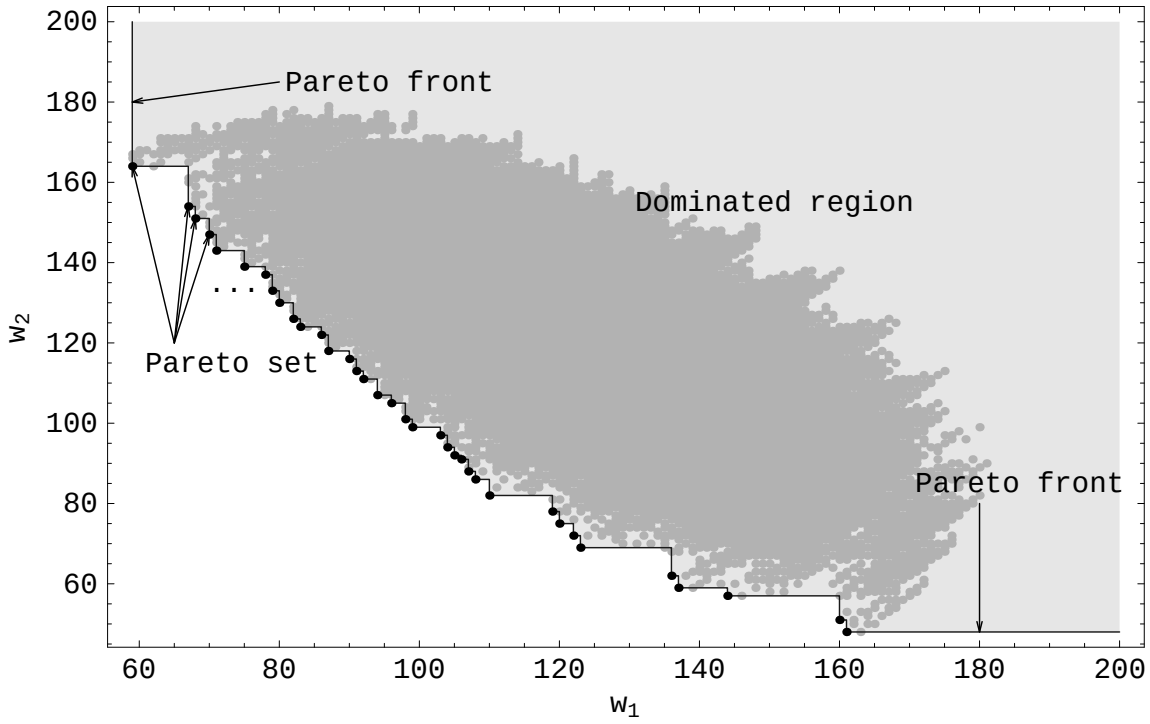


Figure 2.4: Pareto set and dominated region.

consecutive solution in the objective space is smaller then some predefined ϵ value, that is,

$$\max_{i=1,2,\ldots,N-1} \sqrt{\sum_{i=1}^{m} \left(w_i(X_i) - w_i(X_{i+1})\right)^2} < \epsilon \tag{2.16}$$

A well-distributed set is a relevant feature of the quality of the approximation sets, which will be further investigate in Section 2.4.

For the approximations sets we can define four relations [Zitzler *et al.*, 2003]. If $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are two approximations sets, we say that

- $\mathcal{Q}_1$ **strictly dominates** $\mathcal{Q}_2$, $\mathcal{Q}_1 \prec\prec \mathcal{Q}_2$, if every solution in $\mathcal{Q}_2$ is strictly dominated by at least one solution in $\mathcal{Q}_1$, that is,

$$\forall_{Y \in \mathcal{Q}_2} \exists_{X \in \mathcal{Q}_1} : X \prec\prec Y. \tag{2.17}$$

- $\mathcal{Q}_1$ **is better than** $\mathcal{Q}_2$, $\mathcal{Q}_1 \triangleleft \mathcal{Q}_2$, if every solution in $\mathcal{Q}_2$ is weakly dominated by

at least one solution in $\mathcal{Q}_1$ and $\mathcal{Q}_1 \neq \mathcal{Q}_2$, that is,

$$\forall_{Y \in \mathcal{Q}_2} \exists_{X \in \mathcal{Q}_1 \neq \mathcal{Q}_2} : X \preceq Y. \tag{2.18}$$

- $\mathcal{Q}_1$ **dominates** $\mathcal{Q}_2$, $\mathcal{Q}_1 \prec \mathcal{Q}_2$, if every solution in $\mathcal{Q}_2$ is dominated by at least one solution in $\mathcal{Q}_1$, that is,

$$\forall_{Y \in \mathcal{Q}_2} \exists_{X \in \mathcal{Q}_1} : X \prec Y. \tag{2.19}$$

- $\mathcal{Q}_1$ **weakly dominates** $\mathcal{Q}_2$, $\mathcal{Q}_1 \preceq \mathcal{Q}_2$, if every solution in $\mathcal{Q}_2$ is weakly dominated by at least one solution in $\mathcal{Q}_1$, that is,

$$\forall_{Y \in \mathcal{Q}_2} \exists_{X \in \mathcal{Q}_1} : X \preceq Y. \tag{2.20}$$

2.3 Multiple Objective Optimization Methods

2.3.1 Algorithms Complexity

The theory of complexity studies the time and space needed to execute some algorithm (see for example [Ausiello *et al.*, 1999] for more details). The algorithmic complexity of a method depends on the size, n, of the input instance (like the number of nodes or edges of a network) and is classified by a function $f(n)$, upon which the requirements needed to run the algorithm can be estimated. Two parameters are usually considered:

Time complexity The time complexity of an algorithm is used to estimate the number of operations necessary to obtain a solution, like arithmetic operations, comparisons, or memory accesses.

Space complexity The space complexity estimates the amount of memory needed to execute the algorithm, like number of integers, floats, or other types.

Knowing the time and space complexity of two algorithms allows the theoretical comparison of the methods under the same inputed conditions.

Assuming some data input, commonly three types of computational complexity can be estimated: the worst possible case (which is the most used), the best possible case, or the average complexity. The best case usually has not great interest and the average case is difficult to compute due to the existence of probability functions to evaluate in the strict dependency of the data input.

Nevertheless, it is usually considered enough to compute the rate of growth of f as the input size enlarges. For example, if algorithm A performs $3n^2 + n$ operations, it is considered that its growth velocity is of n^2 polynomial order.

To represent the rate of growth of the time or space complexity it is used the following notation:

- $f(n) = \mathcal{O}(g(n))$ if exists c such that $f(n) \leq cg(n)$ for n large enough;
- $f(n) = \Omega(g(n))$ if exists c such that $f(n) \geq cg(n)$ for n large enough;
- $f(n) = \Theta(g(n))$ if $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$.

If $f(n) = \mathcal{O}(g(n))$ then f increases at most at the same rate as g. On the other hand, $f(n) = \Omega(g(n))$ means that f growth at least at the same rate as g. If $f(n) = \Theta(g(n))$ then f as the same rate of growth of g.

The time/space complexity upper bound of a problem is $\mathcal{O}(g(n))$ if exists an algorithm for that problem whose running time/space is $\mathcal{O}(g(n))$. This complexity upper bound provides information about the amount of time/space which is asymptotically sufficient to solve the problem. However, the upper bound does not mean that a precise knowledge of the complexity of the problem is available since more efficient algorithms, with smaller running times/spaces, may exist and we may not be aware of their existence.

On the other hand, the complexity time/space lower bound of a problem is $\Omega(g(n))$ if any algorithm for that problem has at least running time/space $g(n)$. Establishing a lower bound for a problem is a difficult task since it is necessary to prove that this lower bound value holds for all algorithms that solve the problem (known and unknown).

A problem has $\Theta(g(n))$ time/space complexity if its upper bound is $\mathcal{O}(g(n))$ and its lower bound is $\Omega(g(n))$.

The algorithms with complexity $\mathcal{O}(n^k)$ for some k are called **polynomial, efficient or good algorithms** [Jungnickel, 1999]. Problems for which exist efficient algorithms are called **easy problems**. On the other hand, problems for which no polynomial algorithm can exist are called **intractable** or **hard**. In fact, exist a large number of problems called $\mathcal{NP}-$**complete** for which not only no polynomial algorithm is known, but for which is also a good reason to believe that such efficient algorithms cannot exist (see [Garey & Johnson, 1990] for a more thoroughly investigation).

Another form of intractability occurs when the number of the solutions of the problem increase exponentially with the input size of the problem, which is usually represented as $\mathcal{NP} - \sharp$ [Hamacher & Ruhe, 1994].

2.3.2 Classical Methods

Exact Methods

Methods to compute the Pareto set, like the Brute Force methods, exist for the majority of the optimization problems. However, in general they are only capable to manage small instances of the proposed problems, in reasonable time. Beside the Brute Force strategy several others methods have been proposed to explore all possible solutions of the optimization problems. Some examples are the Divide-and-Conquer techniques with variants like the Branch-and-Bound and Branch-and-Cut algorithms [Mitchell, 2002].

For instance, the Branch-and-Bound method is a more efficient way (than the Brute Force method) to guaranty the optimal solution of a problem. The method has two basic operations:

Branching is an intelligent form of covering the admissible region through several admissible subregions. Since this process is recursive in each subregion it can be naturally thought as a tree where the nodes are the constructed regions. This tree is known as the search tree or branch-and-bound tree.

Bounding is a fast way of computing the lower and upper bounds for the optimal solution inside the subregion defined by the branching. If the lower bound of some region is greater than the upper of some other, then that first subregion can be discarded in a process called pruning.

In the bounding phase, when for some subregion the upper and lower bound matches (case in which we have the optimum value) or it is possible to efficiently compute the optimum, that subregion is said to be solved. Ideally the process would stop when all the subregions where either pruned or solved. In practice, it is usual to stop the process after some predetermined time.

This method was able to tackle several problems like the Robust Spanning Tree with Interval Data problem [Montemanni & Gambardella, 2003], the Travelling Salesman Person problem [Hernández-Pérez & Salazar-González, 2004; Zhang, 1993], or the Knapsack problem [Kozanidis & Melachrinoudis, 2004].

Based Single Objective Methods

Due to the intrinsic complexity found on most of the multiple criteria problems, several techniques have been developed to take advantage from the known single objective algorithms. Those methods, where transformations of the multiple objectives problem

into a single objective one are used to establish the procedures, are usually recognized as the classical techniques. Three examples of those techniques are presented next.

Weighted Sum Method Among the classical techniques, the Weighted Sum Method is the widely used. In this case, a set of weighted vectors

$$\Lambda = \left\{ (\lambda_1, \lambda_2, \dots, \lambda_m) : \lambda_i \in \mathbb{R}^+ \wedge \sum_{i=1}^{m} \lambda_i = 1 \right\} \tag{2.21}$$

is initially set. Then, each element $(\lambda_1, \lambda_2, \dots, \lambda_m)$ of Λ is used to obtain a single objective function through the weighted sum of the components of $\mathcal{W}$, defined in formula (2.1). As consequence a new single objective optimization problem is set as

$$\min_{X \in \mathcal{S}} \sum_{i=1}^{m} \lambda_i w_i(X). \tag{2.22}$$

Now, providing the existence of an algorithm for the associated single objective problem, each optimization procedure returns efficient solutions.

To prove that the solutions of (2.22) are efficient solutions of the original multiple objective problem, suppose that X^* is a solution of (2.22), but is not a Pareto solution. Then, there exists $X \in \mathcal{S}$ such that $X \prec X^*$, that is, $w_i(X) \leq w_i(X^*)$ for all $i \in \{1, 2, \dots, m\}$ and $w_j(X) < w_j(X^*)$ for at least one $j \in \{1, 2, \dots, m\}$. Therefore, for vector $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_m)$, with $\lambda_i > 0$, we have $\lambda_i w_i(X) \leq \lambda_i w_i(X^*)$ for all $i \in \{1, 2, \dots, m\}$ and exists $j \in \{1, 2, \dots, m\}$ such that $\lambda_j w_j(X) < \lambda_j w_j(X^*)$ which implies that

$$\sum_{i=1}^{m} \lambda_i w_i(X) < \sum_{i=1}^{m} \lambda_i w_i(X^*). \tag{2.23}$$

This contradicts our hypotheses that X^* is a solution of $\min_{X \in \mathcal{S}} \sum_{i=1}^{m} \lambda_i w_i(X)$. Therefore, X^*, solution of $\min_{X \in \mathcal{S}} \sum_{i=1}^{m} \lambda_i w_i(X)$, must be an efficient solution.

Although this result guarantees efficient solutions, it cannot be used to solve all multiple objective problems due to several facts, like

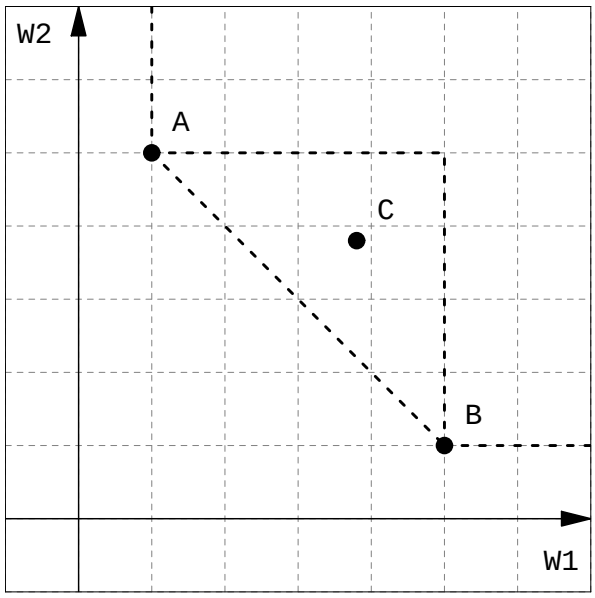


Figure 2.5: Example of non-supported efficient solution.

- The associated single-objective problem might not have an efficient algorithm to solve it [Knowles & Corne, 2001b];
- Different weight vectors need not necessarily lead to distinct solutions, thus becoming a hard task to produce a representative Pareto front;
- Generally, the solutions are not well distributed[Hamacher & Ruhe, 1994]; and
- The Pareto solutions with objectives in a concave region of the Pareto front are not solutions of (2.22) for any weight vector of Λ [Deb, 2001].

The last point states that if $\{A, B, C\}$ are three solutions in the objective space, as sketched in Figure 2.5, then C is not a solution of the Weighted Sum problem.

In fact, as pointed out by Deb [2001] for the bi-objective case, the objective vector for any solution of (2.22), is an element of the convex hull of the objective space, which suggests the next definitions. A solution is called **supported efficient solution** or **extreme efficient solution** if it is the solution of (2.22), for some vector $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m) \in \Lambda$. The remaining solutions are called **non-supported efficient**

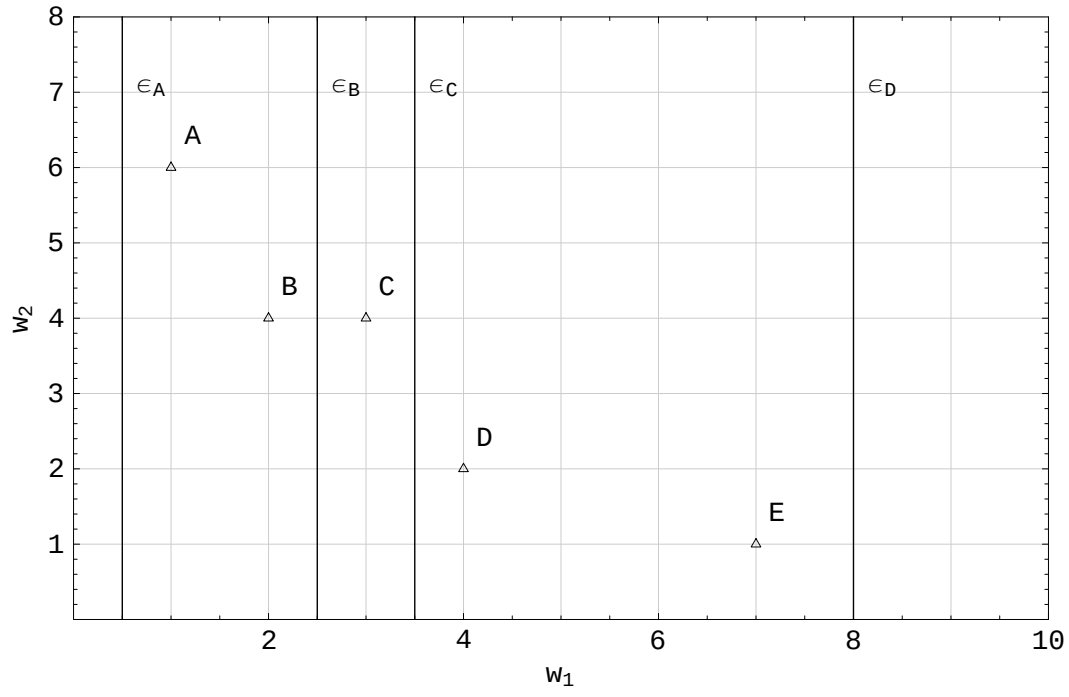


Figure 2.6: ϵ-constraint method.

solutions.

ϵ–**Constraint Method** In this case, one of the objectives is minimized as a single objective function and the remaining objectives are restricted to being smaller than a parameter value:

$$\min_{X \in \mathcal{S}_\mu} w_\mu(X), \tag{2.24}$$

where

$$\mathcal{S}_\mu = \{X \in \mathcal{S} : w_i(X) \leq \epsilon_i, i \neq \mu\} \tag{2.25}$$

Consider the example presented in Figure 2.6 where A, B, D and E are Pareto solutions. In this case, if we apply the restriction $w_1(X) \leq \epsilon_B$ and minimize the second objective, then the solution of (2.24) is B, element of the Pareto front.

In [Miettinen, 1999] is proved that if the solution of (2.24) is unique then that solution is also an optimum of the original multiple objective problem. On the other

hand, if the solution of (2.24) is not unique then it is possible that the returned solution is not on the Pareto front. Considering again the example in Figure 2.6, when the restriction used is ϵ_C then both B and C are solutions of (2.24) although C is not Pareto optimum.

The ϵ–Constraint method has the advantage of being applicable to convex and non convex fronts in undifferentiated mode and the fact that different values of the ϵ_i can produce different solutions of the Pareto front.

On the other hand, as exampled, the returned solution can be non-efficient. Furthermore, difficulties can arise in the definition of the values of the ϵ_i. If one of the ϵ_i is smaller than the minimum of w_i, that is,

$$\epsilon_i < \min_{X \in \mathcal{S}} w_i(X), \tag{2.26}$$

for some $i \in \{1, 2, \ldots, \mu - 1, \mu + 1, \ldots, m\}$ then $\mathcal{S}_\mu$ is an empty set (like the use of ϵ_A in the example in Figure 2.6). On the other hand, if the ϵ_i are greater then the maximum of the i components of the efficient set ($\mathcal{P}^*$), that is,

$$\epsilon_i > \max_{X \in \mathcal{P}^*} w_i(X) \tag{2.27}$$

for all $i \in \{1, 2, \ldots, \mu - 1, \mu + 1, \ldots, m\}$, then there is no restriction on $\mathcal{S}$ (as the use of ϵ_D in the example). Therefore, the values of ϵ_i must be carefully selected and should be incremented in small steps within the interval defined by their respective weights. For a large number of objectives this becomes difficult and time consuming.

Weighted Metric Method The weighted metric method, as the name suggests, uses weighted metrics to transform a multiple objective problem into a single objective one. For example, the single objective problem can be defined as

$$\min_{X \in \mathcal{S}} l_p(X), \tag{2.28}$$

where

$$l_p(X) = \sqrt[p]{\sum_{i=1}^{m} \lambda_i \left| w_i(X) - w_i(X^*) \right|^p} \tag{2.29}$$

is the weighted l_p norm, $(\lambda_1, \lambda_2, \ldots, \lambda_m)$ satisfies (2.21) and X^* is some ideal solution.

If $p = 1$ then the single objective problem is defined as

$$\min_{X \in \mathcal{S}} \sum_{i=1}^{m} \lambda_i \left| w_i(X) - w_i(X^*) \right| \tag{2.30}$$

which is equivalent to the Weighted Sum approach [Deb, 2001]. When $p = 2$ the returned solution is the solution that minimize the weighted Euclidean distance to the ideal point. The case where $p = \infty$ is called the weighted Tchebycheff method, formulated as

$$\min_{X \in \mathcal{S}} l_\infty(X), \tag{2.31}$$

where

$$l_\infty(X) = \max_{i=1,2,\ldots,m} \lambda_i \left| w_i(X) - w_i(X^*) \right|. \tag{2.32}$$

In [Miettinen, 1999] it is proved that any efficient solution is a solution of the weighted Tchebycheff method for some weighting vector.

However, the method has some weakness, like

- The need to know the ideal point, which implies the computation of the minimum values for each objective.
- When the associated single objective problems has not an efficient algorithm to solve it, it becomes a hard task to compute the ideal point and the efficient solutions.
- In the continuous case, as p increases, l_p becomes non-differentiable which implies that the single objective gradient-based methods cannot be used.

2.3.3 Heuristics and Meta-Heuristics

The previous section presented some of the classical optimization methods. In general, those methods transform the multiple objective problem into a single objective one. Nevertheless, those methods have some important weaknesses. Perhaps the most important pitfall is the fact that, in many cases, the resulting single objective minimization problem has not an efficient algorithm to solve it. Other difficulties may occur in the Weighted Sum method by the impossibility of returning any of the efficient solutions in a concave region of the Pareto front. Although the ϵ-Constraint method can compute those solutions, this method has not the assurance of returning efficient solutions. Some other problems associated to some of the described methods are the need of an *a priori* known ideal point and the fact that different optimization parameter like the λ weighting vector or the ϵ_i values do not necessarily produce different solutions.

In the last decades several alternatives to the classical optimization methods have been proposed. Most of these alternatives where first introduced to solve the single objective optimization problems, but, due to their characteristics, were more or less straightforwardly adapted to the multiple objective case. These methods abstractly describe an optimization concept characterized by a set of common optimization steps to be performed independently of the problem, making them generally applicable and flexible. They are recognized as approximation methods or meta-heuristics.

Most of those meta-heuristics require only a set of feasible solutions, a weight function, a neighbourhood function and an efficient method to explore the neighbourhood.

Usually, the meta-heuristics are distinguished in two main classes characterized by

- Maintaining a single solutions that is consecutively improved using some neighbourhood transition rule (allowing in some cases local worsens), as the Tabu Search or the Simulated Annealing algorithms.

- Being population based with some sort of information exchange between the elements, which permits the simultaneous explorations of different regions of the search space, as the Genetic Algorithms or the Ant Colony Optimization Algorithms.

A large number of these methods mimic natural processes and are, generally, recognized as Evolutionary Computation. One of the first studies in Evolutionary Computation, named as Evolutionary Strategies, was made in the 1960's decade to optimize the search for the optimal shapes of bodies in a flow and presented in [Rechenberg, 1973; Schwefel, 1977]. Those ideas were latter developed by others researchers conducting to other evolutionary methods, like Evolutionary Programming, Genetic Algorithm, Genetic Programming, or Swarm Intelligence.

The remaining of this section outlines some of those meta-heuristics that represent the actual trend of the field. With particular relevance it will be made an analysis of the Ant Colony Optimization algorithm.

Swarm Intelligence Algorithms

Swarms The swarms are complex adaptive systems that display emergent behaviour and are the base idea of the Swarm Intelligence algorithms.

These algorithms are established over the conduct of animals that are not considered to be intelligent, like insects. Besides, the myth of the Honey Bee Queen, or the Ant Queen, the last one well personified in the Antz film, has not any relationship to the real swarm conduct, since neither the queen nor any of swarm members has a command role in the swarm dynamics. However, the group behaviour of some of those social creatures, acting as a swarm, reveals aptitude to solve very complex tasks.

One of the first swarm like organism to be studied, from a mathematical point of view, was the slime mold [Johnson, 2001]. The slime mold oscillations between being a

single creature and a swarm are made according to its environment. Those changes are based not in a leader instruction but in a collective phenomenon that is triggered by a substance called cyclic AMP. That substance sweeps the entire community, as each isolated cell relays the signal to its neighbours.

This conduct is now one of the classics studies in bottom-up behaviour. In this problem solving concept answers are produced based on a multitude of elements, rather than in a chain of command supported by a leader. More detailed overviews of those studies can be seen in [Johnson, 2001; Resnick, 1999; Tarasewich & McMullen, 2002].

Ant Colony Optimization

Real Colonies of Ants Ants are social insects that work together for the welfare of the colony in the detriment of the individual. Each one work up to a common goal, making decisions as a response to the surrounding environment.

The communication between ants is based in chemical signals. These chemical signals are called pheromones and are used in a variety of situations like alarm of some threat, recognition of other colony elements, or as a recruitment signal for some activity. The pheromones usually are dropped in trails and depending on their purpose different types of chemical are left by each of the species. Nevertheless, the recruitment objective is the same in both cases [Johnson, 2001].

Relatively to the foraging process, it can be described as follows. First, the scouts ants leave the nest in search for supplies. Each ant does a random search until it finds food, feeds itself and go back to the nest in a direct way (somehow the ants keep a record of their position which allows them to return through a straightforward path). In the way back a pheromone trail is left. When it reaches the nest the scout signals the others about the finding, which will follow the resource/nest trail that was left. As they follow the trail, those other ants reinforce it, which will guide more elements

to the harvest. This will keep ongoing until the food finish. Then the ants stop following the trail which will evaporate. This behaviour allows the exploitation of the founded resource. Figure 2.7 sketches a simulation of the foraging process obtained with StarLogo software [Resnick, 1996, 1999, 2004].

The Swarm Intelligence algorithms mimic those swarm group actions. In general, the Swarm Intelligence algorithms employ a set of unfussy agents that react to environmental signals to solve the proposed problems, using changes introduced by other agents, and acting locally to produce complex global behaviour. Furthermore, the use of multiple agents has the advantage of simultaneously searching for solutions in multiple places with distributed control, which produces robustness and flexibility. This characteristic also allows possible parallelizations of the process in a very straight and efficient way.

ACO Algorithm The Ant Colony Optimization (ACO) algorithms are the more spread Swarm Intelligence algorithms [Botee & Bonabeau, 1998; Dorigo & Stutzle, 2004; Dorigo *et al.*, 1999; Middendorf *et al.*, 2002]. Introduced by Marco Dorigo, the Ant Colony Optimization is one of the most recent meta-heuristics that, as the name suggests, mimics the pillagers behaviour of the ants' colonies [Dorigo *et al.*, 1996].

As in the natural process, the Ant Colony Optimization method is based in a set of artificial agents that communicate using simulated trails of pheromones. Those trails reflect the gathered experience of the agents that have already solved the problem and favour the creation of new solutions. The method comprises a set of iterations where collections of solutions are obtained. At the end of each iteration, the pheromone trails are updated considering the known solutions as well as a certain pheromone evaporation.

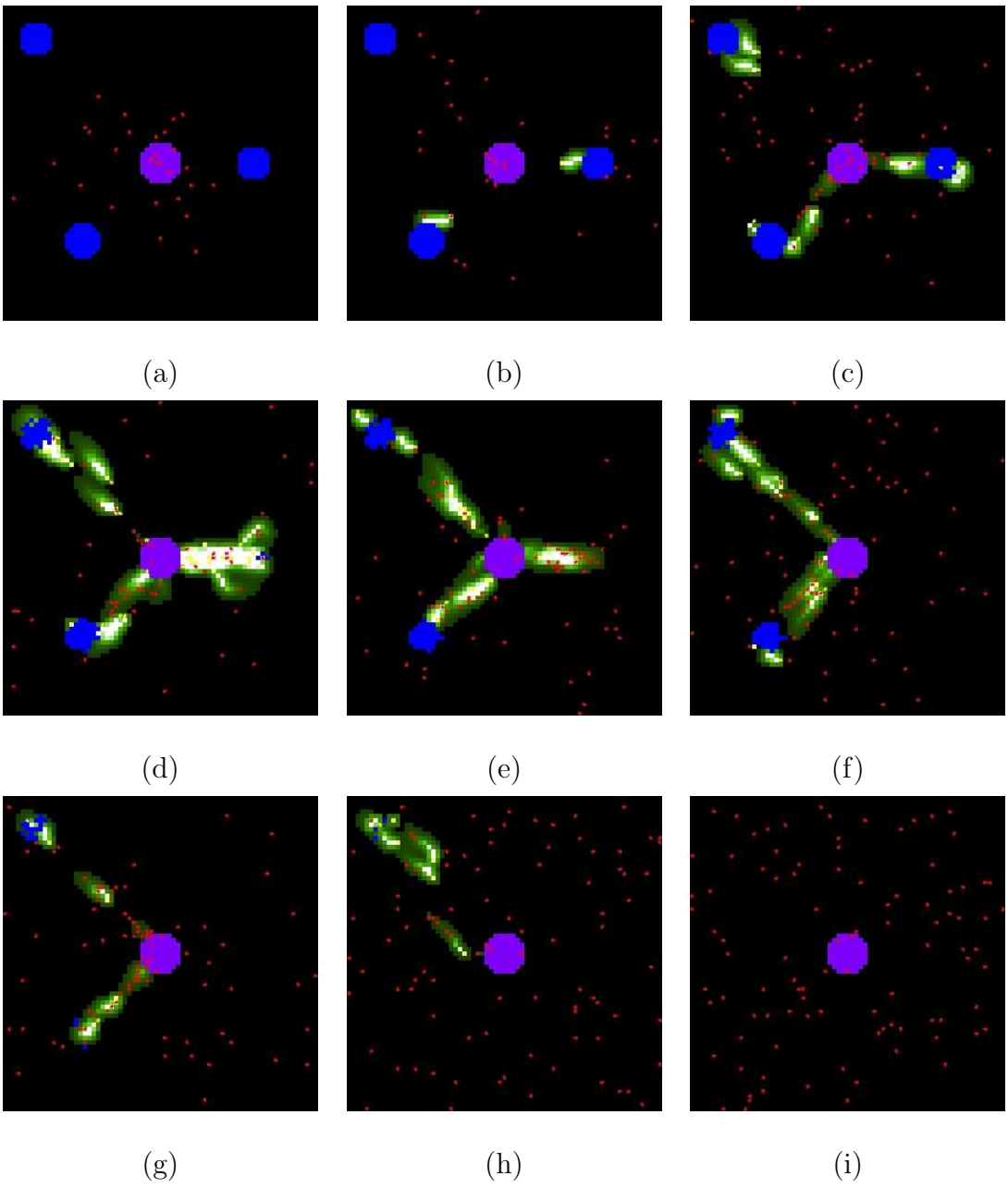


Figure 2.7: StarLogo simulation of the foraging behaviour of an ant colony: (a) scouts start a random search; (b) founded food in the return to the nest they leave a pheromone trail; (c) all sources of food where found; (d)-(e) one of the source was completely harvested and the trail starts to disappear until it completely evaporates; (g)-(i) the other food sources are also finish; The ants start a new random search.

Algorithm 1 ACO - Ant Colony Optimization Algorithm.

```
1: Initialize the pheromone trail.
2: while stopping criterion is not met do
3:     for all ants do
4:         Construct a new solution using the current pheromone trail.
5:         Evaluate the solutions constructed.
6:     end for
7:     Update the pheromone trail.
8: end while
```

Other Swarm Intelligence Algorithms Although the Ant Colony Optimization algorithms are the most well-known in the Swarm Intelligence field, there are other methods based in the same swarm paradigm.

For example, the Particle Swarm Optimization mimics the social models observed in the bird flocking or the fishing schools [Eberhart & Kennedy, 1995a, b]. Each particle of the swarm keeps information about its position and velocity in the search space, its best achieved solution and the global best. Depending on the implementation, the information is either transmitted from each particle to its neighbours or to all of the particles in the swarm. Using the available information and movements that include certain quantities of randomness (usually called craziness, to favour a larger exploration of the search space), the swarm of particles are guided toward the most promising regions of the search space.

Another example is the use of the honey bees behaviour as a metaphor to the optimization of Internet servers [Nakrani & Tovey, 2004; Tovey, 2004].

Swarm Intelligence in Multiple Objective Optimization The application of the Swarm Intelligence algorithm to the multiple objective optimization problems has been developed within several different conceptual approaches. For example, Ant Colony types of algorithms are distinguishable by using [García-Martínez *et al.*, 2004a]

- Single/multiple colonies;
- Single/multiple pheromone matrices;
- Elitist and non-elitist pheromone updating rules; or
- Updating rules that allow ants to update pheromone trails of other colonies.

The multiple objective Ant Colony based algorithms have been applied in some problems, as the Vehicle Routing Problems with Time Windows [Gambardella *et al.*, 1999], the design of water distribution irrigation networks [Mariano-Romero & Morales-Manzanares, 1999], the Travelling Salesman Person problem [Cardoso *et al.*, 2003b; García-Martínez *et al.*, 2004a], the portfolio selection [Doerner *et al.*, 2001], the Minimum Spanning Trees problem [Cardoso *et al.*, 2004], or the optimization of flight routes to avoid hazardous weather [Alam *et al.*, 2006].

In [Ray & Liew, 2002] a Particle Swarm Optimization method is used to approximate the solution of several multiple objective problems including unconstrained and constrained cases with continuous, discrete, or mixed variables. Another example of applications is made in the optimization of the Minimax problem [Laskari *et al.*, 2002].

Genetic Algorithms

First introduced by John Holland in the 1970's, the Genetic Algorithms mimic the Darwinian natural evolutionary principles [Holland, 1975]. This meta-heuristic is perhaps the most used between the evolutionary methods, with proofs of successful applications in many academic and practical problems.

In fact, several books are (or have chapters) concerned to the Genetic Algorithms subject [Aarts & Lenstra, 1997; Parmee, 2001; Sait & Youseff, 1997; Vose, 1999]. Simultaneously, numerous thesis and other works that use and develop the method as a

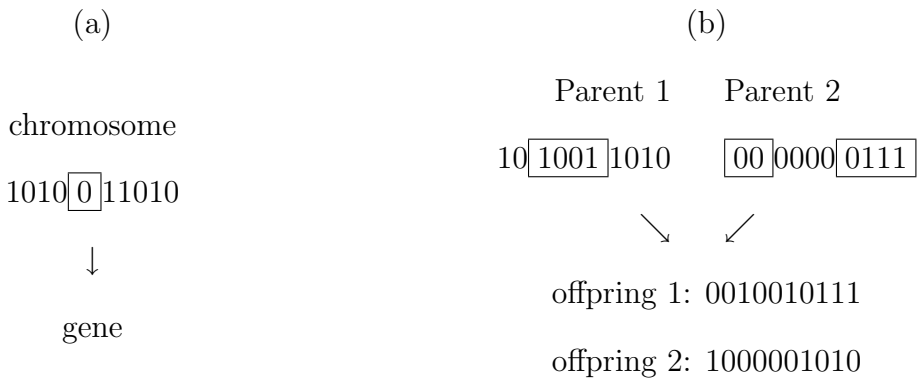


Figure 2.8: Genetic Algorithm (a) Example of binary string codification; (b) Double cut point crossover.

framework were presented, as in [Jesus, 2000; Sinclair, 2001] for single objective problems or as in [Coello-Coello, 1996; Fonseca, 1995; Jaszkiewicz, 2001; Knowles, 2002; Murata, 1997; van Veldhuizen, 1999; Zitzler, 1999] for the multiple objective cases. Surveys and comparisons from some of the Genetic Algorithm variants are made, for example, in [Coello-Coello, 2000; Ehrgott & Dandibleux, 2000; Zitzler *et al.*, 2000].

The Genetic Algorithm's procedure is based in a set of iterations referred as generations. In each generation exists a population or set of solutions. Those solutions are also usually called chromosomes or individuals and can be codified in different ways, Figure 2.8(a).

The first population, P_0, is the population from which the method will hopefully evolve towards the optimum. The following generations, P_i, are breed from the previous one, P_{i-1}, by the application of the evolution process that is based in the competition between the individuals of the population for mating. Each of the individuals has associated a fitness value which represents is quality as a solution. The most apt individuals have more chances to mate and produce offsprings, perpetuating its fitness. To implement the method is common to use three basic operators:

Selection – Used to probabilistically choose the individuals that will perform the

Algorithm 2 Genetic Algorithms (GA).

output: Approximation to the solution of the problem

```
1: function GA()
2:     Initialize population
3:     Evaluate the population
4:     while stopping criteria not verified do
5:         Choose parents
6:         Perform crossover and mutations
7:         Perform fitness based selection
8:         Evaluate the population
9:     end while
10:    return Population
11: end function
```

mating.

Crossover – With two selected individuals, called parents, are generated offsprings that inherit characteristics from both of them, Figure 2.8(b).

Mutation – Does random changes in the offspring in order to possibly introduce new characteristics to the population avoiding stagnation.

A high level description of the process is made in Algorithm 2.

Tabu Search

The Tabu Search method was introduced by Fred Glover as a general meta-heuristic based on a dynamic-neighbourhood search to solve combinatorial problems [Glover & Laguna, 1997].

A neighbourhood optimization method should maintain not only the information about the present solution but also about the search process. Therefore, the use of memories to keep track of the information about the last iterations is essential. This information is used to guide the search procedure in the transition from one solution

to another, by means of restricting the current solution neighbourhood.

If a non-memory based procedure is used, the method would just choose the next solution applying formula $\arg\min_{R \in N(S)} f(R)$, where f is the objective function, S is the current solution, and $N(S)$ is the set of neighbourhood solution of S. This methodology leads to a steepest descent method, which is very susceptible to get trapped in local minimums. To avoid this risk, the method must accept solution that are not better then the present one, which *per se* creates another risk: getting trapped in a cycle between a set of solutions. The Tabu Search method tries to avoid this last menace by forbidding the visit of the last n solutions. This can be made by the use of a dynamic-neighbourhood approach, so that in iteration k the feasible neighbours of S would be a set $N_k(S)$ that does not include the last n visited solution, which are kept in some tabu list T, $N_k(S) = N(S) - T$.

Some application in the literature are the 0-1 Multiple Objective problems [Alves & Clímaco, 2000], the Quadratic Assignment problem, or the Bandwidth Packing problem [Sait & Youseff, 1997]

Algorithm 3 presents the Tabu Search fundamental procedure.

Simulated Annealing

As the name suggests, the Simulated Annealing algorithm is based on the process in which metal is heated and cooled into a minimum energy crystalline structure (the annealing process). The algorithm was first proposed by Kirkpatrick *et al.* [1983] to several combinatorial problems. Basically, the method is a relaxation of the classical greedy local search algorithm where a solution is replaced when it is found a new improved one.

A list of results is presented in [Aarts & Lenstra, 1997], which proves that, under mild conditions, the Simulated Annealing algorithm converge in probability to the

Algorithm 3 Tabu Search (TS).

input: S_0 ▷ *initial solution*
output: S^* ▷ *best approximation achieved*

```
1: function TS(S_0)
2:     S* ← S_0
3:     Set the tabu list to the empty set
4:     k ← 0
5:     while stopping criteria not verified do
6:         k ← k + 1
7:         Compute the set of feasible neighbours, V*
8:         Choose the best solution from V*, S_k ← arg min_{R∈V*} f(R)
9:         Update the tabu list
10:        if f(S*) > f(S_k) then
11:            S* ← S_k
12:        end if
13:    end while
14:    return S*
15: end function
```

set of optimal solutions, that is, the algorithm asymptotically determines an optimal solution with probability 1.

Some application in the literature are the Establishing Positioning Networks [Saleh & Dare, 2002], the 0-1 Multiple Objective problems [Alves & Clímaco, 2000], or the Capacitated Minimum Spanning Tree problem [Torres-Jimenez *et al.*, 1999]

In Algorithm 4 it is presented a high level description of this meta-heuristic.

2.4 Performance Metrics

The studies made in the meta-heuristics field represent a great effort of the scientific community to develop new techniques capable of solving some of the most demanding optimization problems, as we have seen through the non exhaustive list of algorithms exampled in the previous section.

Algorithm 4 Simulated Annealing (SA).

input: S_0, T_0 ▷ *Initial solution and temperature*

output:

function SA(S_0, T_0)
  $S \leftarrow S_0$
  $S^* \leftarrow S$
  $T \leftarrow T_0$
  while Stopping criteria not verified **do**
    Compute a solution R in the neighborhood of S
    if R improves S **then**
      $p \leftarrow 1$
    else
      $p \leftarrow e^{\frac{w(S)-w(R)}{T}}$
    end if
    Set p_0 as a random value in $[0, 1]$
    if $p < p_0$ **then**
      $S \leftarrow R$
    end if
    Update T
    if R improves S^* **then**
      $S^* \leftarrow R$
    end if
  end while
  return S^*
end function

It is also logical that any optimization meta-heuristic has as objective to determine good approximations to the problem solution, whenever it is impossible to achieve the exact solution. However, the confirmation of the quality of those solutions and the report of the worthiness of the results are almost always a difficult task [Barr *et al.*, 1995].

Given two approximations sets $\mathcal{P}$ and $\mathcal{Q}$ several questions can arise [Knowles *et al.*, 2005]:

- Is $\mathcal{P}$ better than $\mathcal{Q}$?
- If it is, by how much?
- If it is not, in what aspects?

In general, the answers to each of these questions are difficult. The proposed classifications usually assess in one of the following categories:

- Accuracy of the approximation set;
- Computational effort – CPU time, memory, number of evaluations, number of iterations;
- Robustness – how does the method behave when small variations on the data input are introduced; or
- Reliability – how does the method works when applied to problems with different classes of difficulties.

In the single objective case it is enough to compare the obtained solution with the optimum solution or the best known approximation, which makes comparing the accuracy of a solution relatively simple. However, comparing the performance of different approximations, obtained with different algorithms or possibly the same but with different parameters, can be much more difficult. This difficulty arises from other multiple factors to consider in the classification attempt as the computational effort, the reliability, or the robustness.

In the multiple objective case the number of dimensions to evaluate is even greater, as a result of the nature of the problem. There is a basic factor that contribute to the general increase of these difficulties: the algorithms return approximation sets with each element represented by a scalar vector. Comparing two single distinct solutions is

rather simple, although it can be inconclusive, depending on their dominance relation (see the definitions of dominance and incomparability in page 24).

The problem is much more complex when we think about approximation sets. If we are "lucky" all elements in one set are dominated by at least one element in the other set and we can easily classify them. On the other hand, if the opposite occurs, we would like to estimate which one is the best approximation.

The earliest documents in the field simply sketched the fronts (up to three objectives) and tried to empirically take conclusions about the global performance of the sets. This approach rapidly become insufficient due to factor like the number of objectives (greater than three), the number of solutions of the front, the number of fronts to be compared, or the inconclusive observation of the sketches where the fronts alternately were better than the others. Those pitfalls conducted to the development of quantitative metrics more simple to compare, than the graphical plots.

Until now there are three approaches to compare multiple solutions obtained with different algorithms, namely the:

- Empirical attainment functions – indicate the probability of achieving a predefined objective;
- Dominance-compliant quality indicators – estimate the fitness of the solutions; and
- Comparison indicators based on utility functions – use utility functions, possibly provided by the decision maker, to compute comparison values.

In fact, the attainment function is computed as

$$\alpha_{\mathcal{P}}(Y) = P\left(\bigcup_{X_i \in \mathcal{P}} [X_i \preceq Y]\right) \tag{2.33}$$

where $\mathcal{P} = \{X_1, X_2, \ldots, X_n\}$ is an approximation set and Y is a goal solution [Fonseca *et al.*, 2001]. Therefore, $\alpha_{\mathcal{P}}(Y)$ is the probability of the goal Y to be attained by at least one of the solutions in $\mathcal{P}$. The empirical attainment function can then be computed as

$$\alpha_{\aleph}(Y) = \frac{1}{n} \sum_{\mathcal{P}_i \in \aleph} \alpha_{\mathcal{P}_i}(Y) \tag{2.34}$$

where $\aleph = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\}$ is a set of n approximation sets obtained by n independent runs from the optimizer.

The dominance-compliant m-ary quality indicator I is a function

$$\begin{array}{rccc} I: & \Omega^n & \rightarrow & \mathbb{R} \\ & (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n) & \mapsto & I(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n) \end{array} \tag{2.35}$$

where Ω is the set off all approximation sets and $I(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n)$ is a real value that specifies the quality of the approximation sets $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n)$ [Zitzler *et al.*, 2003]. The most commonly used quality indicators are the unary and binary. The unary indicators are used to measure the proximity to the Pareto set (or some reference set), the spread, or the uniform distribution of the approximation. On the other hand, the binary indicators compare two approximation sets returning a value that should indicate which one is preferable to the other.

In [Zitzler *et al.*, 2002b, 2003] it is shown that there is no combination of unary quality measures that, in general, can indicate that an approximation $\mathcal{P}$ is better than an approximation $\mathcal{Q}$. Nevertheless, these quality indicators are widely used as combination of several of them that, as we will see in the next sections, is based on the distinct characteristics of each one of them.

As the name suggest, the comparison indicators based in utility functions employ usefulness functions to compute a value that indicates the worth of the approximation set through that function. Usually, the utility function is parametric correspondence and an overall value for the indicator can be computed by combining the results when those parameters change in some set.

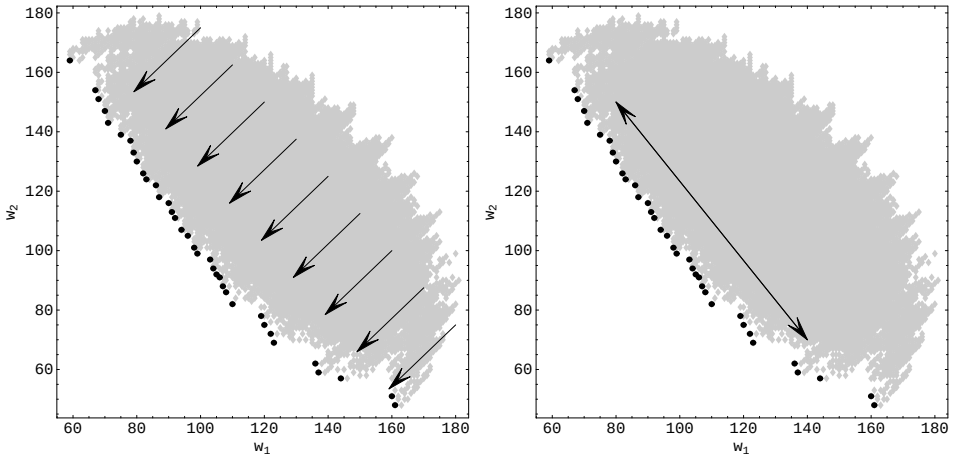


Figure 2.9: Orthogonal goals.

Restricting our analysis to the quality of the approximations sets, they should satisfy some desirable aspects[Bui *et al.*, 2001; Deb, 2001]:

- Be as near as possible to the real Pareto front;
- Have spread solutions all along the Pareto front; and
- Be uniformly distributed.

The objectives of being near and simultaneously be spread all along the Pareto front can be considered orthogonal, as sketched in Figure 2.9. In fact, the first goal requires a depth search in the directions of the Pareto front while the second requires a sweeping search through all the Pareto Front.

For short, the use of quality indicators allows to estimate the worth of one or more approximations sets. Whenever possible they should be compared to the (real) Pareto front or to a reference set. The quality of the approximation set is based in the two orthogonal objectives: minimize the distance to the Pareto front and maximize diversity of the solutions (extension and distribution).

Some of the most used quality indicators for sets of non-dominated solutions are presented next.

2.4.1 Closeness to the Reference Set

Error Ratio - ER

The **Error Ratio** metric [Bui *et al.*, 2001; Deb, 2001; van Veldhuizen & Lamont, 1999] is defined as

$$\begin{array}{rccc} ER: & \Omega^2 & \rightarrow & \mathbb{R}^+ \\ & (\mathcal{P}^*, \mathcal{Q}) & \mapsto & \frac{\sum_{q \in \mathcal{Q}} e(q)}{|\mathcal{Q}|} \end{array} \tag{2.36}$$

where Ω is the set off all approximation sets, $\mathcal{Q}$ is an approximation set, $\mathcal{P}^*$ is the reference set, and

$$e(q) = \begin{cases} 1 & if \quad q_i \notin P \\ 0 & if \quad q_i \in P \end{cases} . \tag{2.37}$$

The Error Ratio metric is one of the simplest metric since it only returns the ratio of the elements in the approximation set $\mathcal{Q}$ that are not in the reference set $\mathcal{P}^*$.

If ER returns a small value then $\mathcal{Q}$ is a good approximation since most of its values are in the Pareto set $\mathcal{P}^*$. In fact, ER ranges from 0 to 1 where $ER = 0$ means that all elements of $\mathcal{Q}$ are elements of $\mathcal{P}^*$, and $ER = 1$ means that no element of $\mathcal{Q}$ is an element of $\mathcal{P}^*$. For the approximation and reference sets presented in Figure 2.10 the error ratio is equal to $ER = \frac{4}{5} = 0.8$.

Nevertheless, there are several disadvantages in the use of this metric, namely

- $\mathcal{P}^*$ must be the exact Pareto front (with the known difficulties of achieving it) since that if $\mathcal{P}^*$ is not complete some optimal solutions of $\mathcal{Q}$ can be forgotten, like the solution e in Figure 2.10;

- $\mathcal{Q}$ can have solutions very near to $\mathcal{P}^*$ but that is not measured, like the solutions a, c, d in Figure 2.10 where $\mathcal{P} = \{A, B, C, D\} = \{(1, 6), (2, 4), (4, 2), (7, 1)\}$ and $\mathcal{Q} = \{a, b, c, d, e\} = \{(1.2, 6.2), (2, 4), (4.2, 2.2), (7.2, 1.2), (8, 0.5)\}$.

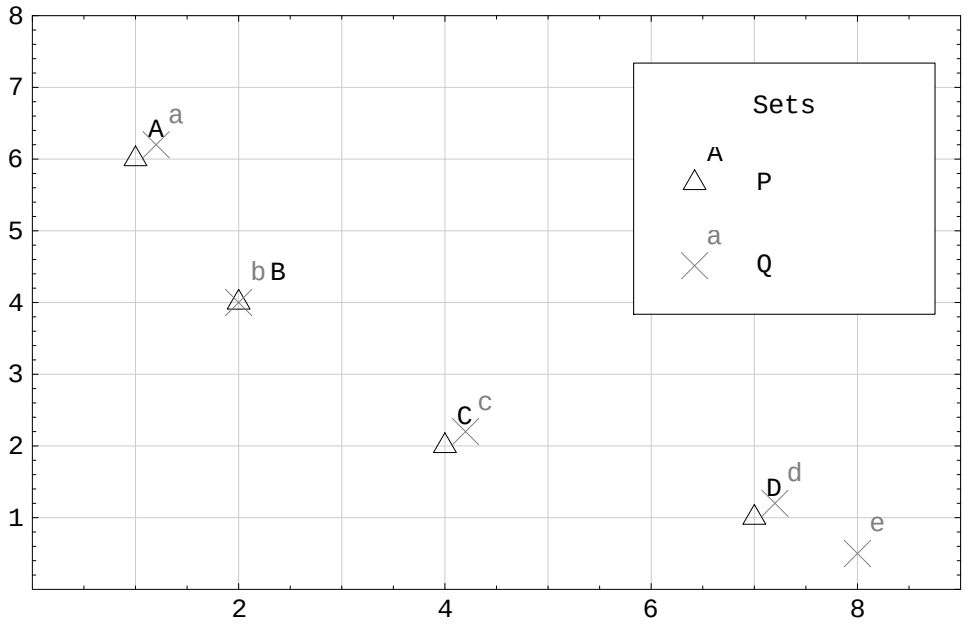


Figure 2.10: Error Ratio example

If the problem allows the use of methods like the Weighted Sum, this metric returns $ER = 0$, even if $\mathcal{Q}$ has one single solution. One of the advantages is its low computational cost, compared with the others.

Set Coverage Metric

The **Set Coverage** metric [Deb, 2001; Knowles, 2002; Zitzler & Thiele, 1999; Zitzler *et al.*, 2000] is computed by

$$\begin{array}{rccl} C: & \Omega^2 & \rightarrow & [0,1] \\ & (\mathcal{P},\mathcal{Q}) & \mapsto & \frac{|\{q\in\mathcal{Q}:(\exists p\in\mathcal{P}:p\preceq q\})|}{|\mathcal{Q}|}, \end{array} \tag{2.38}$$

where Ω is the set off all approximation sets, and $\mathcal{P}$ and $\mathcal{Q}$ are two approximation sets. This metric calculates the proportion of elements in $\mathcal{Q}$ that are weakly dominated by at least one element in $\mathcal{P}$.

If $C(\mathcal{P},\mathcal{Q}) = 0$ then none of the elements of $\mathcal{Q}$ is weakly dominated. On the other hand, if $C(\mathcal{P},\mathcal{Q}) = 1$ then all elements of $\mathcal{Q}$ are weakly dominated by at least one of

the elements in $\mathcal{P}$. However, this metric cannot determine how much an approximation outperforms another if one of the sets completely dominates the other. Since $C(\mathcal{P}, \mathcal{Q}) + C(\mathcal{Q}, \mathcal{P})$ is not necessarily equal to 1, both of the indicators should be considered.

Generational Distance and Maximum Pareto Front Error

The **Generational Distance** [van Veldhuizen, 1999; van Veldhuizen & Lamont, 1999] for $\mathcal{Q}$ is defined as

$$\begin{array}{rccc} GD: & \Omega^2 & \rightarrow & \mathbb{R}_0^+ \\ & (\mathcal{P}^*, \mathcal{Q}) & \mapsto & \frac{\sqrt{\sum_{q \in \mathcal{Q}} d(q)^2}}{|\mathcal{Q}|} \end{array}, \tag{2.39}$$

where $\mathcal{P}^*$ is the Pareto (or a reference) set, $\mathcal{Q}$ is an approximation set, and

$$d(q) = \min_{p \in \mathcal{P}^*} \|\mathcal{W}(p) - \mathcal{W}(q)\| \tag{2.40}$$

is a minimal distance between solutions of $\mathcal{P}^*$ and $\mathcal{Q}$ (in the objective space).

Similarly, the **Maximum Pareto Front Error** [van Veldhuizen, 1999; van Veldhuizen & Lamont, 1999] for $\mathcal{Q}$ is defined as

$$\begin{array}{rccc} MFE: & \Omega^2 & \rightarrow & \mathbb{R}_0^+ \\ & (\mathcal{P}^*, \mathcal{Q}) & \mapsto & \max_{q \in \mathcal{Q}} d(q), \end{array} \tag{2.41}$$

The Generational Distance calculates the average distance of the elements of an approximation set, $\mathcal{Q}$, to the Pareto/reference set, $\mathcal{P}^*$, and the Maximum Pareto Front Error evaluates the maximum distance between the elements of the reference and the approximation set.

Lower values of $GD(\mathcal{P}^*, \mathcal{Q})$ or $MFE(\mathcal{P}^*, \mathcal{Q})$ indicate that the elements of $\mathcal{Q}$ are close to $\mathcal{P}^*$. However, fluctuations of the distances values can induce incorrect ideas about the quality of the approximations. The example of Figure 2.11 has represented

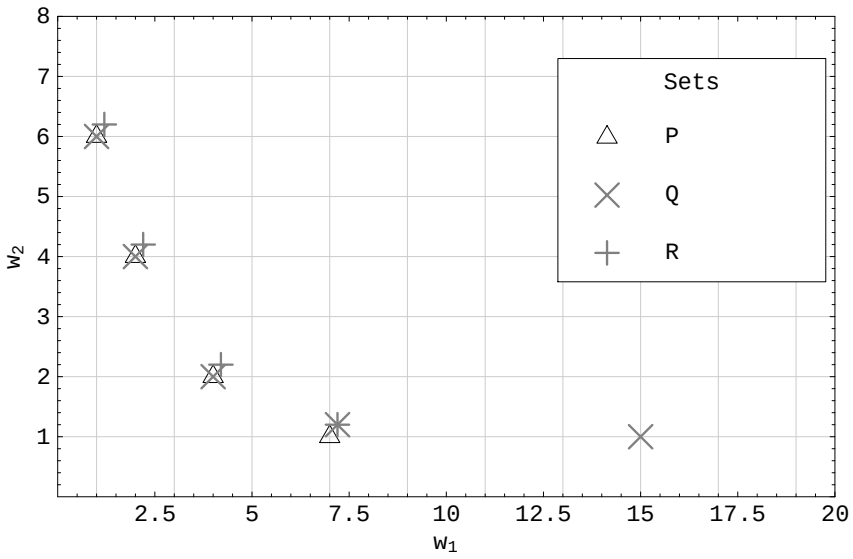


Figure 2.11: Generational Distance and Maximum Pareto Front Error example.

three sets:

$$\begin{aligned}\mathcal{P} &= \{(1,6),(2,4),(4,2),(7,1)\},\\ \mathcal{Q} &= \{(1,6),(2,4),(4,2),(7.2,1.2),(15,1)\}, \text{ and}\\ \mathcal{R} &= \{(1.2,6.2),(2.2,4.2),(4.2,2.2),(7.2,1.2)\}.\end{aligned}$$

In this case, the values of the Generational Distance and the Maximum Pareto Front Error are $GD(\mathcal{P},\mathcal{Q}) = 1.60$, $GD(\mathcal{P},\mathcal{R}) = 0.14$, $MFE(\mathcal{P},\mathcal{Q}) = 8$ and $MFE(\mathcal{P},\mathcal{R}) = 0.4$, although all elements of $\mathcal{R}$ are weakly dominated by at least one of the elements in $\mathcal{Q}$.

When the objectives present different magnitudes it is advisable to introduce the normalization of the values and both of the metrics have a medium computational cost [Knowles & Corne, 2002].

2.4.2 Spread of the Solutions

Spacing (Schott's)

The **Schott's Spacing** metric [Schott, 1995] for $\mathcal{Q}$ is defined as

$$\begin{array}{rccl} Sch: & \Omega & \rightarrow & \mathbb{R}_0^+ \\ & \mathcal{Q} & \mapsto & \sqrt{\frac{1}{|\mathcal{Q}-1|}\sum_{q\in\mathcal{Q}}(d(q)-\bar{d})^2}, \end{array} \tag{2.42}$$

where $\mathcal{Q}$ is an approximation set,

$$d(q) = \min_{p\in\mathcal{P}^*} \|\mathcal{W}(p) - \mathcal{W}(q)\| \tag{2.43}$$

is the minimal distance of solution $q \in \mathcal{Q}$ to $\mathcal{P}^*$ (in the objective space), and

$$\bar{d} = \frac{1}{|Q|}\sum_{q\in\mathcal{Q}} d(q) \tag{2.44}$$

is the mean of the $d(q)$ values.

The Spacing (Schott's) metric calculates the diversity of the solutions by computing how evenly the points are distributed. This is estimated as the statistical variance of the distance between neighbours of the non dominated solutions (in the objective space). Ideally, for an approximation set $\mathcal{Q}$ this metric should return $Sch(\mathcal{Q}) = 0$. However, it is possible that even the Pareto set is not equally distributed and therefore $Sch(\mathcal{P}^*) \neq 0$.

Spread

The **Spread** metric [Deb *et al.*, 2000] is calculated as

$$\begin{array}{rccl} \Delta: & \Omega & \rightarrow & \mathbb{R}^+ \\ & \mathcal{Q} & \mapsto & \frac{\sum_{i=1}^{m} d_i + \sum_{q\in\mathcal{Q}} |d(q)-\bar{d}|}{\sum_{i=1}^{m} d_i + |\mathcal{Q}|\bar{d}} \end{array} \tag{2.45}$$

where $\mathcal{Q}$ is an approximation set, d and $\bar{d}$ are defined in (2.43) and (2.44), and d_i is the distance between the extreme solutions of $\mathcal{P}^*$ and $\mathcal{Q}$ (see Figure 2.12).

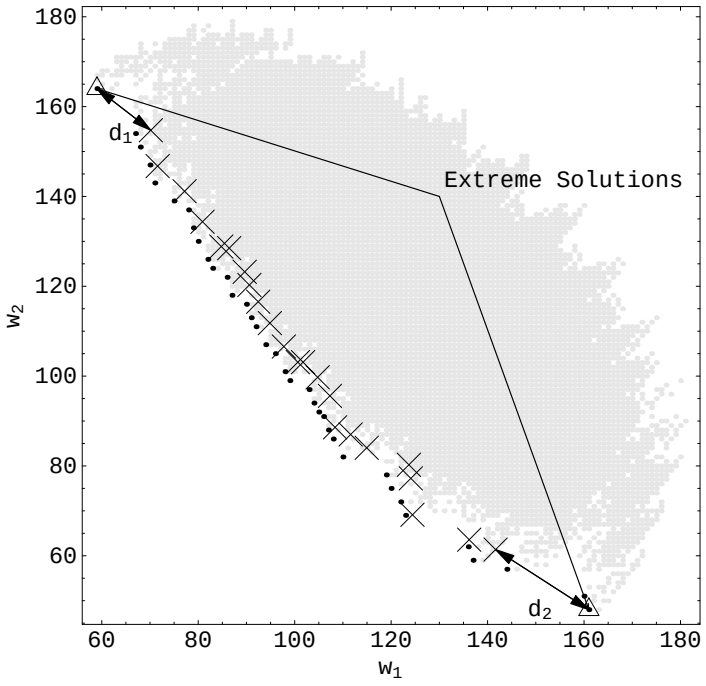


Figure 2.12: Spread metric.

The spread metric is used to calculate the non-uniformity of the distribution. $\Delta(\mathcal{Q})$ is equal to 0 in the ideal distribution: the approximation set include the extreme solution and all consecutive solutions are equally distant (in the objective space). Once more, even the Pareto front can have consecutive solution non equidistant and therefore $\Delta(\mathcal{P}^*) \neq 0$.

2.4.3 Distance to the Reference Set and Spread

Hypervolume or $\mathcal{S}$-metric

The $\mathcal{S}$-metric calculates the hypervolume of the objective space dominated by an approximation set and an anti-ideal solution [Zitzler, 1999]. The anti-ideal solution is computed as an objective vector such that its components are the maximum possible value in each objective. If it is not possible to determine the exact anti-ideal solution it should be used one outside the feasible objective space, such that the rectangle defined

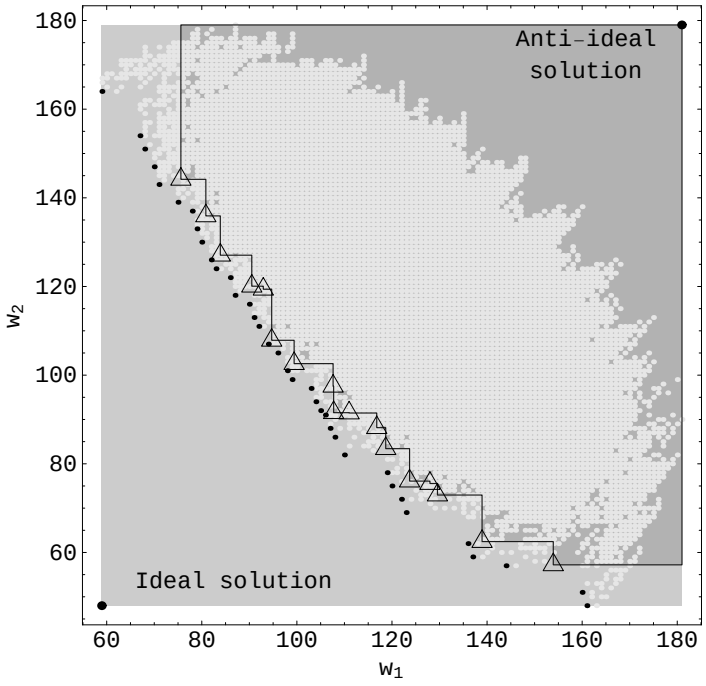


Figure 2.13: Hypervolume metric.

by the ideal and anti-ideal solution encloses the entire space, as sketched in Figure 2.13. However it should be noticed that the use of different points can lead to different results as exampled by Knowles & Corne [2002].

Since the hypervolume depends on the magnitude of the values, the objectives should be normalized. Alternatively, the normalization can somehow be avoided by computing the **hypervolume ratio** of $\mathcal{Q}$ defined as

$$\begin{array}{rccl} HVR: & \Omega & \rightarrow & \mathbb{R}^+ \\ & \mathcal{Q} & \mapsto & \frac{HV(Q)}{HV(P^*)}, \end{array} \tag{2.46}$$

where $\mathcal{Q}$ is an approximation set, $\mathcal{P}^*$ is the Pareto set (or a reference set), and $HV(\mathcal{Q})$ is the hypervolume of the region defined by the elements of $\mathcal{Q}$ and the anti-ideal solution.

This metric has the advantage of measuring both diversity and proximity since values closer to 1 indicate that the approximation set is near to the Pareto set and/or has a higher distribution, all along the Pareto front.

2.4.4 Comparison Indicators Based in Utility Functions

Three metrics based in utility functions were presented by Jaszkiewicz [2001].

$R1$ Metric

The $R1$ metric is defined as

$$R1(\mathcal{P}_1, \mathcal{P}_2, U, p) = \int_{u \in U} C(\mathcal{P}_1, \mathcal{P}_2, u) p(u) du, \tag{2.47}$$

where $\mathcal{P}_1$ and $\mathcal{P}_2$ are two approximation sets, U is a set of utility functions, $u : \mathbb{R}^m \rightarrow \mathbb{R}$, which map each approximation to an utility measure, $p(u)$ is the probability of the utility u, and

$$C(\mathcal{P}_1, \mathcal{P}_2, u) = \begin{cases} 1 & if \quad u^*(\mathcal{P}_1) < u^*(\mathcal{P}_2) \\ \frac{1}{2} & if \quad u^*(\mathcal{P}_1) = u^*(\mathcal{P}_2) \\ 0 & if \quad u^*(\mathcal{P}_1) > u^*(\mathcal{P}_2) \end{cases}, \tag{2.48}$$

with

$$u^*(\mathcal{P}) = \min_{q \in \mathcal{P}} u(q). \tag{2.49}$$

The $R1$ metric measures the probability that an approximation set is better than another over the family of utility functions U. To define U, we can use a family of Tchebycheff utility function defined as

$$u_\lambda(q, r) = \max_{j=1,2,\ldots,m} \{\lambda_j (q_j - r_j)\}, \tag{2.50}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ is a weight vector, q is a solution for which we want to measure the utility and r is a reference point. Therefore, in formula (2.47), U is set as

$$U = \left\{ u_\lambda(q, r) : \lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m) \in]0, 1[^m \wedge \sum_{i=1}^{m} \lambda_i = 1 \right\}. \tag{2.51}$$

If $R1(\mathcal{P}_1, \mathcal{P}_2, U, p) > \frac{1}{2}$ then, according to this measure, $\mathcal{P}_1$ is better than $\mathcal{P}_2$ and it will be not worse if $R1(\mathcal{P}_1, \mathcal{P}_2, U, p) \geq \frac{1}{2}$.

$R1_R$ is defined considering one of the sets as a reference set. For example, if $\mathcal{P}_r$ is the reference set, $R1_R(\mathcal{P}_1, U, p) = R1(\mathcal{P}_r, \mathcal{P}_1, U, p)$ and, therefore, near values of $R1_R(\mathcal{P}_1, U, p)$ to 0.5 indicates that more probably $\mathcal{P}_1$ is a good approximation.

$R2$ Metric

The $R2$ metric is defined as

$$R2(\mathcal{P}_1, \mathcal{P}_2, U, p) = \int_{u \in U} (u^*(\mathcal{P}_1) - u^*(\mathcal{P}_2))p(u)du. \tag{2.52}$$

where $\mathcal{P}_1$, $\mathcal{P}_2$, U, u^*, and p are defined as for equation (2.47). The $R2$ metric returns the difference between the expected values of the utilities of two approximations. Therefore, $\mathcal{P}_1$ is expected to be better than $\mathcal{P}_2$ if $R2(\mathcal{P}_1, \mathcal{P}_2, U, p) < 0$ and not worse if $R2(\mathcal{P}_1, \mathcal{P}_2, U, p) \leq 0$.

Like in the previous case, it is possible to use a reference set $\mathcal{P}_r$ to define

$$R2_R(\mathcal{P}_1, U, p) = R2(\mathcal{P}_r, \mathcal{P}_1, U, p), \tag{2.53}$$

which implies that $\mathcal{P}_1$ is expected to be a good approximation when the values of $R2_R(\mathcal{P}_1, U, p)$ are near to 0.

$R3$ Metric

Finally, the $R3$ metric is defined as

$$R3(\mathcal{P}_1, \mathcal{P}_2, U, p) = \int_{u \in U} \frac{u^*(\mathcal{P}_1) - u^*(\mathcal{P}_2)}{u^*(\mathcal{P}_1)} p(u)du, \tag{2.54}$$

where $\mathcal{P}_1$, $\mathcal{P}_2$, U, u^*, and p are defined as for equation (2.47). The $R3$ metric measures the expected proportion of superiority of one set over another.

Similarly to the two previous cases, provided a reference set it is possible to define

$$R3_R(\mathcal{P}_1, U, p) = R3(\mathcal{P}_r, \mathcal{P}_1, U, p), \tag{2.55}$$

and if $R3_R$ is a value near to 0 then $\mathcal{P}_1$ is expected to be a good approximation.

In practice, the computation of formulas (2.47), (2.52) and (2.54) can be approximated by replacing the integrals by a Riemann sum over U_Λ, where

$$U_\Lambda = \left\{ u_\lambda(q,r) : \lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m) \wedge \lambda_i \in \left\{ \frac{1}{k}, \frac{2}{k}, \ldots, \frac{k-1}{k} \right\} \wedge \sum_{i=1}^{m} \lambda_i = 1 \right\}, \tag{2.56}$$

for some large k, that is, those metric values can be approximated by

$$R1(\mathcal{P}_1, \mathcal{P}_2, r, U_\Lambda, p) = \sum_{u_{\lambda,r} \in U_\Lambda} C(\mathcal{P}_1, \mathcal{P}_2, u_{\lambda,r}, r) p(u_{\lambda,r}), \tag{2.57}$$

$$R2(\mathcal{P}_1, \mathcal{P}_2, r, U_\Lambda, p) = \sum_{u_{\lambda,r} \in U_\Lambda} \left[u^*_{\lambda,r}(\mathcal{P}_1) - u^*_{\lambda,r}(\mathcal{P}_2) \right] p(u_{\lambda,r}), \tag{2.58}$$

and

$$R3(\mathcal{P}_1, \mathcal{P}_2, r, U_\Lambda, p) = \sum_{u_{\lambda,r} \in U_\Lambda} \frac{u^*_{\lambda,r}(\mathcal{P}_1) - u^*_{\lambda,r}(\mathcal{P}_2)}{u^*_{\lambda,r}(\mathcal{P}_1)} p(u_{\lambda,r}). \tag{2.59}$$

2.4.5 Metrics Resume

The metrics can be classified in some aspect [Knowles & Corne, 2002; Zitzler *et al.*, 2003], like

- A metrics M is called a direct comparative metric if M compares two sets: given the approximation sets $\mathcal{P}$ and $\mathcal{Q}$, $M(\mathcal{P}, \mathcal{Q})$ measures how much better is $\mathcal{P}$ in relation to $\mathcal{Q}$.

- A direct comparative metric M is symmetric, if for all non dominated sets, $M(\mathcal{P}, \mathcal{Q}) = k - M(\mathcal{Q}, \mathcal{P})$ for some k.

- If M is based on a reference set then M is called a reference metric. All direct comparative metrics can be used as reference sets, although the opposite is not true.

- A metric is a cardinal measure if is based on counting the number of vector in the non-dominated set.

- If the metric is trichotomous and transitive than it induces a complete ordering of all possible non dominated sets.

In a more elaborated manner, three outperforming relations are defined as follows [Jaszkiewicz, 2001]. Let $\mathcal{P}$ and $\mathcal{Q}$ be two approximation sets and $ND(\mathcal{P}, \mathcal{Q})$ is the non dominated subset of $\mathcal{P} \cup \mathcal{Q}$ then

- $\mathcal{P}$ **weakly outperforms** $\mathcal{Q}$, $\mathcal{P}O_W\mathcal{Q}$, if $\mathcal{P} \neq \mathcal{Q}$ and all solutions in $\mathcal{Q}$ are weakly dominated by at least one solution in $\mathcal{P}$, that is,

$$\mathcal{P} \neq \mathcal{Q} \text{ and } ND(\mathcal{P}, \mathcal{Q}) = \mathcal{P}. \tag{2.60}$$

- $\mathcal{P}$ **strong outperforms** $\mathcal{Q}$, $\mathcal{P}O_S\mathcal{Q}$, if all solutions in $\mathcal{Q}$ are weakly dominated by at least one solution in $\mathcal{P}$ and exists at least one that is dominated, that is,

$$ND(\mathcal{P}, \mathcal{Q}) = \mathcal{P} \text{ and } B \backslash ND(\mathcal{P}, \mathcal{Q}) \neq \emptyset. \tag{2.61}$$

- $\mathcal{P}$ **completely outperforms** $\mathcal{Q}$, $\mathcal{P}O_C\mathcal{Q}$, if each element of $\mathcal{Q}$ is dominated by at least one of the elements of $\mathcal{P}$, that is,

$$ND(\mathcal{P}, \mathcal{Q}) = \mathcal{P} \text{ and } B \cap ND(\mathcal{P}, \mathcal{Q}) = \emptyset. \tag{2.62}$$

If two approximation sets verify O_C relation then they also verify O_S and if they verify O_S relation then they also verify O_W. This is usually represented as $O_C \Rightarrow O_S \Rightarrow O_W$.

	ER	C	GD	MFE	Δ	Sch	HVR	$R1$	$R2$	$R3$
Direct comparative	×	✓	×	×	×	×	×	✓	✓	✓
Symmetric	×	×	×	×	×	×	×	✓	✓	✓
Reference Metric	✓	×	✓	✓	×	×	✓	×	×	×
Cardinal	✓	✓	×	×	×	×	×	×	×	×
Complete Ordering	×	×	×	×	×	×	×	✓	×	×
Scaling independent	✓	✓	✓	×	×	×	✓	✓	×	×
O_W compatible	×	×	×	×	×	×	✓	✓	✓	✓
O_S compatible	×	✓	✓	×	×	×	✓	✓	✓	✓
O_C compatible	×	✓	✓	×	×	×	✓	✓	✓	✓
Utility function	×	×	×	×	×	×	×	✓	✓	✓

Table 2.3: Summary table of the properties verified by the metrics.

(✓-verifies the relation; ×- the relation does not verify, not applicable or unknown.)

If O_X is one of the outperformance relation defined above, $\mathcal{P}$ and $\mathcal{Q}$ are two approximation sets such that $\mathcal{P}O_X\mathcal{Q}$ and I is a metric then I is (weakly) compatible if I indicates that $\mathcal{P}$ is (no worse) better than $\mathcal{Q}$.

Table 2.3 resumes the properties that each of the above metrics satisfy (more details can be found in [Knowles, 2002]).

As in [Knowles & Corne, 2002] we recommend the use of $R1$, $R2$ and $R3$, three of the metrics suggested in [Jaszkiewicz, 2001] for the multiple objective maximization continuous case. These metrics have the advantage of providing comparisons based on probabilities, are non-cardinal metrics, independent of a reference set (although they induce three other metrics that depend on a reference set – $R1_R, R2_R$ and $R3_R$), are adaptable to various utility functions, and satisfy the complete, the strong and the weak outperformance relations. On the other hand, they can have a high computational cost, which is related to the necessity of setting a family of utility functions that, for

some cases as the Tchebycheff utility, can also introduce some other difficulties like the computation of an ideal point.

2.5 Spanning Trees

This section recalles some basic definitions and algorithms on arborescences. Before going on it is necessary to define a network. An undirected network is a tuple

$$\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z}), \tag{2.63}$$

where $\mathcal{V} = \{1, 2, \ldots, n\}$ is the set of nodes, $\mathcal{E} \subset \{\{i, j\} : i, j \in \mathcal{V}\}$ is the set of edges and

$$\begin{array}{rccl} \mathcal{Z}: & \mathcal{E} & \rightarrow & \mathbb{R}^m \\ & \mathcal{Z}(e) & \mapsto & (z_1(e), z_2(e), \ldots, z_m(e)) \end{array} \tag{2.64}$$

is a function that associates a weight vector to each edge.

Unless stated otherwise, it will be considered that all networks are undirected. Nevertheless, a directed network would be defined similarly with the exception of the elements of $\mathcal{E}$, which must indicate the direction of the edge with a starting and a ending node: $\mathcal{E} \subset \{(i, j) : i, j \in \mathcal{V}\} = \mathcal{V} \times \mathcal{V}$.

To simplify the notation, e_{uv} represents an edge defined by nodes u and v of $\mathcal{V}$. Furthermore, two nodes are adjacent if exists an edge connecting them and the degree of a node, u, is defined as the number of its adjacent nodes, $\delta(u)$. A network is said connected if exists a path between every pair of nodes.

$\mathcal{N}' = (\mathcal{V}', \mathcal{E}', \mathcal{Z}')$ is a subnetwork of $\mathcal{N}$ if $\mathcal{V}'$ is a subset of $\mathcal{V}$, $\mathcal{E}'$ is a subset of $\mathcal{E}$ (with all the edges defined over $\mathcal{V}'$) and $\mathcal{Z}'$ is the restriction of $\mathcal{Z}$ to $\mathcal{E}'$.

In the remaining of this section it will be considered the single objective case, that is, $m = 1$. The case where $m > 1$ will be subject of analysis in Section 3 and following.

Most of the network types considered in this thesis are spanning trees. Mathematically, a **spanning tree** of a connected network $\mathcal{N}$ is a connected subnetwork of $\mathcal{N}$ that contains all nodes of $\mathcal{N}$ and does not contain any cycle. More generally, a forest is defined as a subnetwork containing no cycles, where any two nodes are connected by at most one path.

Let $\mathcal{T}_\mathcal{N} \subseteq 2^\mathcal{E}$ be the set of all spanning trees over $\mathcal{N}$. For a complete network, Caley's formula states that $|\mathcal{T}_\mathcal{N}| = |\mathcal{V}|^{|\mathcal{V}|-2}$ (see for example [Coppersmith & Lotker, 1996]). More generally, Kirchhoff's theorem [Masbaum, 2002; Norman, 1993] states that the number of spanning trees of an undirected network is

$$|\mathcal{T}_\mathcal{N}| = \frac{1}{|\mathcal{V}|} \prod_{i=1}^{|\mathcal{V}|-1} \lambda_i \tag{2.65}$$

where $\lambda_1, \lambda_2, \ldots, \lambda_{|\mathcal{V}|-1}$ are the eigenvalues of the admittance matrix $L = [l_{i,j}]$, computed as

$$l_{i,j} = \begin{cases} \delta(i) & \text{if } i = j \\ -1 & \text{if node } i \text{ is adjacent to node } j \text{ and } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{2.66}$$

In particular, the Minimum Spanning Tree problem solution is a spanning tree with total minimal weight. Using the sum objective function defined in (2.2), the minimum spanning tree problem is stated as

$$\min_{T \in \mathcal{T}_\mathcal{N}} w_1(T), \tag{2.67}$$

where $w_1(T) = \sum_{e \in T} z_1(T)$.

The Minimum Spanning Tree problem has been studied for several decades and a list of historical developments can be found in documents like [Graham & Hell, 1985]. The following sections present three classical procedures to solve the problem.

Algorithm 5 Borůvka's algorithm.

input: $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z})$ ▷ *weighted undirected network*
output: MST ▷ *Minimum spanning tree*

function MST_BORŮVKA($\mathcal{N}$)
  for $i \in \{1, 2, \ldots, |\mathcal{V}|\}$ **do**
    $W_i \leftarrow \{v_i\}$ ▷ $\mathcal{V} = \{v_1, v_2, \ldots, v_{|\mathcal{V}|}\}$
  end for
  $MST \leftarrow \emptyset$
  $F \leftarrow \{W_1, W_2, \ldots, W_{|\mathcal{V}|}\}$
  while $|MST| < |\mathcal{V}| - 1$ **do**
    for $U_i \in F$ **do**
      find a minimal weight edge e_{uv} such that $u \in U_i$ and $v \in U_i' \neq U_i$
      $MST \leftarrow MST \cup \{e_{uv}\}$
    end for
    for $U_i \in F$ **do**
      $MERGE(U_i, U_i')$
    end for
  end while
  return MST
end function

2.5.1 Borůvka's Algorithm

This algorithm covers problems where all the edges have different weights. It starts with a forest constituted by the vertices of $\mathcal{N}$. Then it examines each vertex to find the lighter edge that connects two distinct forest elements, and merges those elements by adding that edge. Since the result is always a forest, the process is repeated until a single (spanning) tree is completed [Jungnickel, 1999].

Algorithm 5 describes the procedure where the $MERGE$ function represents the disjoint union of the subtrees into F (a set that contains the forest elements).

Algorithm 6 Prim's algorithm.

input: $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z})$ ▷ *weighted undirected network*
output: MST ▷ *Minimum spanning tree*

1: **function** MST_PRIM($\mathcal{N}$)
2: Select a node v from $\mathcal{V}$
3: $T \leftarrow \{v\}$
4: $MST \leftarrow \emptyset$
5: **while** $|MST| < |\mathcal{V}| - 1$ **do**
6: Find the minimal weight edge e_{uv} such that $u \in T$ and $v \in \mathcal{V} - T$
7: $T \leftarrow T \cup \{v\}$
8: $MST \leftarrow MST \cup \{e_{uv}\}$
9: **end while**
10: **return** MST
11: **end function**

2.5.2 Prim's Algorithm

The Prim's algorithm was first described by Vojtěch Jarník in 1930 [Jarník, 1930]. Later it was independently rediscovered by Robert Prim and Edsger Dijkstra in the 1950's decade [Prim, 1957].

Basically, the procedure starts by picking one node. Then the edges with small weights that keep the feasibility are added until the spanning tree is complete (see for example [Jungnickel, 1999]). Algorithm 6 has a high level description of the process.

The complexity of the algorithm depends on step 6. If the data structure used to keep the information is an adjacency matrix then the algorithm has $\mathcal{O}(|\mathcal{V}|^2)$ complexity. However, using a binary heap the complexity can be improved to an $\mathcal{O}(|V| \log |V| + |\mathcal{E}| \log |\mathcal{V}|)$ and if a Fibonacci heap is used, then its complexity is $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|)$.

2.5.3 Kruskal's Algorithm

Kruskal's Algorithm is perhaps the easiest to understand of the the three algorithms here presented [Kruskal, 1956].

Algorithm 7 Kruskal's algorithm.

input: $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z})$ ▷ *weighted undirected network*
output: MST ▷ *Minimum spanning tree*

```
1: function MST_KRUSKAL(N)
2:     Sort the elements of E in non decreasing weights: {e_1, e_2, ..., e_|E|}
3:     i ← 1
4:     while |MST| < |V| − 1 do
5:         if MST ∪ {e_i} is a forest then
6:             MST ← MST ∪ {e_i}
7:         end if
8:         i ← i + 1
9:     end while
10:    return MST
11: end function
```

The process starts with an empty forest. Then, while the spanning tree is not complete, the edges are successively tested, sorted by the increase values of their weights. An edge is included in the forest if it does not create any cycle, that is, it keeps the forest property (see for example [Jungnickel, 1999]).

Algorithm 7 presents the description of the procedure which has an $\mathcal{O}(|\mathcal{E}| \log |\mathcal{E}|)$ time complexity due to the necessity of sorting the edges in an increasing order.

2.5.4 Other Methods

The three classical procedures presented above have been improved by several researchers:

- Gabow *et al.* [1989] presented a $\mathcal{O}(|\mathcal{E}| \log \beta(|\mathcal{E}|, |\mathcal{V}|))$ method where β is the slow growing function defined by $\beta(m, n) = \min\{i : \log^{(i)} m \leq \frac{m}{n}\}$;
- A randomized linear-time algorithm for finding minimum spanning trees with expected running time of $\mathcal{O}(|\mathcal{E}|)$ was proposed by Kleint & Tarjan [1994];

- A deterministic algorithm with time complexity $\mathcal{O}(|\mathcal{E}|\alpha(|\mathcal{E}|,|\mathcal{V}|))$ was proposed by Chazelle [2000], where α is the inverse of the Ackermann's function (an even slowly growing function than β) defined as

$$\alpha(m,n) = \min\left\{i \geq 1 : A\left(i, \left\lfloor \frac{m}{n} \right\rfloor\right) \geq \log_2 n\right\}, \tag{2.68}$$

where

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases} ; and \tag{2.69}$$

- Chazelle *et al.* [2005] presented a method to approximate the minimum spanning tree weight in sublinear time.

2.5.5 Related Problems

Several variants of the Minimum Spanning Tree problem have been studied by computer science researchers. Those variants arise from imposing restrictions to one or more of the trees parameters like the degree of the nodes, the number of leaves, the diameter of the tree (maximum distance between two nodes) and others.

In [Ausiello *et al.*, 1999] it is a list of variants of the spanning tree problem, including several $\mathcal{NP}$ -complete cases.

2.6 Summary

In this chapter has been presented some preliminary definitions and results necessary for the remaining thesis. Those concepts were specially concerned with the multiple objective optimization of networks (in particular spanning trees).

The multiple objective optimization problems are, by nature, extremely difficult to solve. The conjunction of incompatible objectives implies that the solution of a

multiple objective problem is not a single solution but a set of trade off answers. The necessity to find equilibrium between the objectives implies that hardly ever exists specific algorithms to exactly solve the problems in acceptable time. In the deterministic field, the most common solutions include the transformation of the multiple objectives problem into a single objective problem for which, hopefully, exists an efficient algorithm capable of solving it.

Recent meta-heuristics have proved that they return good approximations to the most demanding problems, recurring to suitable adaptations for the problems.

On the other hand, the use of different methods require the existence of quality indicator capable of, at least probabilistically, infer which of the methods has a more acceptable behaviour.

In the end of these preliminaries chapter, some basic results on spanning trees were recalled, subject that will be further developed in the following chapters, for the multiple objective case.

I think that I shall never see
A graph more lovely than a tree.
A tree whose crucial property
Is loop-free connectivity.
A tree which must be sure to span.
So packets can reach every LAN.

First the Root must be selected
By ID it is elected.
Least cost paths from Root are traced
In the tree these paths are placed.
A mesh is made by folks like me
Then bridges find a spanning tree.

Radia Perlman, inventor of the STP

3

Multiple Objectives Spanning Trees

Contents

3.1 Overview

The Multiple Objective Minimum Spanning Trees problem, classified as a combinatorial optimization problem with NP-complete complexity, is the main concern of this chapter. For this problem exists a general absence of theoretical results, which motivates the analysis of a set of properties that will be proposed in the following sections.

Relatively to the Multiple Objective Minimum Spanning Trees problem, several theoretical and practical applications use arborescences representations, which are as-

sumed as unavoidable in many strategic optimization fields, as the final result or as some intermediate step of a larger problem. For example, arborescences are applied in practical circumstances in

VLSI circuits – It is known that the design, build and functioning of VLSI circuits has to be optimized using, most of the times, spanning and/or Steiner trees. For example, the topological optimization of the circuits creates high performance interconnections through the minimization of the total wire length, the critical path length, or the registered delays[Atallah, 1999; Cong *et al.*, 1996].

Telecommunications networks

- IEEE standard 802.1D (Spanning Tree Protocol [802.1D, 2004]) – In redundant telecommunication networks there may exist several paths in which two nodes can connect. The existence of multiple paths requires the definition of a set of rules to avoid traffic congestion in internal network loops. The Spanning Tree Protocol configures a spanning tree structure accessed by each bridge or router, which is the only way that traffic can be sent to all visible nodes. The spanning tree is defined using the shortest distance to a root bridge or router, using bandwidth as a measurement. The bi-objective spanning tree-based genetic algorithm that minimizes the communication cost and the average message delay is presented by Gen & Li [1998]; and Abuali *et al.* [1994] study the probabilistic minimum spanning tree. Other relevant parameters can be optimized like the reliability of the edges, the probability of packet corruption, or the average number of edges that a random packet has to cross to reach a destination.

- Quality of Service problems – The Spanning Tree Protocol, in the intent of avoiding traffic loops, forces the packets to be tunnelled into the branches of

a single tree. This implies that often, even in normal circumstances, some edges end up being congested, although the blocked (not used) branches have plenty of available bandwidth[Yu *et al.*, 2003]. Possible solutions include the use of VLAN spanning trees that maintain instance trees for each VLAN configured in the network [Cisco Systems, 2005].

Electrical or cable TV networks – By nature, electrical and cable TV networks assume a spanning tree topology. In fact, as referred in the previous chapter, Borůvka was the first to formalize the problem and use its proposed solution to optimize an electrical network. Other application examples are illustrated in the ability to handle large-scale distribution-network problems as presented by Carvalho *et al.* [2001].

Road networks – In this field the use of spanning trees or related problems is also used often [Hu, 1974]. For example, in the design stage parameters like total length, environmental impact, network diameter, or node degree have to be optimized. The emergency circulation plans is another example of problem that can be solved as a spanning tree where factors, like the (average) required time and distance to reach every possible emergent point or region in the network, have to be optimized.

Astronomy and astrophysics – Spanning tree are used to characterize the aggregation level of a given set of points, identifying clusters associated to galaxies[Adami & Mazure, 1999].

Medical imaging – In magnetic resonance imaging the reconstructed images can be decomposed into amplitude images and phase maps. The minimum spanning tree algorithm is used as an unwrapping method to process the acquired signal and the sequenciation of those images [An *et al.*, 2000].

As exampled, by the number of possible applications along with their practical importance, the study of arborescences problems is made in very distinct areas of science.

Often the problem resumes to find a tree with minimal total weight: the Minimum Spanning Tree problem. However, very distinct cases can appear based in aspects, like the type of optimization problem (minimization/maximization), the restrictions on the nodes/edges, number of leaves, or the diameter of the tree. A comprehensive description of these problems can be find in [Ausiello *et al.*, 1999].

In the remaining of this chapter, it is presented the mathematical formalization of the problem followed by an overview of its computational complexity along with some known algorithmic proposals. The last sections introduce a set of results and do an analysis that motivates the subsequent chapters.

3.2 Problem Definition

To formalize the problem it is necessary to recall the network definition previously presented. An undirected network is a tuple

$$\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z}), \tag{3.1}$$

where $\mathcal{V}$ is the set of nodes or vertices, $\mathcal{E}$ is the set of edges defined between the nodes of $\mathcal{V}$ and $\mathcal{Z}$ is a weight vector function that maps each element of $\mathcal{E}$ into $\mathbb{R}^m$. Furthermore, it will be considered that all networks are connected, that is, there exists always a path between every pair of nodes.

For a subnetwork T of $\mathcal{N}$, the sum objective function, also called weight or cost

vector of T, is defined as in (2.4) by

$$\begin{aligned} \mathcal{W}(T) &= (w_1(T), w_2(T), \ldots, w_m(T)) = \\ &= \left(\sum_{e \in T} z_1(e), \ldots, \sum_{e \in T} z_m(e) \right) = \\ &= \sum_{e \in T} \mathcal{Z}(e), \end{aligned} \tag{3.2}$$

where $e \in T$ means that e is an edge of T and z_i $(i = 1, 2, \ldots, m)$ are the components of the function $\mathcal{Z}$ defined in (2.64).

A spanning tree T is an acyclic and connected sub-network of $\mathcal{N}$ so that the set of nodes of T is equal to $\mathcal{V}$,

$$T = (\mathcal{V}, \mathcal{E}_T \subset \mathcal{E}, \mathcal{Z}). \tag{3.3}$$

It will be also considered the use of the classical dominance relations, set efficiency and Pareto's optimality for the multiple criteria optimization problems (refer to Chapter 2).

If $\mathcal{T}_\mathcal{N}$ is the set of all spanning trees over $\mathcal{N}$ then the optimization problem can be stated as

$$\min_{T \in \mathcal{T}_\mathcal{N}}{}^{+} \ \mathcal{W}(T). \tag{3.4}$$

where $\min^+$ indicates that all objectives are to be minimized.

To simplify the notation, let $\mathcal{V}_T$ denote the set of nodes of network T, $e_{uv} \in \mathcal{E}$ is the edge defined by nodes u and v and $T' = T - \{e\} \cup \{f\}$ is a network obtained by removing edge e and adding edge f to T, that is,

$$T' = (\mathcal{V}, \mathcal{E}_T - \{e\} \cup \{f\}, \mathcal{Z}), \tag{3.5}$$

where $\mathcal{E}_T$ is the set of edges of T.

3.3 Complexity

Several complete studies about the complexity of the Multiple Objective Minimum Spanning Trees problems are present in different papers. Overviews about the complexity of the spanning tree problems on undirected graphs are made in [Camerini *et al.*, 1980, 1983, 1984]. Their comprehensive study resumes the complexity of several single and multiple objective instances, classifying them as solvable in polynomial time or $\mathcal{NP}$ -complete. They prove or present references that for more than fifty variants, and excluding some very rare cases, the (weighted) undirected multiple objective spanning trees problems are $\mathcal{NP}$-complete problems. This includes the Multiple Objective Minimum Spanning trees problem, the main concern of this document.

Later, in [Hamacher & Ruhe, 1994] it was proved that the problem is also $\mathcal{NP} - \sharp$, that is, it can have an exponential number of efficient trees.

Methods to scan all spanning trees, like the recursion algorithms presented in [Gabow & Myers, 1978; Ramos *et al.*, 1998; Shioura *et al.*, 1997], have a high order of complexity with one of the best taking $\mathcal{O}(|\mathcal{T}_{\mathcal{N}}| + |\mathcal{V}| + |\mathcal{E}|)$ time and $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ space, which usually became untreatable for small networks.

The linear and non-linear combination of objectives for the directed and the undirected networks are analysed in [Dell'Amico & Maffioli, 1996, 2000]. As expectable, results show that the linear combination of "easy" problems continues to be "easy", which does not necessarily verify for the non-linear case. This implies that for each linear combination of the objectives, the Multiple Objective Minimum Spanning Trees problem is transformed into a single objective problem, for which exists an efficient algorithm to solve it. This process can be used to obtain a set of efficient solutions, which are recognized as supported solutions.

A Prim's based algorithm to obtain all efficient spanning trees was proposed by Corley [1985]. Unfortunately, the process is based over a wrong premise (as exampled

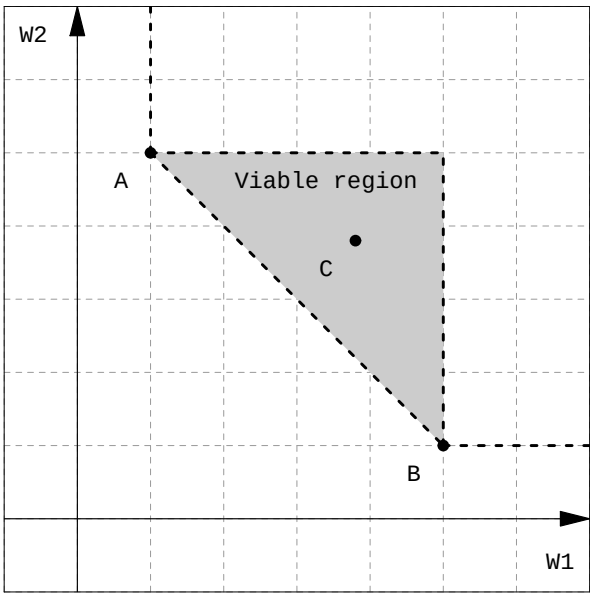


Figure 3.1: Viable region.

in [Hamacher & Ruhe, 1994]) that if T' is a subtree of an efficient spanning tree T, e_{uv} is an edge of $\mathcal{E}$ such that $u \in \mathcal{V}_{T'}$ and $v \notin \mathcal{V}_{T'}$ then e_{uv} is an edge in an efficient spanning tree if and only if e_{uv} is efficient over the set of edges defined between $\mathcal{V}_{T'}$ and $\mathcal{V} - \mathcal{V}_{T'}$.

Also in [Hamacher & Ruhe, 1994] it is presented a two-stage method to construct a well-distributed set of efficient solutions, generalized to a well-distributed approximation set in formula (2.16), for the Multiple Objective Minimum Spanning Trees problem. In the first stage, the supported solutions are computed using the Weighted Sum method, described in Section 2.3, and a strategy based in the slopes values between lexicographically consecutive solutions. In the second stage, a neighbourhood search is made to complete the well-distributed set looking for solutions in the viable regions. Here the viable regions are the non-dominated triangular regions defined by lexicographic consecutive solutions in the objective space. Figure 3.1 presents the viable region (in grey), and three solutions: A and B are supported solutions and C is not a supported solution. The process stops in any of the steps, whenever the well-

distributed condition is verified. The algorithm presents some limitations as the fact that it is established over results for the bi-objective case and it does not predict the cases where the fronts have gaps, where it is impossible to satisfy the well-distributed condition.

Another two-phase method is also presented in [Steiner & Radzik, 2003]. The method starts by computing the set of supported solutions (like in [Hamacher & Ruhe, 1994]). Then, they apply a k-Best Minimum Spanning Trees algorithm to search for other efficient solutions in the viable region. The proposed method is then compared with another two-phase method where a Branch-and-Bound technique replaces the k-Best Minimum Spanning Trees in the second phase. They conclude that, for the tested problems, the k-Best Minimum Spanning Trees strategy outperforms the two-phase Branch-and-Bound method.

Another Branch-and-Bound algorithm for the bi-objective case is presented in [Sourd *et al.*, 2006]. The method uses the same line of process that was above described: firstly it computes the supported solutions, which are then used to find the others efficient solutions by a Branch-and-Bound method.

However, we are not aware of any reference to the study of the Multiple Objective Minimum Spanning Trees problem using meta-heuristic techniques.

Some exceptions issue variants of the Multiple Objective Minimum Spanning Trees problem, like the k-Degree Multiple Objective Minimum Spanning Trees problem, where genetic algorithms were proposed [Knowles & Corne, 2001a; Knowles *et al.*, 1999; Raidl & Julstrom, 2000; Zhou & Gen, 1999], and the Multiple Destination Routing Problem [Leung *et al.*, 1998].

3.4 Efficiency and Dominance

As already referred, there exists a general absence of documented theoretical results associated to the Multiple Objective Minimum Spanning Trees problem. Therefore, as an effort to better characterize the problem in study, in this section we generalize some of the propositions known for the single objective case and introduce some new results. More specifically, several theorems with premises over the edges local, global, and cut efficiency are proved. Those results allow to conclude that under certain conditions it is possible to assure that some edges belong to the efficient trees.

In the first remark, it is proved that if for some sub-network of $\mathcal{N}$ we can replace an edge by another that dominates the first one, then the resulting network also dominates the former.

Remark 3.1: Let e be an edge of a non empty sub-network T of $\mathcal{N}$. If f is an edge of $\mathcal{E} - \mathcal{E}_T$ and f dominates e, $f \prec e$, then a network T' obtained by adding f to T and removing e from T, $T' = T - \{e\} \cup \{f\}$, dominates T.

Proof. Since f dominates e then $z_i(f) \leq z_i(e)$ for all $i \in M = \{1, 2, \ldots, m\}$ and $z_j(f) < z_j(e)$ for some $j \in M$. Therefore, from the first inequality, for all $i \in M$ the following assertion is verified

$$w_i(T') = w_i(T) + z_i(f) - z_i(e) \leq w_i(T) \tag{3.6}$$

and, from the second, it is guaranteed that exists $j \in M$ such that

$$w_j(T') = w_j(T) + z_j(f) - z_j(e) < w_j(T), \tag{3.7}$$

which proves that T' dominates T. □

Considering that $\mathcal{E}|_T$ is the set of edges defined in $\mathcal{N}$ by $\mathcal{V}_T$ and $\mathcal{N}|_T = (\mathcal{V}_T, \mathcal{E}|_T, \mathcal{Z})$ then, we are able to define the tree efficiency concept and prove that a spanning tree

is efficient if and only if all its subtrees are also efficient, as follows. We say that T is a **efficient subtree** if T is efficient over $\mathcal{N}|_T$. By the previous definition, $T \in \mathcal{T}_\mathcal{N}$ is an **efficient tree** if there is no other tree $S \in \mathcal{T}_\mathcal{N}$ such that S dominates T.

Theorem 3.1. *A spanning tree is efficient if and only if all its subtrees are efficient.*

Proof. Suppose that T has a subtree T' that is not efficient over

$$\mathcal{N}|_{T'} = (\mathcal{V}_{T'}, \mathcal{E}|_{T'}, \mathcal{Z}). \tag{3.8}$$

Then, there exists a subtree $S' \in \mathcal{N}|_{T'}$ such that $S' \prec T'$, that is, $w_i(S') \leq w_i(T')$ for all $i \in M$ and exists $j \in M$ such that $w_j(S') < w_j(T')$. Now, for

$$S = T - T' \cup S' \tag{3.9}$$

we have

$$w_i(S) = w_i(T) - w_i(T') + w_i(S') \leq w_i(T) \text{ for all } i \in M \tag{3.10}$$

and

$$w_j(S) = w_j(T) - w_j(T') + w_j(S') < w_j(T) \text{ for some } j \in M, \tag{3.11}$$

which means that S dominates T and consequently we have a contradiction. Therefore, all subtrees of T must be efficient.

To prove the reciprocal, it is enough to observe that T is a subtree of T and by the hypothesis all subtrees of T are efficient. □

Theorem 3.1 suggests that if a subtree of an efficient spanning tree is replaced by other efficient subtree then the resulting spanning tree would also be efficient. In fact, this is not true in all situations. Consider, for example, the network in Figure 3.2 and the eleven possible spanning trees with weights $\mathcal{W}(T_1) = (19, 19)$, $\mathcal{W}(T_2) = (18, 18)$, $\mathcal{W}(T_3) = (19, 19)$, $\mathcal{W}(T_4) = (18, 22)$, $\mathcal{W}(T_5) = (16, 20)$, $\mathcal{W}(T_6) = (17, 21)$,

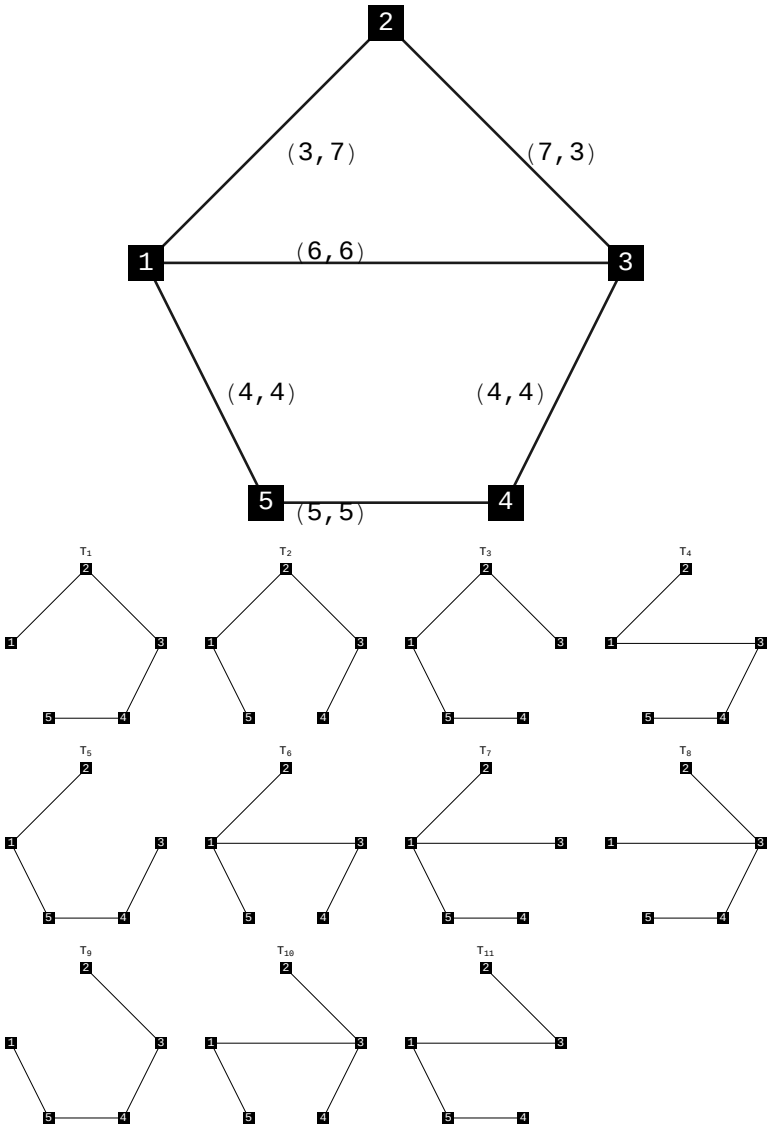


Figure 3.2: A breakdown network into spanning trees.

$\mathcal{W}(T_7) = (18, 22)$, $\mathcal{W}(T_8) = (22, 18)$, $\mathcal{W}(T_9) = (20, 16)$, $\mathcal{W}(T_{10}) = (21, 17)$ and $\mathcal{W}(T_{11}) = (22, 18)$. The efficient spanning trees are T_2, T_5 and T_9. Over the sub-network defined by nodes $\{1, 2, 3\}$, $S_1 = \{e_{12}, e_{23}\}$ and $S_2 = \{e_{12}, e_{13}\}$ are two efficient subtrees. Although tree T_2 is efficient, the tree obtained by replacing the subtree S_1 of T_2 by subtree S_2 conducts to a tree that is not efficient, T_6.

Next are introduced two concepts that will be used in the following remarks. A **cut of** $\mathcal{V}$ is a partition of $\mathcal{V}$ in two non empty sets $\mathcal{V}_1$ and $\mathcal{V}_2$, that is, $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$

and $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$. Let us denote the set of the edges incident with one node in $\mathcal{V}_1$ and with a node in $\mathcal{V}_2$ by $\mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)$ and $e_{uv} \in \mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)$ will mean that e_{uv} is an edge of the cut $\mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)$ such that $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$. Furthermore, if $\mathcal{N}$ is a connected network, we say that an edge $e_{uv} \in \mathcal{E}$ is a **bridge** if by removing it the network becomes disconnected.

The next remark states a condition to replace an edge of a tree so that the resulting network is also a tree.

Remark 3.2: Let T be a tree, $e_{uv} \in \mathcal{E}_T$ and $\{\mathcal{V}_u, \mathcal{V}_v\}$ the cut induced by removing e_{uv} from T. If $e_{wz} \in \mathcal{E}(\mathcal{V}_u, \mathcal{V}_v)$ then $T' = T - \{e_{uv}\} \cup \{e_{wz}\}$ is a tree.

Proof. Let T_u and T_v be the subtrees of T defined by $\mathcal{V}_u$ and $\mathcal{V}_v$, respectively. Since T is a tree, T_u and T_v are also trees (they are subtrees of T) and by hypothesis they are node-disjoint. Therefore, for $e_{wz} \in \mathcal{E}(\mathcal{V}_u, \mathcal{V}_v)$,

$$T' = (\mathcal{V}, \mathcal{E}_{T_u} \cup \mathcal{E}_{T_v} \cup \{e_{wz}\}, \mathcal{Z}) \tag{3.12}$$

cannot contain any cycles and is connected, which proves that T' is a tree. □

Let $\mathcal{E}_u$ be the set of edges with starting node u. Then, it is possible to introduce the local efficiency of the edges as follows. An edge $e_{uv} \in \mathcal{E}$ is **local** or **node efficient** if e_{uv} is not dominated by any edge in $\mathcal{E}_u$, that is,

$$\forall e_{uw} \in \mathcal{E}_u : e_{uw} \not\prec e_{uv}. \tag{3.13}$$

Furthermore, an edge is **local** or **node dominant** if e_{uv} dominates all edges in $\mathcal{E}_u - \{e_{uv}\}$, that is,

$$\forall e_{uw} \in \mathcal{E}_u - \{e_{uv}\} : e_{uv} \prec e_{uw}. \tag{3.14}$$

Similar definitions can be made for cuts as follows. If $\{\mathcal{V}_1, \mathcal{V}_2 = \mathcal{V} - \mathcal{V}_1\}$ is a cut of $\mathcal{V}$ then an edge $e_{uv} \in \mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)$ is said $\{\mathcal{V}_1, \mathcal{V}_2\}$**-cut efficient** if e_{uv} is not dominated

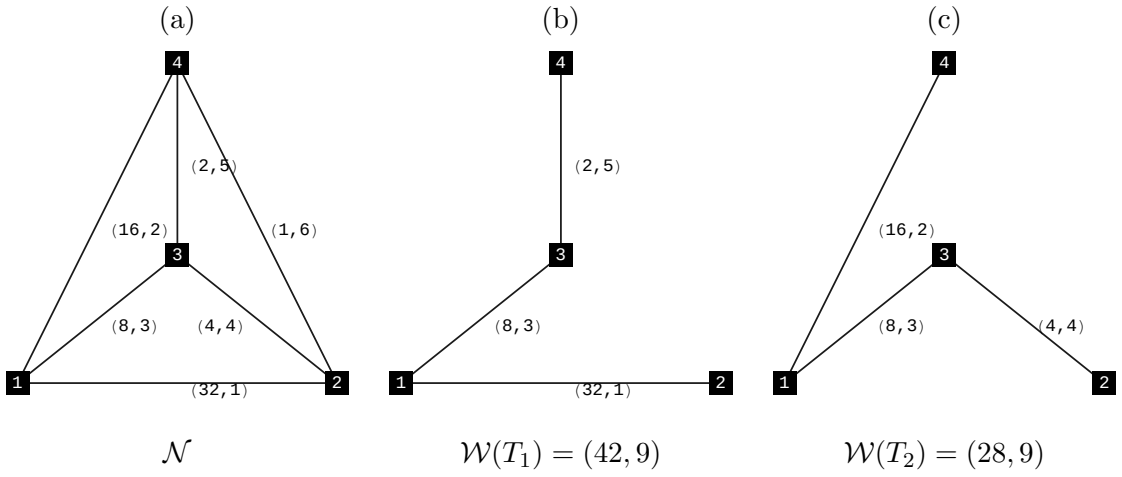


Figure 3.3: Hamacher and Ruhe example.

by any edge in $\mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)$ and edge $e_{uv} \in \mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)$ is $\{\mathcal{V}_1, \mathcal{V}_2\}$**-cut dominante** if e_{uv} dominates every element of $\mathcal{E}(\mathcal{V}_1, \mathcal{V}_2) - \{e_{uv}\}$ or e_{uv} is a bridge.

As already referred, in [Corley, 1985] it is made an attempt to obtain a necessary and sufficient condition for the Multiple Objective Minimum Spanning Trees efficiency, based on the cut efficiency of the edges. Latter it was found a counter-example for the proposed result [Hamacher & Ruhe, 1994], which is sketched in Figure 3.3. It can be observed that all edges are local efficient but exists at least one spanning tree that is not efficient (for example T_1).

In fact, we are not aware of any necessary and sufficient condition, based on the local efficiency of the edges, proving that a tree is or is not efficient. Even if an edge is locally dominated it can still be a part of an efficient spanning tree, as exampled in Figure 3.4. In that example, Edge e_{15} is dominated by edges e_{12} and e_{45}. Nevertheless, e_{15} belongs to the efficient spanning trees $T_3 = \{e_{12}, e_{15}, e_{45}, e_{23}\}$ and $T_4 = \{e_{12}, e_{15}, e_{45}, e_{34}\}$.

In the following two theorems it is proved that, in some conditions, a same edge can belong to all the efficient spanning trees.

Theorem 3.2. *If e_{uv} is local dominant or e_{uv} is a bridge then e_{uv} belongs to all the efficient trees.*

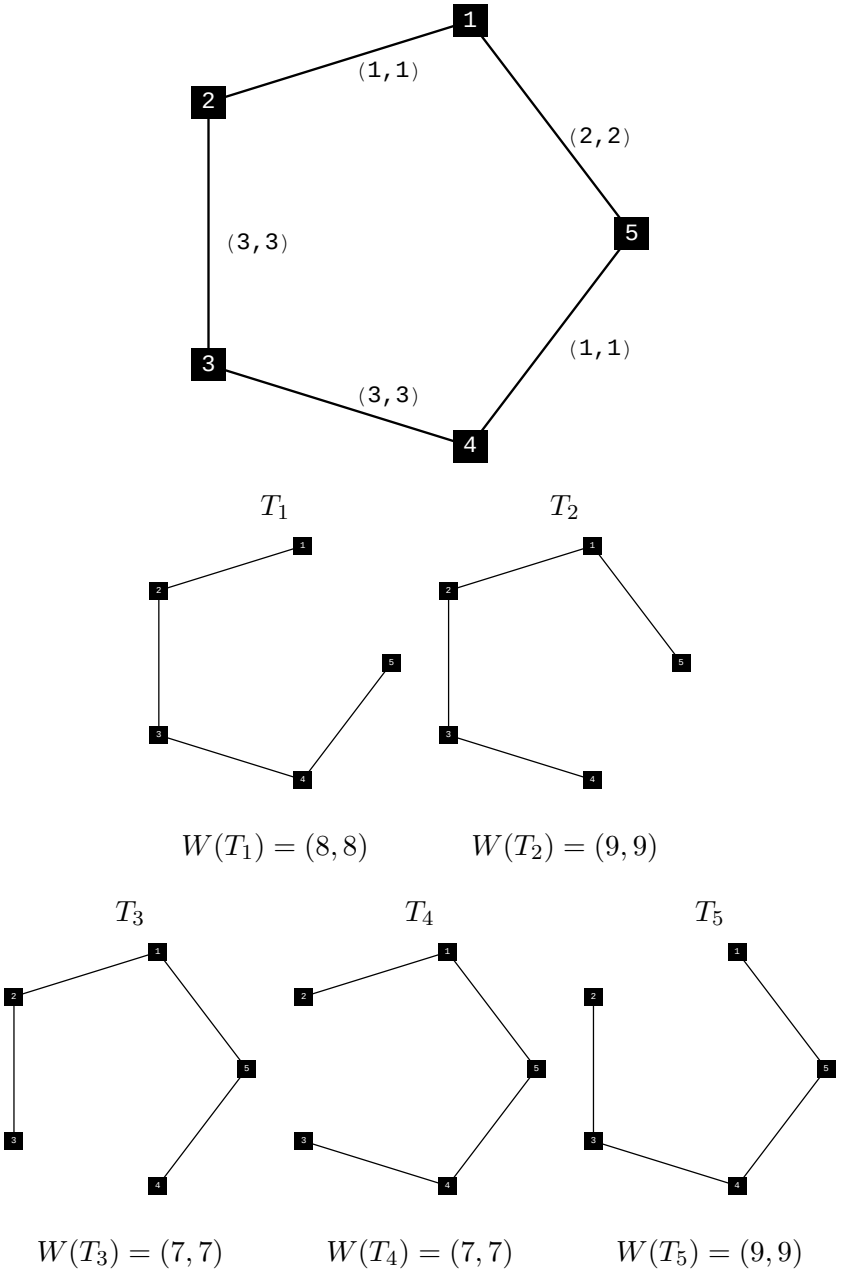


Figure 3.4: Example of a locally dominated edge that belongs to the efficient spanning trees.

Proof. The case in which e_{uv} is a bridge is trivial. Suppose now, that e_{uv} is node dominant, T is an efficient tree from $\mathcal{T}_{\mathcal{N}}$, $(u, u_1, u_2, \ldots, u_p, v)$ is the path from u to v in T, e_{uv} does not belong to T and S is the tree obtained by replacing edge e_{uu_1} by edge e_{uv} in T (Remark 3.2). Now, using Remark 3.1 and since e_{uv} dominates e_{uu_1}, we

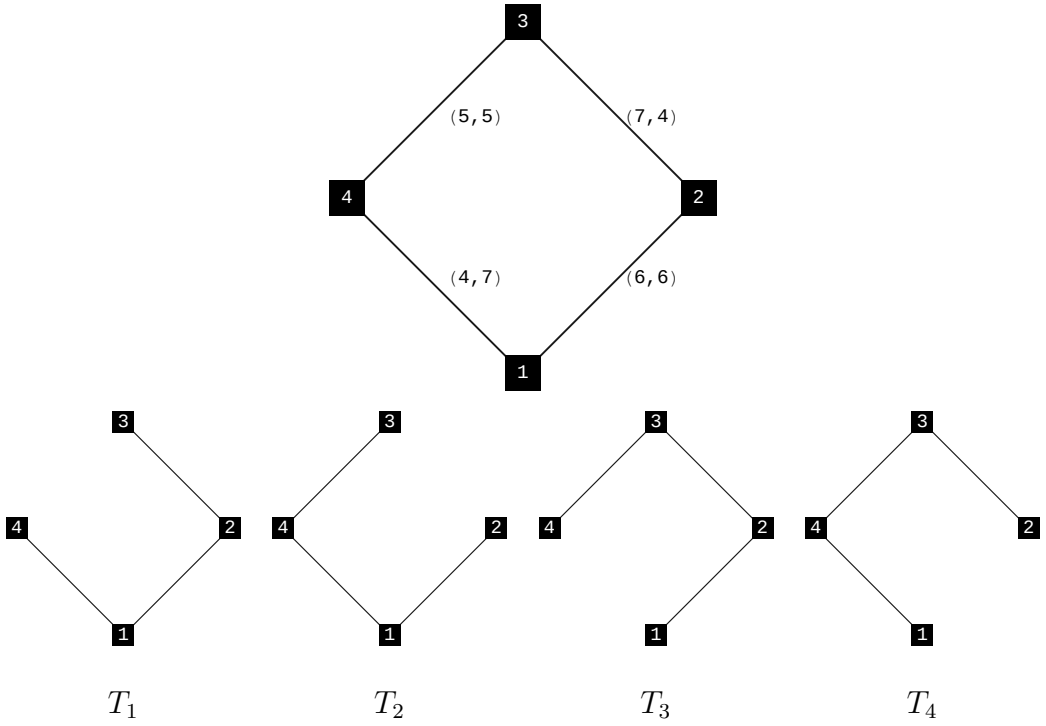


Figure 3.5: Example where all the edges are node efficient.

can conclude that S dominates T. Therefore, T is not an efficient tree, contradicting our hypothesis, which implies that e_{uv} must belong to all efficient trees. □

From Figure 3.5 it is possible to observe that the reciprocal of Theorem 3.2 is not valid. In that example, there are four spanning trees $T_1 = \{e_{12}, e_{23}, e_{34}\}$, $T_2 = \{e_{14}, e_{23}, e_{34}\}$, $T_3 = \{e_{14}, e_{12}, e_{34}\}$ and $T_4 = \{e_{14}, e_{12}, e_{23}\}$, with weights $\mathcal{W}(T_1) = (18, 15)$, $\mathcal{W}(T_2) = (16, 16)$, $\mathcal{W}(T_3) = (15, 18)$ and $\mathcal{W}(T_4) = (17, 17)$, respectively. In this case, edge e_{34} belongs to all efficient spanning tree, T_1, T_2 and T_3, although it is not a bridge neither local dominant.

However, it is easy to see that e_{34} dominates all the edges in the cut $\mathcal{V}_1 = \{1, 4\}$, $\mathcal{V}_2 = \{2, 3\}$. This suggests the next theorem where it is stated another condition concluding that the same edge can belong to all efficient spanning trees.

Theorem 3.3. *Let $\{\mathcal{V}_1, \mathcal{V}_2\}$ be a cut. An edge e_{uv} is $\{\mathcal{V}_1, \mathcal{V}_2\}$-cut dominant if and only if e_{uv} belongs to all efficient spanning trees.*

Proof. The case in which $\mathcal{E}(\mathcal{V}_1, \mathcal{V}_2) - \{e_{uv}\} = \emptyset$ it is trivial that e_{uv} belongs to all spanning tree, since it is a bridge. Suppose now that $\mathcal{E}(\mathcal{V}_1, \mathcal{V}_2) - \{e_{uv}\} \neq \emptyset$, e_{uv} does not belong to some efficient tree T and let e_{wz} be an edge in $\mathcal{E}(\mathcal{V}_1, \mathcal{V}_2) - \{e_{uv}\}$ that belongs to the path between u and v in T. Then, by Remark 3.2, $S = T - \{e_{wz}\} \cup \{e_{uv}\}$ is a spanning tree and, since $e_{uv} \prec e_{wz}$, from Remark 3.1 we can conclude that S dominates T which contradicts our hypothesis that T is an efficient spanning tree. Thus e_{uv} must belong to all efficient spanning trees.

To prove the reciprocal suppose that e_{uv} is not a bridge (if it is a bridge then by definition e_{uv} is cut dominant as required). Assuming that e_{uv} belongs to all efficient trees but does not exist a cut $\{\mathcal{V}_1, \mathcal{V}_2 = \mathcal{V} - \mathcal{V}_1\}$ such that $e_{uv} \in \mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)$ and e_{uv} is cut dominant then, since e_{uv} is not cut dominant, one of the following must happen:

- e_{uv} is cut dominated for all cuts, in which case it is obvious that it cannot belong to all efficient spanning trees. In fact, it can not belong to any efficient spanning tree, since the exchange of e_{uv} for any dominating edge, of the edges induced by the cut, would create a dominating solution; or

- Exists at least a cut $\{\mathcal{V}_1, \mathcal{V}_2 = \mathcal{V} - \mathcal{V}_1\}$ such that e_{uv} is $\{\mathcal{V}_1, \mathcal{V}_2\}$-cut efficient. Since e_{uv} is $\{\mathcal{V}_1, \mathcal{V}_2\}$-cut efficient (but not dominant) exists $e_{u'v'} \in \mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)$ such that e_{uv} is not comparable to $e_{u'v'}$, $e_{uv} \sim e_{u'v'}$. This implies the existence of some $j \in M$ such that $z_j(e_{u'v'}) < z_j(e_{uv})$. Let T be the efficient spanning tree with minimal j-weights, that is,

$$T = \arg\min_{S \in \mathcal{P}^*} w_j(S) \tag{3.15}$$

  where $\mathcal{P}^*$ is the Pareto set. Then $T' = T \cup \{e_{u'v'}\} - \{e_{uv}\}$ is a tree such $w_j(T') < w_j(T)$ which contradicts the hypothesis stated in (3.15).

□

In the next result it is proposed a non-optimality condition. If T is a tree and e an edge that does not belong to T then $T \cup \{e\}$ contains an unique cycle denoted by $C_T(e)$.

Theorem 3.4. *If a spanning tree T is efficient and e belongs to $\mathcal{E} - \mathcal{E}_T$ then e does not dominate any edge in the cycle $C_T(e)$, that is, $\forall_{e \in \mathcal{E}-\mathcal{E}_T} \forall_{f \in C_T(e)} : e \not\prec f$.*

Proof. Suppose that T is an efficient spanning tree, and exists $e \in \mathcal{E} - \mathcal{E}_T$ and $f \in C_T(e)$ such that $e \prec f$. By Remark 3.1, the tree $S = T - \{f\} \cup \{e\} \prec T$, since $e \prec f$ which contradicts our hypothesis and therefore e cannot dominate any edge $f \in C_T(e)$. □

The reciprocal of the above theorem is not valid. For example, in Figure 3.6(a) it is represented a network which has a unique efficient tree $T_{ef} = \{e_{12}, e_{23}, e_{24}\}$, Figure 3.6(b). Now, although edge e_{14} does not dominate any edge of tree $T = \{e_{12}, e_{23}, e_{34}\}$, Figure 3.6(c), T is not efficient.

Theorem 3.5. *If T is a spanning tree such that for all edges e of $\mathcal{E} - \mathcal{E}_T$ and for all edges f of $C_T(e) - \{e\}$ we have $f \prec e$ then T is the unique efficient spanning tree.*

Proof. Let S_0 be a spanning tree, $\mathcal{E}_{S_0} - \mathcal{E}_T \neq \emptyset$ (if it was the empty set then $S_0 = T$) and e an edge in $\mathcal{E}_{S_0} - \mathcal{E}_T$. If we remove e from S_0, we obtain two subtrees of S_0: S_0' and S_0''. On the other hand, we know that e is dominated by all edges in the path $C_T(e) - \{e\}$ and $S_0' \cup S_0'' \cup C_T(e) - \{e\}$ will be connected again (not necessarily a tree). If e' is the edge that connects S_0' with S_0'' and $S_1 = S_0 - \{e\} \cup \{e'\}$ is the tree obtained by removing e from S_0 and adding e', then $\mathcal{W}(S_1) = \mathcal{W}(S_0) - \mathcal{Z}(e) + \mathcal{Z}(e')$. Now, since by hypothesis $e' \prec e$ then for all $i \in M$ it verifies $z_i(e) \leq z_i(e')$ and exist $j \in M$ such that $z_j(e) < z_j(e')$, which implies that $w_i(S_1) \leq w_i(S_0)$ for all $i \in M$ and exist $j \in M$ such that $w_j(S_1) < w_j(S_0)$ and, therefore, $S_1 \prec S_0$. This process can be repeated while $S_i - T \neq \emptyset$, obtaining a sequence of trees such that $S_k \prec \cdots \prec S_2 \prec S_1 \prec S_0$, with

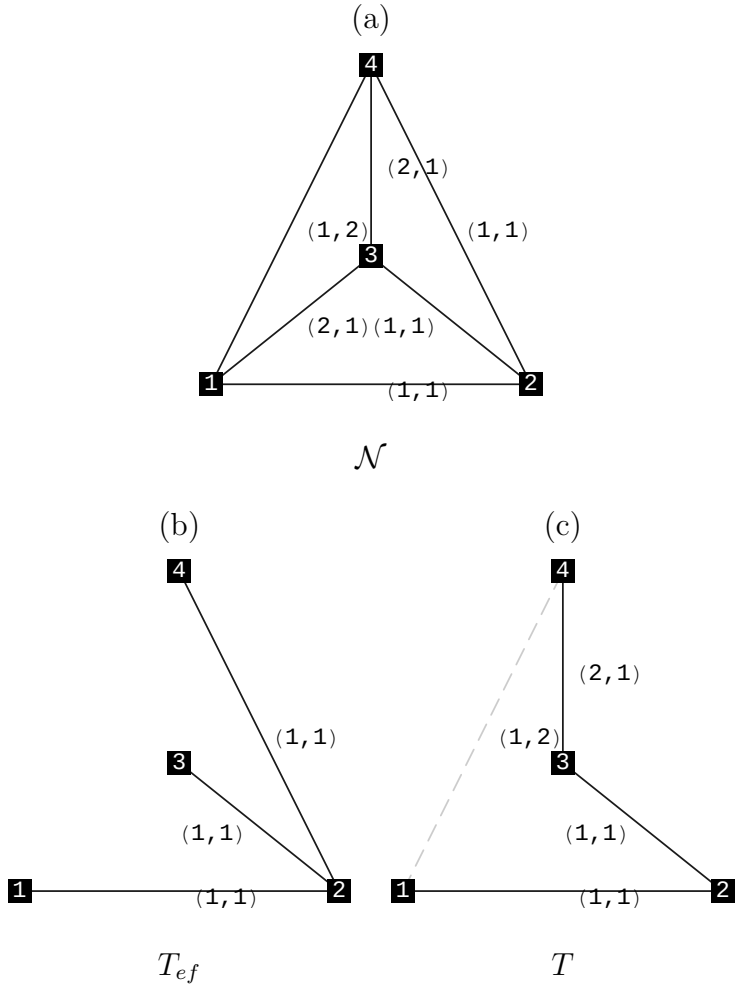


Figure 3.6: Network example.

$k \leq n-1$. At the end $S_k = T$, since $S_k - T = \emptyset$, which implies that any tree is dominated by T. □

The global efficiency of the edges can be defined as follows. If $\{\mathcal{V}_1, \mathcal{V}_2\}$ is a cut and

$$\widehat{\mathcal{E}}(\mathcal{V}_1, \mathcal{V}_2) = \{e \in \mathcal{E}(\mathcal{V}_1, \mathcal{V}_2) : f \nprec e, f \in \mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)\} \tag{3.16}$$

is the set of the efficient edges over $\mathcal{E}(\mathcal{V}_1, \mathcal{V}_2)$ then the union of the efficient edges for all cuts $\{\mathcal{V}_1, \mathcal{V}_2\}$ over $\mathcal{V}$, that is,

$$\mathcal{E}_{\mathcal{C}} = \bigcup_{\mathcal{V}_1 \subsetneq \mathcal{V}} \widehat{\mathcal{E}}(\mathcal{V}_1, \mathcal{V} - \mathcal{V}_1), \tag{3.17}$$

is called the **set of the efficient edges-cuts** over $\mathcal{N}$.

Using the above notion it is possible, in some particular conditions, to restrict the number of edges that can belong to an efficient tree.

Theorem 3.6. *Let $\mathcal{E}_{\mathcal{C}}$ be the set of the efficient edges-cuts over $\mathcal{N}$ and $\mathcal{N}' = (\mathcal{V}, \mathcal{E}_{\mathcal{C}}, \mathcal{Z})$ a connected network. If T is an efficient spanning tree then all the edges of T belong to $\mathcal{E}_{\mathcal{C}}$.*

Proof. Let S_0 be a spanning tree and $e \notin \mathcal{E}_{\mathcal{C}}$ an edge in S_0. If we remove e from S_0 we get two subtrees, S_0' and S_0''. On the other hand, since $\mathcal{N}'$ is connected, there exists at least one edge e' in $\mathcal{E}_{\mathcal{C}}$ connecting S_0' to S_0'' that does not belong to S_0. Therefore, e' dominates e, and $S_1 = S_0' \cup S_0'' \cup \{e'\} = S_0 - \{e\} \cup \{e'\}$ dominates S_0 (Remark 3.1). The process can be repeated while S_i has at least one edge in $\mathcal{E} - \mathcal{E}_{\mathcal{C}}$, generating on the way a sequence of dominated trees, $S_k \prec \cdots \prec S_2 \prec S_1 \prec S_0$ for some $k \leq n-1$. Therefore, all efficient spanning trees only have edges in $\mathcal{E}_{\mathcal{C}}$. Conversely, they are dominated by at least one spanning tree with edges in $\mathcal{E}_{\mathcal{C}}$. □

The reciprocal of this Theorem 3.6 is not valid. Returning to the example proposed in Figure 3.3, the set of efficient edges over $\mathcal{N}$ coincides with $\mathcal{E}$, $\mathcal{E}_{\mathcal{C}} = \mathcal{E}$, but there are trees that are no efficient.

3.5 Conclusions

In the previous sections, several results were examined as an effort to establish properties capable of enhancing the known methods or suggest new methodologies. These properties are in general based on the local and the global efficiency of the edges, allowing in some circumstances to enumerate the Pareto set.

For example, in Theorem 3.1 it is presented a necessary and sufficient condition for a tree to be efficient, based in its subtrees. Though, it is not known a general condition

that allows the addition of new edges to an efficient subtree with certainty of keeping the efficiency and obtaining all possible solution.

Another example is the Theorem 3.3 where it is presented a necessary and sufficient condition that could be used to establish an *a priori* set of edges that belong to all efficient spanning trees. However, a problem emerges when the computational effort is considered. For example, in a random weighted tree the probability a component of the weight being greater than another is $p = 0.5$. Therefore, an edge has probability 0.5^m of dominating another which, for a cut with r edges, implies that the probability of a particular edge dominates all the edges in the cut would be 0.5^{rm}. Therefore, to allow a proper use of the result

- It should exist a strong correlation between the edges weights, such that p is near to 1;
- The number of weights (m) should be very small;
- The network should be sparse such that r has also a good probability of being small; and
- A good heuristic to explore the $2^{|\mathcal{V}|} - 2$ possible cuts should be applied.

Nevertheless, this problem has several great advantages when used in the study of meta-heuristics, like

- The problem is the basis for several more interesting problems due to their practical applications;
- Adaptations from the algorithms for the single objective case (like Prim's or Kruskal's algorithms) allow obtaining fast approximations to the Pareto front;
- The existence of Brute Force methods (applicable to very small instances) allows to obtain the exact Pareto Front;

- The associated Weighted Sum problem has efficient algorithms to solve it;
- The possibility of obtaining an approximation set using a Weighted Sum strategy with weights computed as suggested by Hamacher & Ruhe [1994] for the bi-objective case, extensible to more objectives as in [Steiner & Radzik, 2003]; or
- The existence of algorithmic strategies that allow to obtain good approximations to the Pareto front through the computation of solutions in the viable region using, for example, Branch-and-Bound strategies or k-best algorithms.

When we consider deterministic processes, the two-phase methods, in [Hamacher & Ruhe, 1994; Steiner & Radzik, 2003], seem to be the best proposal. From the three alternatives for the second phase the use of Branch-and-Bound techniques, always dependent on the bounding procedure, is expected to have a high time complexity. The local search strategy can be considered similar to the k-best scheme and should have a better performance than the former one, but a high time complexity should remains.

As far as we know, none of the methods was tested with networks such that the Pareto front presents one or several large concave regions or large gaps.

However, perhaps the biggest disadvantage is the fact that we should expect them to be inefficiently or hardly adaptable to slight variations of the problem. Principally, when those modifications imply that the Weighted Sum problem has not an efficient method to solve it.

Therefore, the next chapter is devoted to the establishment of a repository of network generators, capable of producing a set of difficulties to most of the methods like anti-correlated edges weights, concave fronts and fronts with gaps. Two new Ant Colony based methods are presented, tested and, discussed with already studied instances of the Multiple Objective k-Degree Minimum Spanning Trees problem and

Multiple Objective Travelling Salesman Person, in the subsequent chapters.

3.6 Summary

This chapter has been devoted to the analysis of deterministic solutions for the Multiple Objective Minimum Spanning Trees problem. The first sections do a survey of the problem complexity and of the algorithmic solutions proposed by other authors for its resolution. Before the general absence of analytical properties, the next section introduces several results and examples that allow to better characterize the problem in study. The chapter ends with the presentation of conclusions, which suggest the need of alternative methods for the problems in study.

No amount of experimentation can ever prove me right; a single experiment can prove me wrong.

Albert Einstein

4

Archetypes of Networks Generators

Contents

4.1 Overview

The use of heuristics and meta-heuristics in the optimization processes requires that those methods have to be tested with known problems and confronted with their solutions, to deduct their overall performance and accuracy. The use of large sets of problems reduces the risk of over-fitting the methods although, in an extensive collection, some classes of features may be rare or absent.

Commonly, various types of difficulties arise when we try to explore some specific problem, as

- The need to find repositories for the problems in study (or possibly similar);
- The need to know the optimums or quasi-optimums of the test problems; and
- The lack of diversity of the previously studied instances.

This situation implies that the proposed algorithms are first tested with classical problems, like some libraries of functions for the continuous case or the classical Travelling Salesman Person problem for the discrete case, which is a touchstone for many meta-heuristics [Cirasella *et al.*, 2001; Dorigo *et al.*, 1996, 1999; Johnson & McGeoch, 1997].

These lack of choices is even more patent in the multiple objective combinatorial optimization, where only a few disperse sets of results are available. Therefore, this

section proposes and presents the Multiple Objective Spanning Tree repository – MOST – Project [Cardoso *et al.*, 2006a, b]. It is an archive for the large multidisciplinary applications problems of the multiple objective spanning trees. The objective of the MOST Project is to establish, maintain, and successively improve an intuitive and large set of problems along with their optimal or quasi-optimal solutions, on which different algorithms can quickly be tested and classified relatively to their performance: processing time and accuracy.

MOST can be accessed via Internet at http://est.ualg.pt/adec/csc/most.

In the MOST Project problems, are classified according to three types of generators: nodes, edges and weights.

There are five types of nodes generators in consideration:

- Grid Nodes Generator – produces a lattice regular cloud of nodes;
- Triangular Nodes Generator – produces a triangular regular cloud of nodes;
- Uniform and Normal Nodes Generators – are instance cases of the Statistical Nodes Generator, which uses random generators associated to statistical distributions. As deducted from their names, the uniform and normal distribution are used in this cases; and
- Clusters Nodes Generator – where clusters of nodes are set using combinations of the previous generators.

Reporting to the edges classes, six types of generators are proposed:

- Grid Edges Generator – applied over the Grid Nodes Generator produces a Manhattan-like network;
- Delaunay Edges Generator – uses the Delaunay triangulation to set the edges. A particular case is set when applied to the Triangular Nodes Generator, called

the Triangular Edges Generator;

- k-Convex Edges Generator – uses an onion triangulation to set the edges;
- Complete Edges Generator – returns a complete network;
- Voronoi Edges Generator – uses the Voronoi diagram to set the edges and has a particular case when applied to the Triangular Nodes Generator, called the Hexagonal Edges Generator; and
- Clusters Edges Generator – uses some of the previous generators to set the edges between and in the clusters.

For the edges weights case it were considered three generators:

- Random Weights Generator – randomly sets the weights;
- ρ-Correlated Weights Generator – uses a parameter ρ to define the weights such that the their correlation is equal to ρ; and
- Concave Weights Generator – defines the weights such that the Pareto front has one or more large concave regions.

Therefore, this chapter is divided as follows. The next three sections describe the methods that were used to generate the networks. The remaining sections present some examples of the Pareto fronts, obtained with the generated networks and a summary of the chapter.

4.2 Nodes Generators

The reproduction of practical problems is one of the main objectives of the MOST Project. After a previous analysis, it was decided to start by considering five types of

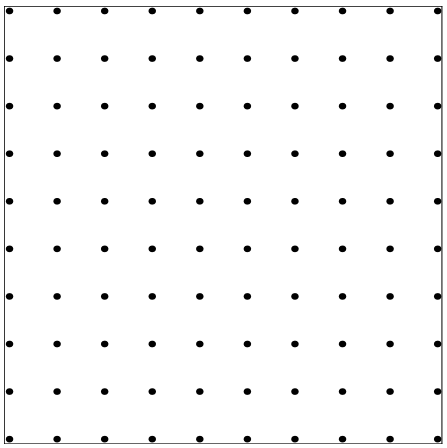


Figure 4.1: Graphical representation of the nodes distribution obtained with the $GNG_{10,10}$.

nodes generators: Grid, Triangular, Uniform, Normal and Cluster Nodes Generator. The first two generators can be classified as regular since they return regularly distributed clouds of nodes. The third and forth cases, are instances of a generator that uses pseudo-random numbers associated to statistical distributions. In the fifth case, to create clusters of nodes, the previous generators are employed, first to calculate a set of centres for the clusters, followed by the generation of a set of nodes located in their neighbourhood. Next, it is given a more detailed description of each one of those methods.

4.2.1 Grid Nodes Generator (GNG)

The simplest case is the Grid Node Generator. This case produces a set of nodes that are distributed over a $m \times n$ lattice. To simplify, $GNG_{m,n}$ will represent an instance of that generator, with m lines and n columns of nodes,

$$\mathcal{V} = GNG_{m,n} = \{(x, y) : x \in \{1, 2, \ldots, n\}, y \in \{1, 2, \ldots, m\}\}.$$

In Figure 4.1 we can see an example of a $GNG_{10,10}$.

Algorithm 8 Triangular Nodes Generator algorithm.

input: m, n, l, $h = \frac{l\sqrt{3}}{2}$ ▷ *m, n – Number of lines and number of nodes by line; l – distance between neighbour nodes; h - distance between lines (see Figure 4.2(a))*

output: $\mathcal{V}$ ▷ *set of nodes*

1: $\mathcal{V} \leftarrow \emptyset$
2: **for all** $(x, y) \in \{1, 2, ..., n\} \times \{1, 2, ..., m\}$ **do**
3: **if** x is odd **then**
4: $\mathcal{V} \leftarrow \mathcal{V} \cup \{(x \times l, y \times h)\}$
5: **else**
6: $\mathcal{V} \leftarrow \mathcal{V} \cup \{(x \times l + \frac{l}{2}, y \times h)\}$
7: **end if**
8: **end for**
9: **return** $\mathcal{V}$

4.2.2 Triangular Nodes Generator (TNG)

Similar to the GNG case, the Triangle Nodes Generator, sets m lines with n nodes per line, $TNG_{m,n}$, distributed according with Algorithm 8. Figure 4.2 depicts the nodes distribution for a $TNG_{10,10}$.

4.2.3 Statistical Nodes Generator (UNG, NNG)

The Statistical Nodes Generator uses random number generators defined by two statistical distributions, Z_X, Z_Y, (respectively associated to x and y coordinates), to yield as much nodes as necessary. In Algorithm 9 it is made a description of the process, where $random(Z)$ is a function that returns a (pseudo) random number with specified distribution Z.

We considered two instances of the Statistical Nodes Generator, obtained with the uniform and the normal distributions.

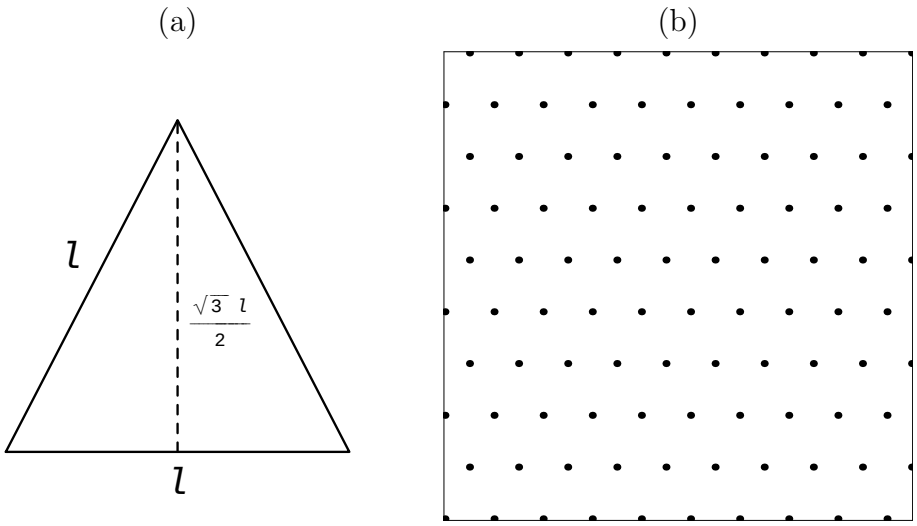


Figure 4.2: (a) Height of an equilateral triangle with side l. (b) Graphical representation of nodes distribution obtained with $TNG_{10,10}$.

Algorithm 9 Statistical Nodes Generator.

input: Z_X, Z_Y, n ▷ *Z_X, Z_Y – statistical distributions; n – number of nodes*
output: $\mathcal{V}$ ▷ *set of nodes*

1: $\mathcal{V} \leftarrow \emptyset$
2: **for** $i = 1, 2, ..., n$ **do**
3: $\quad \mathcal{V} \leftarrow \mathcal{V} \cup \{(random(Z_X), random(Z_Y))\}$
4: **end for**
5: **return** $\mathcal{V}$

Uniform Nodes Generator (UNG)

As mentioned above, the Uniform Nodes Generator is a particular case of the Statistical Nodes Generator, and UNG_n represents an instance of UNG with n nodes. In this case, $Z_X \sim U[x_{min}, x_{max}]$ and $Z_Y \sim U[y_{min}, y_{max}]$, where $U[a, b]$ is the uniform distribution over the interval $[a, b]$, and $x_{min}, x_{max}, y_{min}$, and y_{max} are generator parameters. This produces a cloud of nodes uniformly distributed over the rectangle $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$.

In Figure 4.3(a) we can see an example of the distribution for $n = 1000$ nodes, UNG_{1000}, with $x_{min} = y_{min} = 0$ and $x_{max} = y_{max} = 1000$.

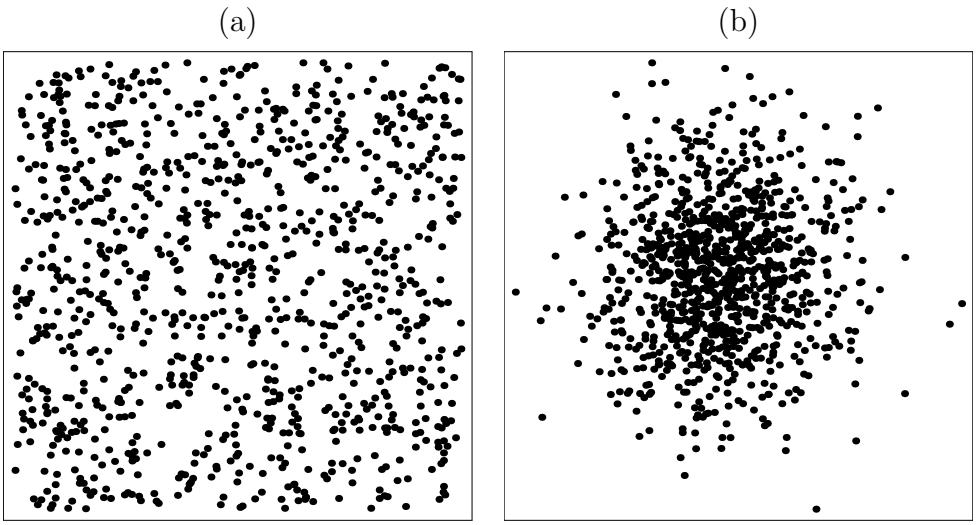


Figure 4.3: Graphical representation of nodes distribution for: (a) UNG_{1000}; and (b) NNG_{1000}.

Normal Nodes Generator (NNG)

The Normal Nodes Generator is also a particular case of the Statistical Nodes Generator and NNG_n represents an instance with n nodes. In this case, $Z_X \sim N(\mu_X, \sigma_X)$ and $Z_Y \sim N(\mu_Y, \sigma_Y)$, where $N(\mu, \sigma)$ is the normal distribution with mean μ and standard deviation σ and μ_X, μ_Y, σ_X and σ_Y are generator parameters. The cloud of nodes is distributed around (μ_X, μ_Y) with horizontal and vertical deviations associated to the values of σ_X and σ_Y, respectively.

The cloud of nodes for NNG_{1000}, with $\mu_x = \mu_y = 10000$ and $\sigma_x = \sigma_y = 100$, is sketched In Figure 4.3(b).

4.2.4 Clusters Nodes Generator ($ClNG$)

The last nodes generator to be considered is the Clusters Nodes Generator. The $ClNG$ uses, in two steps, the previous generators to produce a set of m clusters with n nodes each. In the first phase, the m clusters centres are generated, $c = (c_x, c_y)$, around which, in the second phase, the clusters are completed by the addition of the remaining $n - 1$

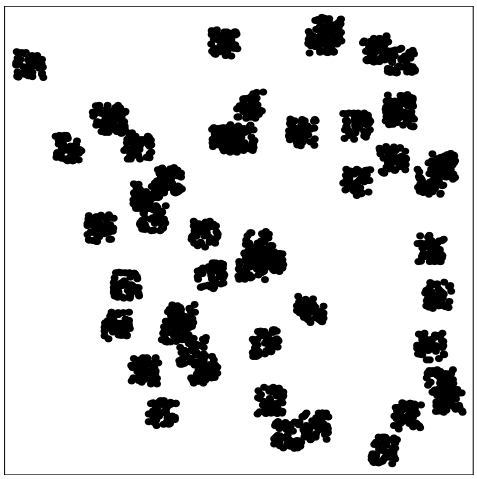


Figure 4.4: Graphical representation of nodes distribution for a $ClNG_{20,50}$.

nodes. An instance with m clusters of n nodes each will be represented by $ClNG_{m,n}$. In this case, that obviously depends on the centres location and the dimensions of the clusters, we usually obtain a set of separated sub-clouds of nodes.

In Figure 4.4 we can see an example of 50 clusters with 50 nodes each, $ClNG_{50,50}$. We used a UNG_{50} to generate the centres with $x_{min} = y_{min} = 0$ and $x_{max} = y_{max} = 1000$. Then each cluster was completed using a UNG_{49}, in the neighbourhood of the centres, defined by the parameters $x_{min} = c_x - \delta_x$, $x_{max} = c_x + \delta_x$, $y_{min} = c_y - \delta_y$ and $y_{max} = c_y + \delta_y$, where $\delta_x = \delta_y = \sqrt{1000}$. As a results, the number of nodes is perfectly balanced between the clusters. Nevertheless, others combinations of the generators can produce some not so well balanced cases.

4.3 Edges Generators

This section describes the six methods used to generate the networks edges, $\mathcal{E}$. The first, the Grid Edges Generator, is only applicable to GNG instances and returns a Manhattan-like topology. The next two are called Delaunay and k-Convex Edge Generator since they use a Delaunay triangulation and an onion triangulation. The

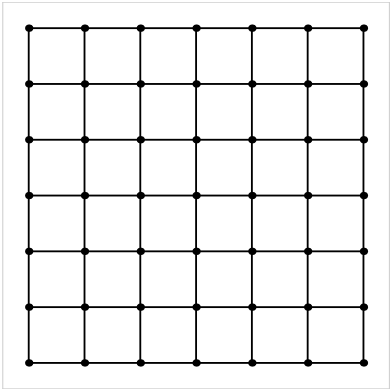


Figure 4.5: Graphical representation of a network obtained with $GEG_{7,7}$.

Fourth section considers the complete networks case. It follows the definition of the Voronoi Edges Generator, based on the Voronoi diagram, which has the particularity of replacing the set of nodes by the nodes induced by the diagram. Finally, the Cluster Edge Generator uses a set of nodes obtained with the $ClNG$ and uses some of the previous methods to generate the edges.

4.3.1 Grid Edges Generator (GEG)

The Grid Edges Generator uses the nodes obtained with the GNG. An edge e_{pq} defined by nodes p and q with coordinates (p_x, p_y) and (q_x, q_y), respectively, belongs to $\mathcal{E}$ if

$$(|p_x - q_x| = 1 \wedge p_y = q_y) \vee (p_x = q_x \wedge |p_y - q_y| = 1)\,. \tag{4.1}$$

This network topology appears, for instance, in Manhattan-like cities networks and VLSI circuits problems where often the components must be connected over a grid. Figure 4.5 sketches an example of GEG over a $GNG_{7,7}$.

4.3.2 Delaunay Edges Generator (DEG)

The Delaunay Edges Generator uses a Delaunay triangulation to define the edges. Starting from a set of nodes $\mathcal{V}$, the Delaunay triangulation is the decomposition of the

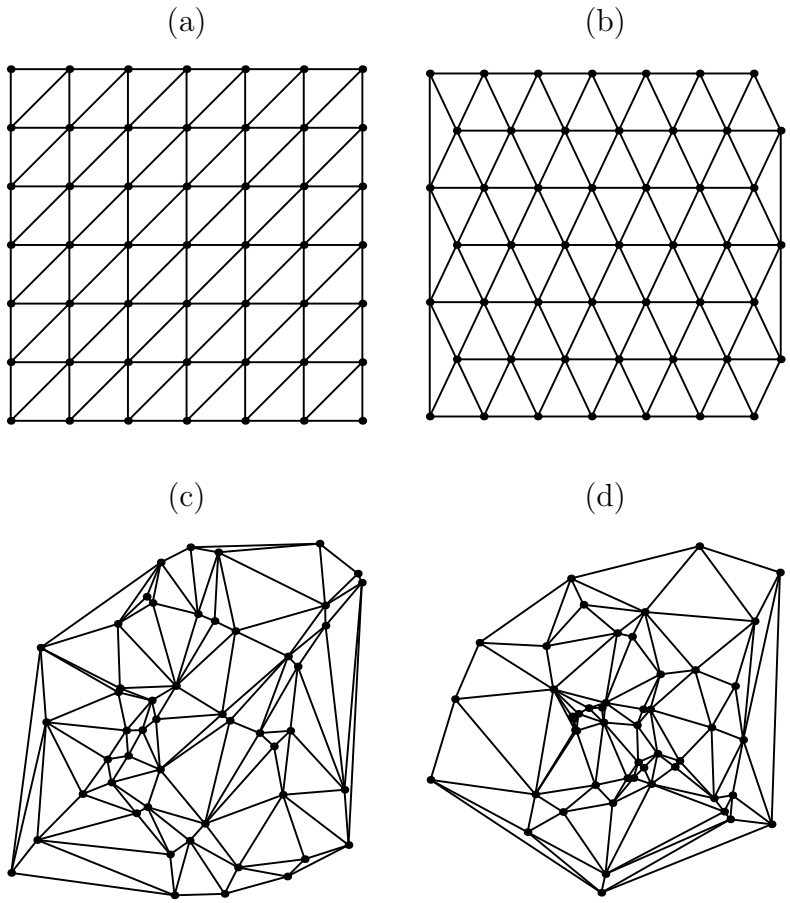


Figure 4.6: Graphical representation of networks obtained with the DEG from the set of nodes obtained with: (a) $GND_{7\times7}$; (b) $TNG_{7\times7}$; (c) UNG_{50}; and (d) NNG_{50}.

convex-hull defined by $\mathcal{V}$ in triangles, such that two nodes are connected if and only if they lie on a circle whose interior contains no other node of $\mathcal{V}$ (see for example [de Berg *et al.*, 1997]). This triangulation maximizes the minimum angle of the triangles that compose it. It is also known that the Euclidean minimum spanning tree of a set of nodes is a subset of the Delaunay triangulation for these same set of nodes.

In Figure 4.6(a)-(d) we can see four networks obtained with a $GNG_{7\times7}$, a $TNG_{7\times7}$, a UNG_{50} and a NNG_{50}, respectively. It is obvious that for the first two cases the triangulation is not unique whether, for the other two cases, depends on the general position of the nodes (no three nodes are in the same line and no four in the same

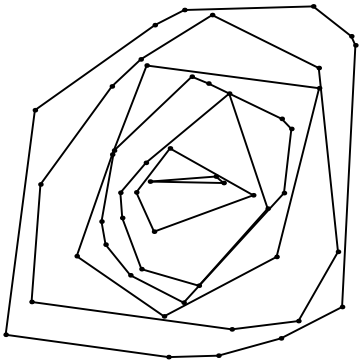


Figure 4.7: k-convex hulls or onion peeling of a UNG_{50}.

circle).

Triangle Edges Generator (TEG)

The Triangle Edges Generator is a particular case of DEG when applied to the TNG. An example of the TEG for a $TNG_{7,7}$ is sketched in Figure 4.6(b).

4.3.3 k-Convex Edges Generator ($k-CEG$)

A convex hull of a set of nodes is the smallest convex set that contains those nodes (see for example [Abellanas *et al.*, 1996; de Berg *et al.*, 1997] for more details). If P is a set of points in the plane, the **onion peeling** or **onion layer** of this set can be computed as follows. Let H_0 be the convex hull of P. Now remove the points on the boundary of H_0 from P and let H_1 be the convex hull of what remains in P. Iteratively repeat the process until no more points are left in P. Figure 4.7 depictes the resulting k-convex hulls for a UNG_{50}.

The onion peeling has been studied in areas like computational geometry [Cortés *et al.*, 2005; Sack & Urrutia, 2000], search algorithms [Chang *et al.*, 2000] and pattern recognition [Fadili *et al.*, 2004; Poulos *et al.*, 2004].

Any triangulation that includes the k-convex hulls, is called a onion triangulation.

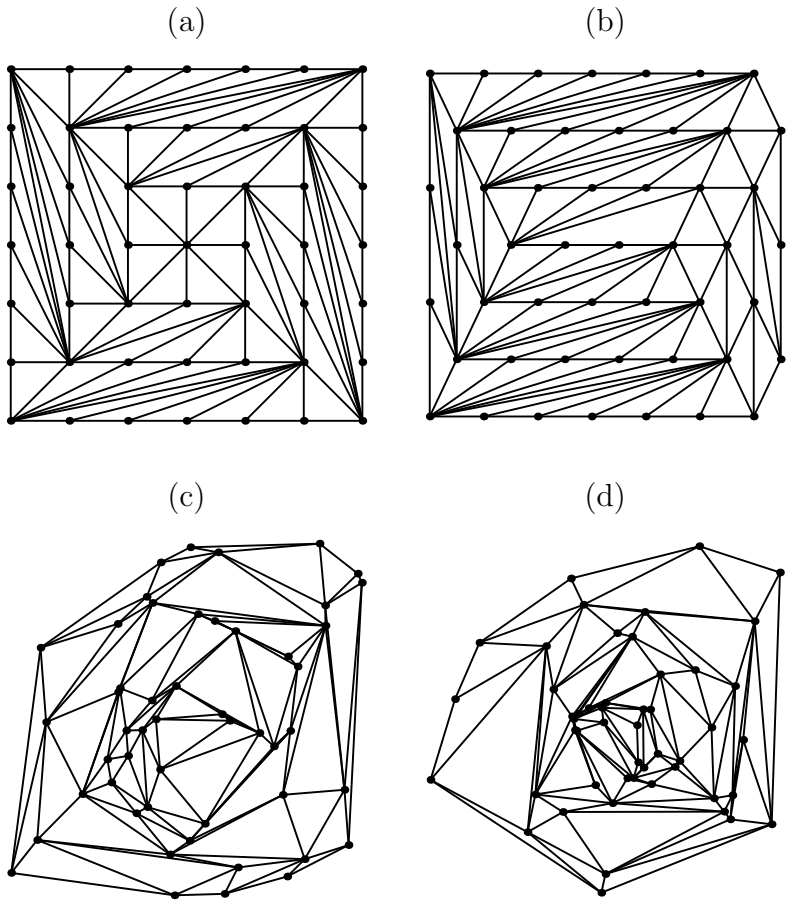


Figure 4.8: Graphical representation of networks obtained with $k - CEG$ for nodes generated with: (a) $GNG_{7,7}$; (b) $TNG_{7,7}$; (c) UNG_{50}; and (d) NNG_{50}.

In particular, a very simple and elegant method supported on rotating callipers was used to complete the triangulation[Pirzadeh, 1999; Toussaint, 1983].

The $k - CEG$ uses the onion triangulation, described above, to naturally define the edges of the network. In Figure 4.8 we can see the graphical representation of four networks obtained with $k - CEG$.

This k-hull kind of topology can also be found in many city traffics networks.

4.3.4 Complete Edges Generator (CEG)

As the name suggests the Complete Edges Generator defines the $\binom{|\mathcal{V}|}{2}$ edges between each pair of nodes. Most of the real world problems do not allow every node to be connected to each other. However, in a more academic point of view, it is usual to consider and test this kind of networks as a worse case scenario [Knowles & Corne, 2001a].

4.3.5 Voronoi Edges Generator (VEG)

The Voronoi Edges Generator uses the Voronoi diagram, dual of the Delaunay triangulation, to generate a network (see for example [de Berg *et al.*, 1997; Sack & Urrutia, 2000]). In this case, any of the previous nodes generators can be used to start the process. However, as exampled in Figure 4.9(a), the final set of nodes does not coincide with the starting set of nodes $\mathcal{V}$, since the edges of the Voronoi diagram are not directly defined by $\mathcal{V}$. Therefore, $\mathcal{V}$ is replaced by the set of nodes induced by the Voronoi diagram. Note that this set has not necessarily the same cardinality of the original $\mathcal{V}$.

Figures 4.9(b)-(d) show examples of networks obtained with a UNG_{50} and a NNG_{50}. The Voronoi diagram is associated to problems like the Post Office problem [Knuth, 1973] and this topology pattern is very often found as the frontiers of geopolitical maps.

Hexagonal Edges Generator (HEG)

The Hexagonal Edges Generator is a particular case of VEG when applied to a TNG. In Figure 4.9(d) we can see an example for a $TNG_{7,7}$.

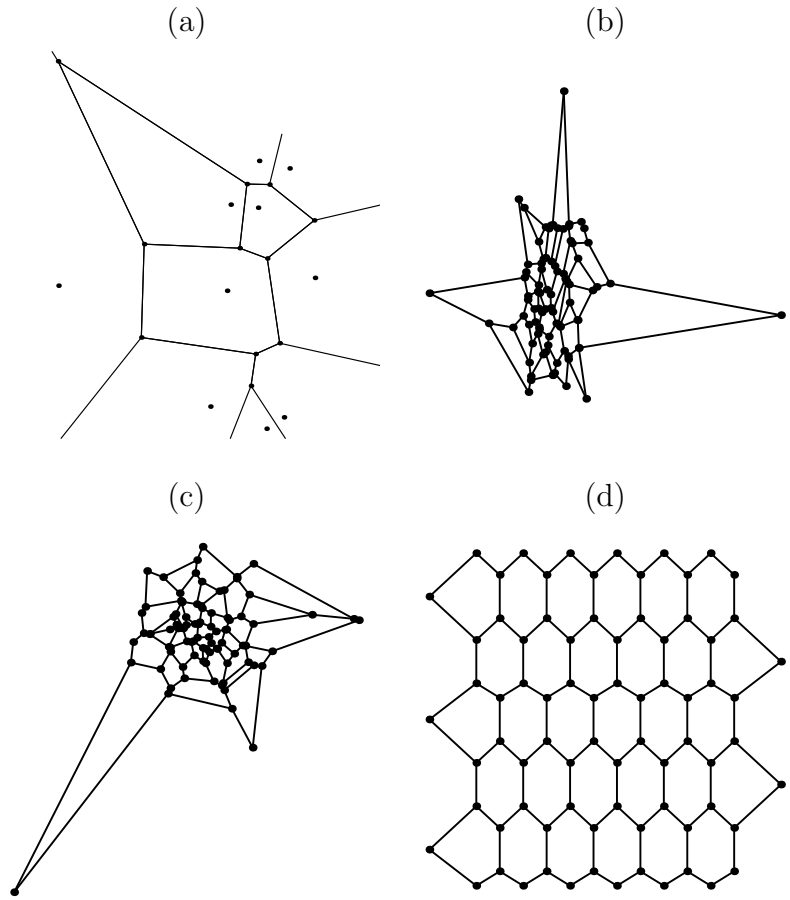


Figure 4.9: (a) Voronoi diagram for a set of ten points; Graphical representation of the networks obtained with the Voronoi Generator for: (b) NNG_{50}; (c) UNG_{50}; and (d) $TNG_{7,7}$.

4.3.6 Clusters Edges Generator ($ClEG$)

The Clusters Edges Generator uses the previous edges generators in two phases. First, the centre nodes of the clusters are connected (see Section 4.2.4). Then, the process is repeated over the nodes of each cluster, including their respective centres. In Figure 4.10 we can see two examples for the same set of nodes, $ClNG_{10,15}$. In Figure 4.10(a) the DEG was used to connect the centres and the nodes of the clusters. In the case of Figure 4.10(b) it was adopted the $k-CEG$ to connect the centres and the CEG for the clusters. It is possible to identify real applications where this topology appears,

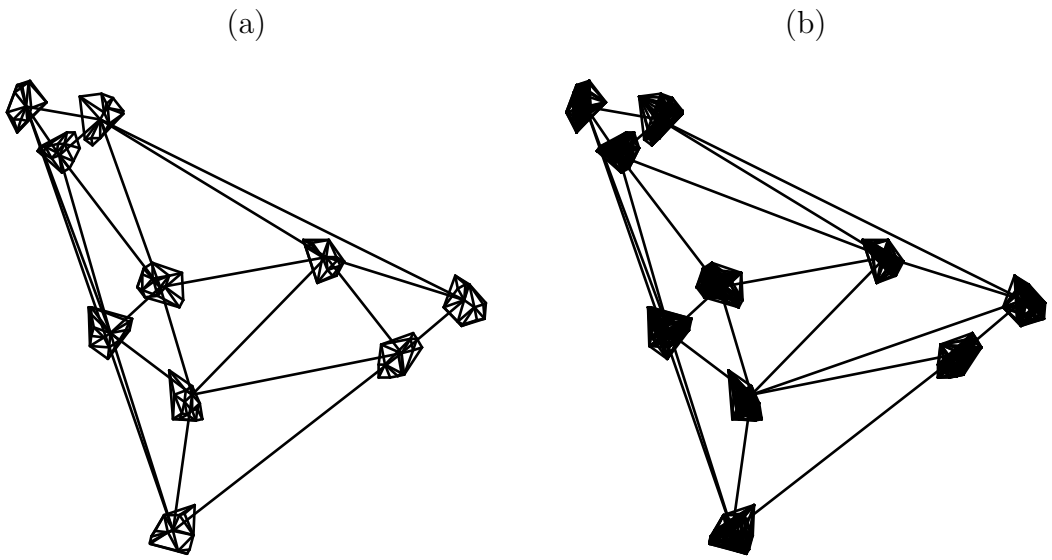


Figure 4.10: Graphical representation of networks obtained with the $ClEG$ for $ClNG_{10,15}$ with: (a) DEG to connect the centres and the clusters nodes; (b) $k-CEG$ to connect the centres and CEG to generate the edges for the clusters.

like power, phone or cable distribution networks and country traffic maps, where each cluster may represent a village and each village is connected to neighbour villages.

4.4 Weights Generators

Another important aspect of the networks are the edges weights. In this section three types of weights generators are presented:

- The Random Weights Generator – is important, for instance, in the study of the non Euclidean networks;
- The ρ–Correlated Weights Generator – originates networks with interesting characteristics, like the fact that the Pareto front varies from a large cardinality when $\rho \approx -1$ (the graphical representation of the Pareto front for bi-objectives problems assumes an almost straight line shape), to a very small cardinality when

$\rho \approx 1$;

- The Concave Weights Generator – has another very important characteristic: it can produce large concave Pareto fronts, known to be hard to tackle by classical algorithms like the Weighted Sum.

4.4.1 Random Weights Generator (RWG)

The Random Weights Generator sets the weights using a discrete integer uniform random distribution over a predefined set. Figure 4.11(a) sketches the cloud of weights produced by the RWG for a complete network with 50 nodes.

4.4.2 $\rho-$Correlated Weights Generator ($\rho - CWG$)

In this case, the objective is to set the edges weights, so that the correlation between them is some predefined value $\rho \in [-1, 1]$. Except for the cases where the edges where generated using GEG or TEG, we considered the first weight of the edge e as the (integer) Euclidean length of e. For the exceptions, the first weight is a random integer in $[1, M_d]$, where M_d is the maximum Euclidean distance between every par of nodes. This two cases have this special treatment since they are regular networks, which implies that the problem could just be reduced to the single objective case (the first weight would be equal in all edges).

Algorithm 10 describes the process that is used to establish the weights vector, where $|e|$ is the Euclidean length of e and $random(U[-1, 1])$ is a function that returns a (pseudo) random number with uniform distribution in the interval $[-1, 1]$. The values of Y are subject to a normalization process called in line 4. To do this, Y (that ranges between $[-1, 1]$ after line 3) is replaced by $aY + b$, where $p(x) = ax + b$ is the interpolating polynomial of points $\{(\min(Y), m), (\max(Y), M)\}$ and $[m, M]$ is the

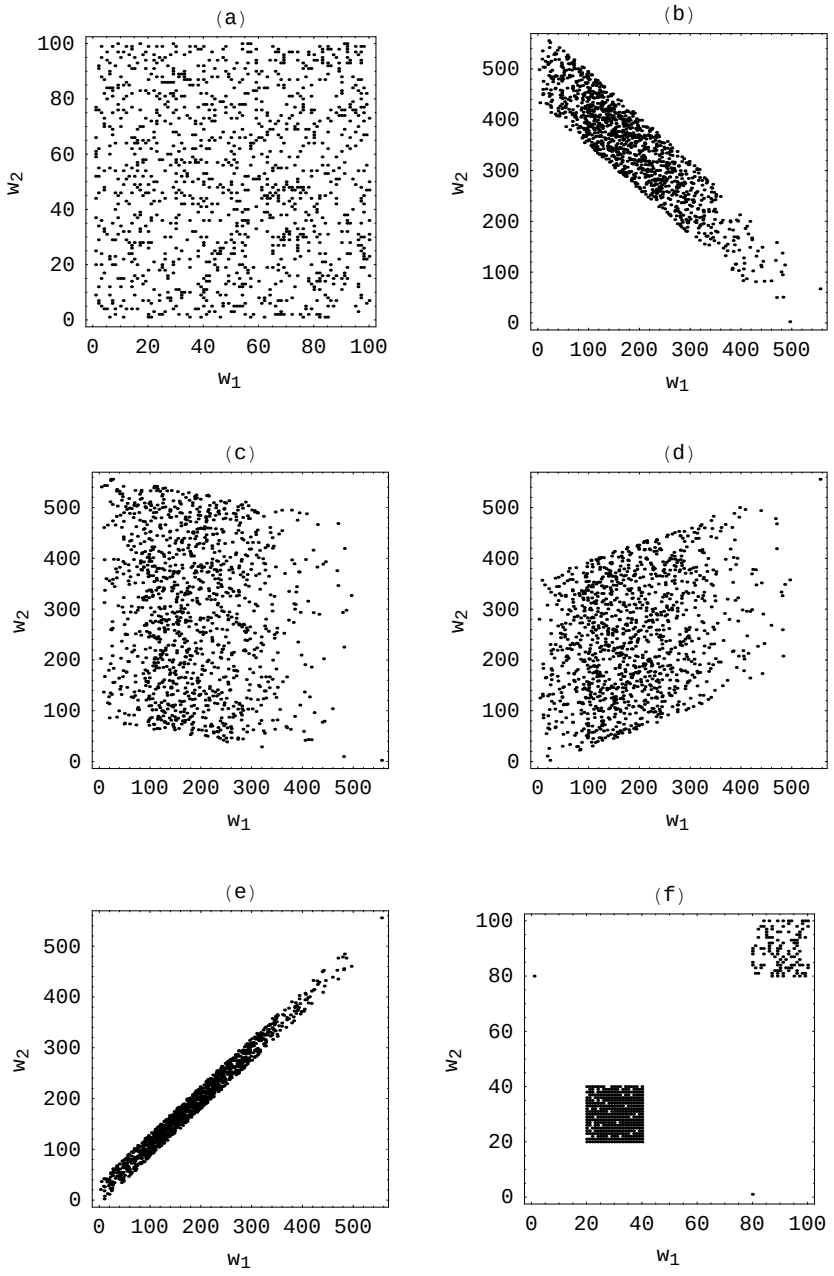


Figure 4.11: Cloud of weights for a complete network of 50 nodes using: (a) RWG; (b) $(-0.9) - CWG$; (c) $(-0.1) - CWG$; (d) $(0.3) - CWG$; (e) $(0.99) - CWG$ and (f) $CWG_{5,10,100}$.

interval where the values should range. Note also that the correlation between X and Y is equal to the correlation between X and $aY + b$ for $a \neq 0$.

Algorithm 10 Setting edges with ρ-correlated weights

input: $\rho, \mathcal{E} = \{e_1, e_2, \ldots, e_p\}$ ▷ ρ - *desired correlation*

output: $\mathcal{Z}$ ▷ $\mathcal{Z} : \mathcal{E} \rightarrow \mathbb{R}^m$

1: Set $W \leftarrow \begin{bmatrix} 1 & |e_1| & random(U[-1,1]) \\ 1 & |e_2| & random(U[-1,1]) \\ & \vdots & \\ 1 & |e_p| & random(U[-1,1]) \end{bmatrix}$ and $X \leftarrow \begin{bmatrix} |e_1| \\ |e_2| \\ \vdots \\ |e_p| \end{bmatrix}$

2: Do the QR decompositions of W: $W = QR$

3: Set $Y \leftarrow \begin{bmatrix} 1 & \rho & \sqrt{1-\rho^2} \end{bmatrix} \times Q^T$

4: Normalize Y

5: Set $\forall_{e_i \in \mathcal{E}} : z_1(e_i) \leftarrow \lfloor X_i \rfloor$

6: Set $\forall_{e_i \in \mathcal{E}} : z_2(e_i) \leftarrow \lfloor Y_i \rfloor$

Four examples of the clouds of weights obtained with ρ equal to -0.9, -0.1, 0.3 and 0.99 for a complete network with 50 nodes are presented In Figure 4.11(b)-(e).

4.4.3 Concave Weights Generator (CWG)

In the Concave Weights Generator, the weights are generated to produce a large concave Pareto front. This case is an adaptation from the generator presented by Knowles & Corne [2001a] for the k-Degree Minimum Spanning Trees problem.

This generator needs two small integers ξ and η, such that $\xi < \eta \ll M - \xi$ (where M is the maximum admitted weight), and three special nodes n_1, n_2, n_3. These nodes have an important role in the concave front and should have a degree as large as possible. Heuristically, n_1, n_2 and n_3 are defined as follows: the selection process starts by setting n_1 as the node with the highest degree. The second node, n_2, is chosen from the nodes with highest degree adjacent to node n_1. From the remaining nodes, if there are nodes adjacent simultaneously to n_1 and n_2 it is chosen the node with highest degree to be n_3. Otherwise, the same choice is made from the nodes adjacent to n_1 or n_2. The existence of candidate nodes with the same degree leads to a random selection between them. The weights of the edges $e \in \mathcal{E}$ are set according to Algorithm 11.

Algorithm 11 Setting edges weights so that the Pareto set has a large concave front.

input: $e_{ij} \in \mathcal{E}$, M, ξ, η, $N = \{n_1, n_2, n_3\}$, ▷ *e_{ij} is the edge defined by nodes i and j; M is the maximum weight; ξ and η are small integers, such that $\xi < \eta \ll M - \xi$; n_1, n_2, n_3 are the special nodes(see Section 4.4.3); and $U_d(a, b)$ - discrete integer uniform over $\{a, a+1, \ldots, b\}$.*

output: $z(e_{ij}) = (z_1, z_2)$ ▷ *e_{ij} weight vector*

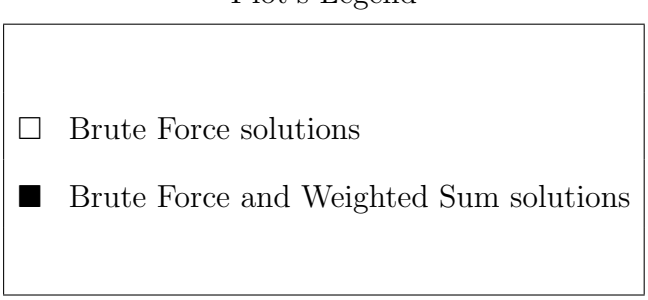
1: **if** $i, j \notin N$ **then return** $z(e_{ij}) = (U_d(\xi, \eta), U_d(\xi, \eta))$
2: **end if**
3: **if** $i \in N \overset{\bullet}{\vee} j \in N$ **then return** $z(e_{ij}) = (U_d(M - \xi, M), U_d(M - \xi, M))$
4: **end if**
5: **if** $i = n_1 \wedge j = n_2$ **then return** $z(e_{ij}) = (\xi, \xi)$
6: **end if**
7: **if** $i = n_1 \wedge j = n_3$ **then return** $z(e_{ij}) = (1, M - \xi)$
8: **end if**
9: **if** $i = n_2 \wedge j = n_3$ **then return** $z(e_{ij}) = (M - \xi, 1)$
10: **end if**

Figure 4.11(f) sketches the cloud of weights obtained for a complete network with 50 nodes, $\xi = 20$, $\eta = 40$, and $M = 100$.

4.5 Examples

In Figure 4.12 we can see the graphical representations of the exact Pareto fronts, solutions of the Multiple Objectives Minimum Spanning Trees problem, for thirty examples of networks (non-filled boxes). Simultaneously, it is represented the set of solutions obtained with 1000 runs of the Weighted Sum method (filled boxes) for the weights following parameters in

$$\Lambda = \left\{ (l_1, l_2) : l_1 = \frac{k}{1001} \wedge k \in \{1, 2, \ldots, 1000\} \wedge l_2 = 1 - l_1 \right\}.$$



The exact fronts were obtained using a Brute Force method implemented in C++ and the tests were run on a set of LINUX OS workstations with Intel Pentium^{TM}III $533Mhz$ processors and $128Mb$ of RAM. Due to the time requirements, inherent to the Brute Force algorithm's complexity, the use of this method was restricted to the cases with less than 10^{11} distinct networks.

The identification of the problems is implicit in the name that is written in the top of each figure, and follows the sequence: $[< nodes\ generator >]< |\mathcal{V}| >[< edges\ generators >]< |\mathcal{E}| >[< weights\ generator >] < parameters > [NST] < number\ of\ spanning\ trees > .net$. For example,

$$[ClNG]100[ClEG][DEG][k-CEG]227[ro-CWG]0.5[NST]49.net$$

is a network with 100 nodes and 227 edges generated using the $ClNG$ for the nodes and $ClEG$ with DEG and $k-CEG$ to connect inter and intra clusters, respectively. The weights were set using the $0.5-CWG$ and the number of distinct spanning trees as an order of magnitude of 10^{49}.

In the sketches of Figure 4.12 we can observe some of the characteristics previously referred, as fronts with large gaps (examples: i, ii, iv, v , vi, ix, x, xii, xii, xiv, xvi, xviii, xx, xxii, xxiv, xxv, xxvi, xvii, xxx), fronts with concave regions (examples: i, ii, vi, vii, vii, ix, xi, xvi, xvii, xviii, xx, xxii, xxiv, xxv, xxix), and anti-correlated fronts (examples: xv, xxi, xxiii). It is also observable that, in some cases, the Weighted Sum

	GNG	TNG	UNG	NNG	$ClNG$
GEG	✓	×	×	×	×
DEG	✓	✓	✓	✓	✓
$k-CEG$	✓	✓	✓	✓	✓
CEG	✓	✓	✓	✓	✓
VEG	✓	✓	✓	✓	✓
$ClEG$	×	×	×	×	✓

Table 4.1: Possible combinations of the nodes and edges generators.

method does not allow to establish representative fronts for the exact solution as in Figures 4.12 i, ii, vii, ix, xi, xv, xiv, xx, xxiii.

4.6 Summary

Before the generalized absence of libraries of problems for discrete multiple objective problems, this section has been devoted to the formalization of a framework to build a repository of networks test problems.

One of the main objectives has been to define a set of generators that would allow the generation of networks possessing, as much as possible, the same topological and edges weights characteristics, as some of the most important practical networks. Simultaneously, some more academic problems were considered, as the Complete Network Generator or the Concave Weights Generator.

Table 4.1 resumes the possible combinations of all the proposed generators. Obviously, some of the possible combinations take the meaningfulness of the sub-adjacent idea, like the use of a DEG over a $ClNG$.

The combination of the generators produces a diversified set of problems. In fact, without the possible variation in the parameters of the generator, it is possible to set

sixty-six distinct types of networks. Among all this networks there are several cases with distinct difficulties for the optimization processes.

This section ends with a set of examples of Pareto fronts obtained for the Multiple Objectives Minimum Spanning Trees problem over network instances created with those generators. The Pareto fronts were obtained using a Brute Force method for a set of small networks and it was possible to confirm some of the Weighted Sum method pitfalls.

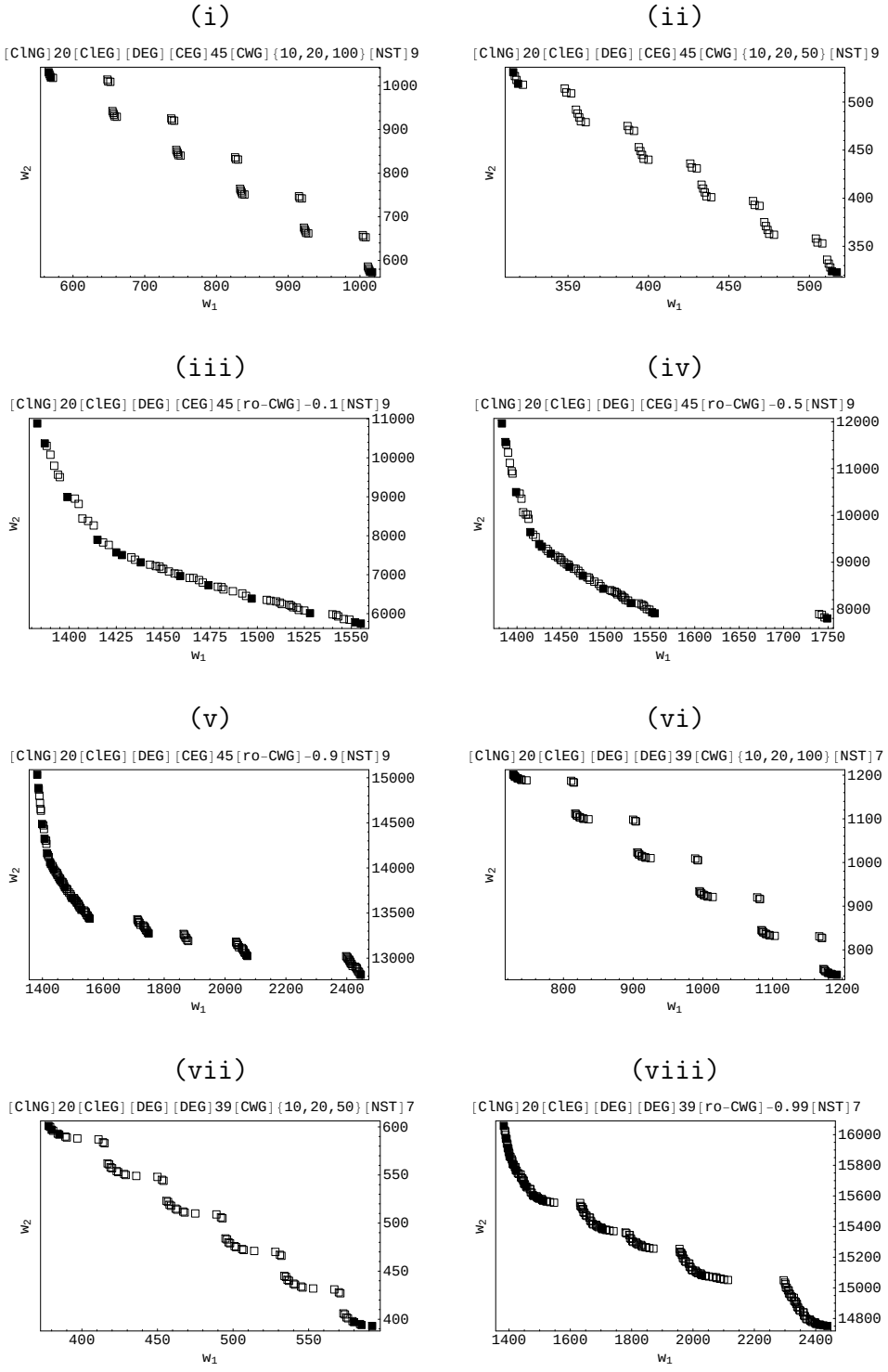


Figure 4.12: Examples of Pareto fronts

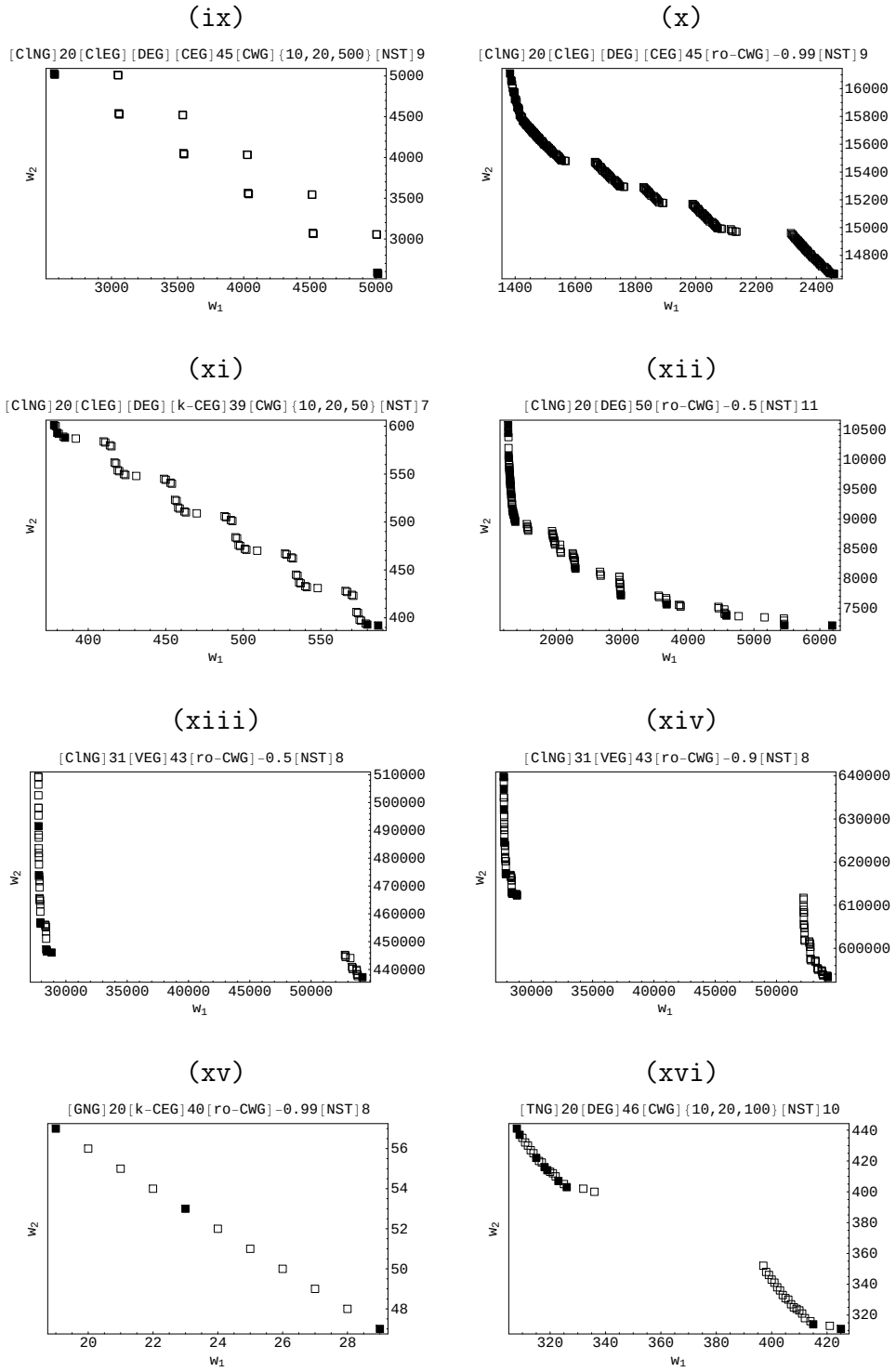


Figure 4.12: Examples of Pareto fronts (continuation)

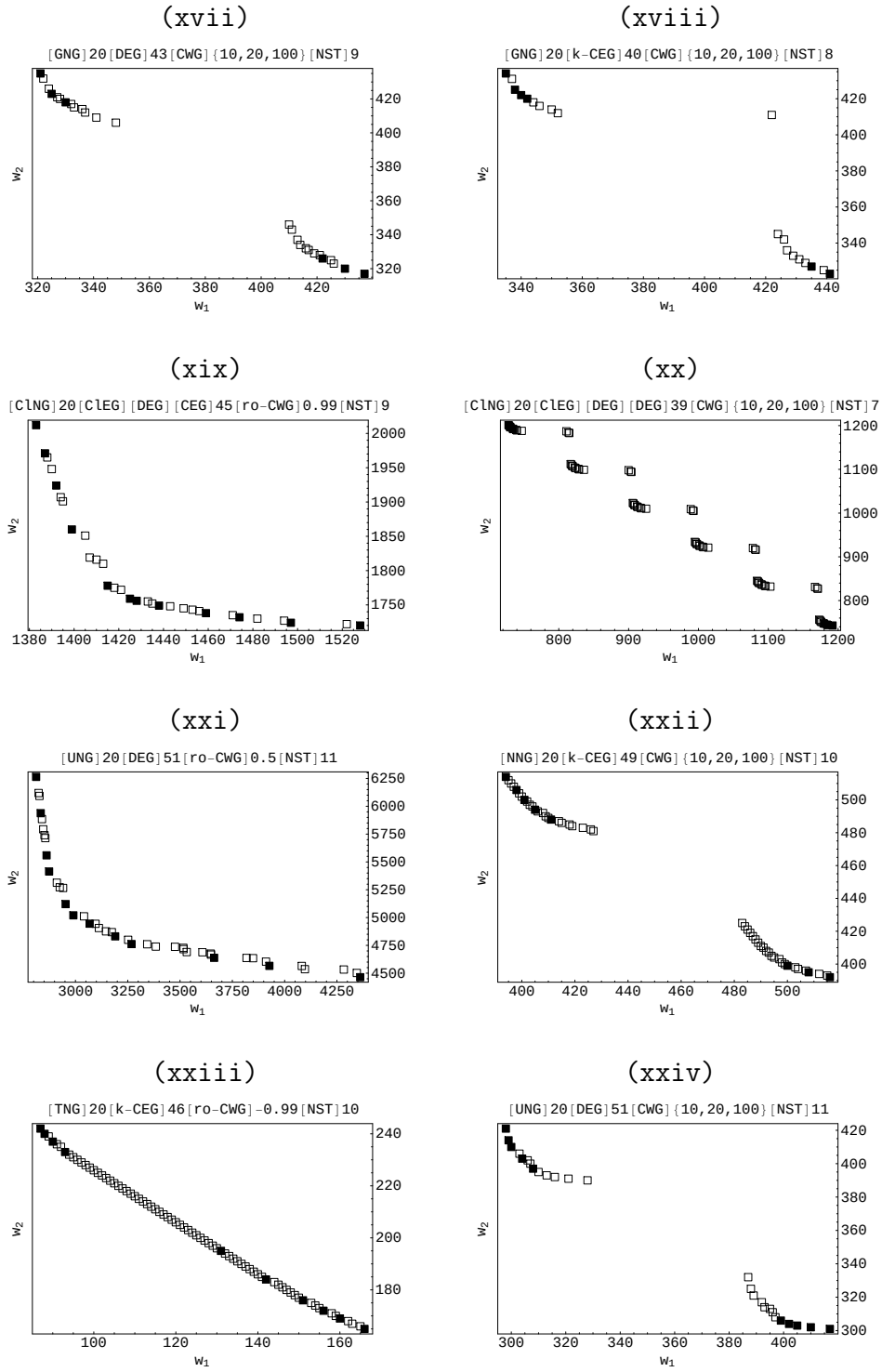


Figure 4.12: Examples of Pareto fronts (continuation)

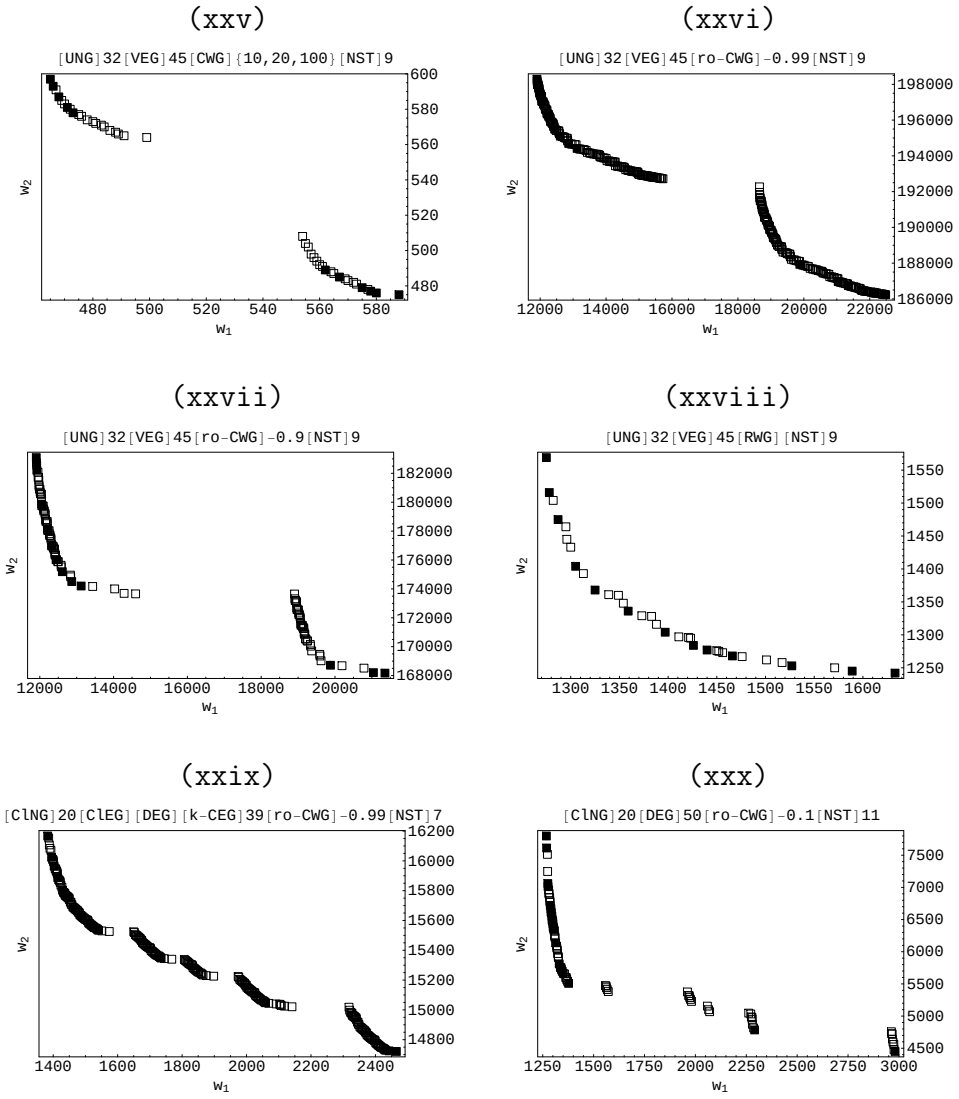


Figure 4.12: Examples of Pareto fronts (continuation)

The ants said: together we will be able to transport an elephant

TOGO proverb

The constant creeping of ants will wear away the stone

USA proverb

5

Ant Colony Heuristics for the Multiple Objective Problems

Contents

5.1 Overview

The previous chapters discussed some of the fragilities associated to the use of exact and approximation methods in the resolution of most of the multiple objective problems. In fact, the small number of exact methods, which are in their majority based on the branch-and-bound strategy, are not applicable to large instances of the problems and/or with several objectives. Other approaches transform the multiple objective problem into a single objective problem, recurring to the combination of the weights vector as a real function or through the establishment of priorities among the objectives. However this approach has also some pitfalls, has it was verified.

As a practical alternative to the computation of the Pareto solution, an approximation set is often considered as an acceptable response to the problem solution. In Section 2.3.3 were surveyed a set of meta-heuristic strategies that in the last decades gained consensus as pragmatic approximation methods. Their success is based in features like their flexible application to many of the optimization problems and, mainly because, they accomplishes good approximation sets in reasonable time.

The Evolutionary Algorithms are among the meta-heuristic with more success. In its majority they are based in populations of solutions/agents,which allow to simultaneously explore more than one region of the search space, reducing the risks of getting trapped in some local optima, and are less susceptible to problem noises.

However, the computational requirements demanded by the meta-heuristics imply that their performance might be compromised when applied to large scale problems. One possible solution is the parallelization which distributes the tasks through a set of computers, taking advantage of the population of agents. Another alternative is the use

of hybrid methods which allows exploring the best qualities of the various algorithms, accomplishing an improved global process.

In particular, two algorithms based on the Ant Colony Optimization (ACO) paradigm [Dorigo & Stutzle, 2004; Dorigo *et al.*, 1999] are proposed in this chapter: the Multiple Objective Network optimization based on an ACO (`MONACO`) and the ϵ-Depth ANT Explorer (ϵ-`DANTE`).

Like others Ant Colony Optimization algorithms, `MONACO` mimics the ant's foraging behaviour supported by a communication process, which relies on a chemical pheromone trail, signalling a good path to some supply location. However, one of the main features, which differentiates `MONACO` from almost all the others Ant Colony algorithms, is the fact that `MONACO` process uses a pheromone vector associated to each atomic piece, as if there were several layers of pheromones. Each component of those vectors is associated to a weight and, as before, represents how worthy the elements were, for some time-window, in the construction of the solutions relatively to the associated weight. Those numerical values are used by each ant/agent[1] to pseudo-randomly support the construction of the solutions.

When a solution is complete, it is evaluated and the Pareto set is update accordingly to its fitness. This solution is possibly used to update the pheromone trails (it depends on the ACO version) to be discarded next. This implies that the computational effort needed to build the solution is not completely explored, being necessary to start all over again.

The ϵ-`DANTE` method has a similar approach to the `MONACO` method since it uses a set of agents to construct solutions based on pseudo-random processes, founded in numerical pheromone layers associated to the potential solutions components. However,

[1]Note that most of the times, it will be used a common Ant Colony Optimization terminology where the constructing agents are referred as ants and the numerical pheromone vector are known as pheromones or pheromone trails.

the method has one main difference from the previous known Ant Colony algorithms: a restricted depth search oriented by the pheromone trails is made in selected situations. In other words, the method constructs a solution similarly to the MONACO process but, if it achieves a solution in an ϵ range to the approximation set then it does not discard it. Instead, the referred depth search is done with limited maximum level, maximum number of branches in each level, and possibly other features, like maximum computation time.

Therefore, this chapter is structured as follows. In the next section it is presented the MONACO algorithm. This method is used to simulate a network of flows and solves two combinatorial problems: the Multiple Objective Travelling Salesman Person and the Multiple Objective Minimum Spanning Trees problem. In Section 5.3 it is proposed the ϵ-DANTE algorithm and are presented the results obtained for a set of instance for two problems: the Multiple Objective Travelling Salesman Person and the Multiple Objective k-Degree Minimum Spanning Trees. The obtained results are then compared with reference sets presented by other authors using exact methods and Evolutionary Algorithms. With ϵ-DANTE it is described an Angle-Pheromone Updating strategy, which uses parts of the elements of the approximation set as contributors to the pheromone vectors trails, in successive time windows.

5.2 MONACO Algorithm

5.2.1 Application to the Multiple Objective Flows Simulation

The MONACO algorithm was first proposed by Cardoso *et al.* [2003a] as an ACO based algorithm for a multiple objective network optimization problem with a time discrete approach. The optimization process uses a single colony with a multi-level pheromone trail and an heuristic. The combination of these elements allows to introduce a pseudo-

random formula that is used in the construction of the approximations, to select the components, of the optimum paths.

More precisely, the method determines approximations to the paths that minimize the total weights for the flows generated between each pair of nodes of the network

$$\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z}, \mathcal{O}), \tag{5.1}$$

where $\mathcal{V}, \mathcal{E}$ and $\mathcal{Z}$ are defined in (3.1) and $\mathcal{O} = \{O_k : k \in I\}$ is a set of origin/destination flows matrices. Each matrix is defined as $O_k = \left[o_{i,j}^{(k)}\right]$, where $o_{i,j}^{(k)}$ is the flow from node i to node j at tick $k \in I$, for the time window of observation I.

During the process, is set a queue of agents/ants in each edge, corresponding to the flow elements that have not yet arrived to the end of those edges, since it is considered that a certain amount of ticks is necessary to allow the passing of the edges by each ant.

In the path construction, each ant is considered as a certain amount of flow with the same origin s, same destination t and leaving s at the same tick. Then, the path for that flow is constructed by the ant taking into account the multi-pheromone trail and an heuristic to improve the choices of suitable routes. Those pheromone trails, in the same number as the number of weights (m), represent the weight of an edge regarding the objective of guiding the ant to the destination node, t. Therefore, for each terminal node t, there are m pheromone trails that catalyse the construction of the path toward the destination node, that is, the pheromone trails deposited in the edges are associated to the quality observed for that edge in previous paths to t.

Knowing the necessary ticks to cross an edge, the problem resumes to choose the next edge whenever the ant arrives into a node that is not t (the ant stops when it reaches t). Mathematically, when the ant reaches node u, the probability of choosing

edge e_{uv} is given by the formula

$$p_{uv} = \begin{cases} \frac{d(e_{vt})^{-\alpha_0} \prod_{k=1}^{m} \left(\tau_{k,t}(e_{uv})\right)^{\alpha_k}}{\sum_{\{w:e_{uw}\in\mathcal{E}\}} \left[d(e_{wt})^{-\alpha_0} \prod_{k=1}^{m} \left(\tau_{k,t}(e_{uw})\right)^{\alpha_k}\right]} & \text{if } e_{uv} \in \mathcal{E} \\ 0 & \text{if } e_{uv} \notin \mathcal{E} \end{cases}, \tag{5.2}$$

where $\tau_{k,t}(e_{uv})$ is the quantity of the k-pheromone to node t in edge e_{uv} (analogous for $\tau_{k,t}(e_{uw})$), $d(e_{vt})$ is the Euclidean distance between nodes v and t (analogous for $d(e_{wt})$) and $\alpha_k \in \mathbb{R}_0^+$ $(k = 0, 1, \ldots, m)$ are parameters that emphasis the heuristic $(k = 0)$ and the relative importance of the k-pheromone trail $(k = 1, 2, \ldots, m)$, that is, the relative importance of the k weight in the final value.

Therefore, supposing that the ant is in a node u and wants to go to node t, a random-heuristic selection of the next node is made from the adjacent nodes, considering their previous ability in the construction of a good path. The heuristic factor gives preference to nodes that are closer to t by placing in the formula the inverse of the Euclidean distance from those nodes to t, which can be seen a guidance ability, while the pheromone vectors work as a memory of the previous paths. In Example 5.1 it is possible to observe the strict dependence of the probabilities on the parameters.

Example 5.1: Figure 5.1 presents a section of a network, where two costs are taken into account considering the values given in Table 5.1. Suppose also that the ant is on node a and t is its destination node. Then using (5.2) with $\alpha_0 = \alpha_1 = \alpha_2 = 1$ the probabilities are $p_{ab} = 0.03$, $p_{ac} = 0.43$, $p_{ad} = 0.29$ and $p_{ae} = 0.26$. Considering $\alpha_0 = \alpha_2 = 1$ and $\alpha_1 = 2$ then $p_{ab} = 0.02$, $p_{ac} = 0.36$, $p_{ad} = 0.40$ and $p_{ae} = 0.22$. Comparing the two cases it is possible to confirm that the variation of the probabilities, and therefore the choices of the ants, is closely related to the values of the parameters.

Relatively to the pheromones trails, when the process starts, they are all set equal. The construction of an effective pheromone trail is achieved considering a set of cycles.

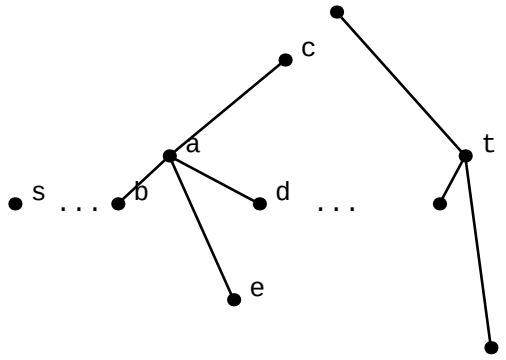


Figure 5.1: Random path determination example.

node (x)	distance from x to t	$\tau_{1,t}(a,x)$	$\tau_{2,t}(a,x)$
b	4	1.0	1.0
c	1	1.5	2
d	1.5	2.5	1.5
e	2	1.5	3

Table 5.1: Random path determination example.

In each cycle, a pre-determined number of ticks are run and each ant that arrives to its destination node contributes to the variation of the pheromone trails. After each cycle the k-pheromone trails for node t are updated using the formula:

$$\tau_{k,t}(e_{uv}) = \rho_k \tau_{k,t}(e_{uv}) + \Delta\tau_{k,t}(e_{uv}), \quad k \in \{1, 2, \ldots, m\}, \; t \in \mathcal{V} \tag{5.3}$$

where $0 < \rho_k \leq 1 \;\; (k = 0, 1, \ldots, m)$ is the persistence factor of the trail ($1 - \rho_k$ is trail evaporation factor) and $\Delta\tau_{k,t}(e_{uv})$ is the quantity of k-pheromone leaved by the ants that went through edge e_{uv} with destination t, in this cycle. That quantity is usually set as the inverse of the k weight of the path determined by the ant. In other words,

Algorithm 12 `MONACO` Algorithm – Flow simulation problem

input: $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z}, \mathcal{O})$, I ▷ *Network and ticks interval ($I = [T_i, T_f]$)*

1: Load the Network
2: **for all** $t \in \mathcal{V}$ **do**
3: **for all** $k \in \{1, 2, \ldots, m\}$ **do**
4: Initialise the k-pheromone trails ($\tau_{k,t}$) to node t
5: **end for**
6: **end for**
7: **repeat**
8: **for** $i \in I$ **do**
9: Update ant-packets already in the network (Algorithm 13).
10: Add new ant-packets to the networks (based on the origin/destination matrix for tick i).
11: **end for**
12: **for all** $t \in \mathcal{V}$ **do**
13: **for all** $k \in \{1, 2, \ldots, m\}$ **do**
14: Update the k-pheromone trails associated to t using (5.3).
15: **end for**
16: **end for**
17: Remove all remaining ants from network
18: **until** stopping criteria is met

if $\mathcal{A}^t_{u,v}$ is the set of all ants that went to t through edge e_{uv} and $\pi_a(s,t)$ represents the path of $a \in \mathcal{A}^t_{u,v}$, then

$$\Delta\tau_{k,t}(e_{uv}) = \sum_{a \in \mathcal{A}^t_{u,v}} \frac{Q}{w_k(\pi_a(s,t))}, \tag{5.4}$$

where $w_k(\pi_a(s,t))$ is the k component of the sum objective function $\mathcal{W}$, defined as in (3.2), and Q is a constant related to the amount of pheromone laid by the ants.

Algorithm 12 globally sketches the `MONACO` computational model and Algorithm 13 presents a more detailed phase, focused on the ants update.

The computational model was implemented using *C++* programming language and a set of tests were used to verify its robustness, liability and accuracy. The computational environment used was a Intel Pentium$^{\text{TM}}$IV with a $2.0Ghz$ processor, $256Mb$ of

Algorithm 13 Ants update.

```
1: for all ants in network do
2:     if ant arrived at the end of an edge then
3:         if ant arrived to its destination node, t then
4:             for all k ∈ {1, 2, ... , m} do
5:                 Update Δτ_{k,t} using (5.4)
6:             end for
7:             Remove ant from network
8:         else
9:             Use (5.2) to random-heuristically determine next edge and move the ant
                   to that edge.
10:        end if
11:    end if
12: end for
```

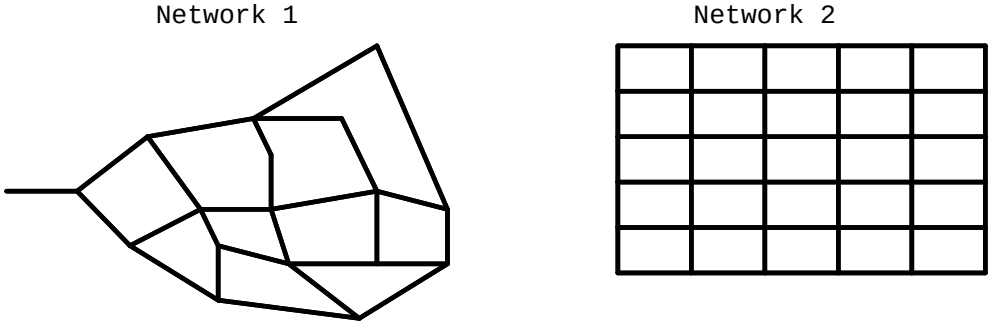


Figure 5.2: Example of tested networks.

RAM and Windows XP OS.

Two examples of networks used in the tests are depicted in Figure 5.2: Network 1 with 18 nodes and 27 edges and Network 2 with 25 nodes and 40 edges. For each problem, it was considered a weight vector associated to each edge with two components: number of ticks to cross the edge and its Euclidean distance. It was also considered 100 ticks per cycle, 20 cycles as a stopping criteria for Algorithm 12, persistence parameters $\rho_d = \rho_t = 0.5$ (associated to the number of ticks and Euclidean distance, respectively) and $\alpha_d, \alpha_t, \alpha_h \in \{1, 2\}$ (associated, in this case, to the distance weight, the ticks weight and the heuristic, respectively). In general, it were used 122400 and 240000 ants/cycle

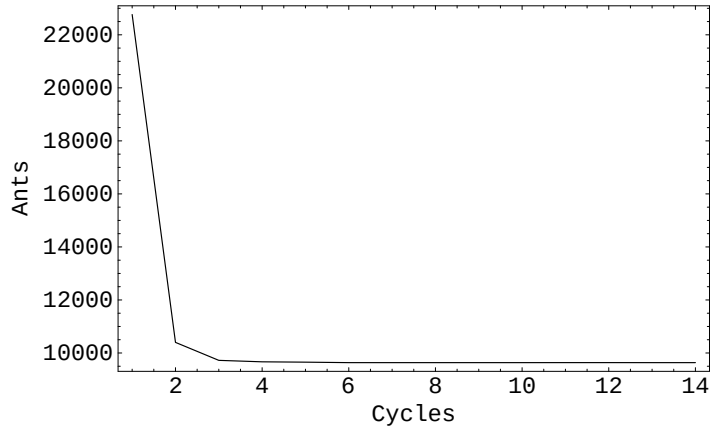


Figure 5.3: Number of ants present in the network 1 at the end of the cycles - $\alpha_d = 2$ and $\alpha_t = \alpha_h = 1$.

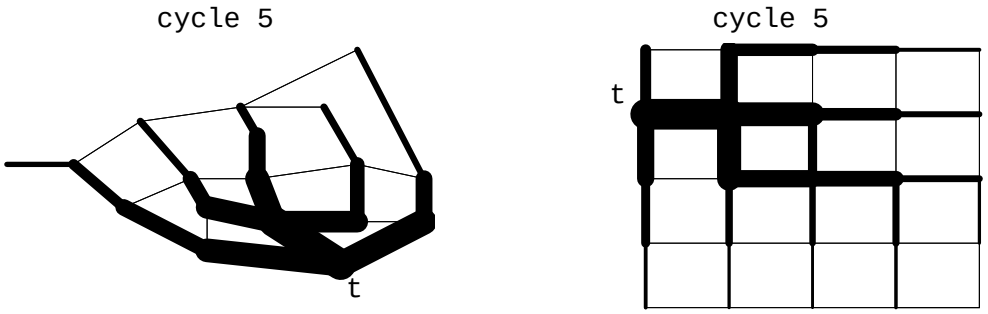


Figure 5.4: Pheromone trails relative to distance weight for node t - the thickness of the edges corresponds to the quantity of pheromone present in cycle 5 ($\alpha_d = 2$ and $\alpha_t = \alpha_h = 1$).

for Network 1 and Network 2, respectively.

Figure 5.3 sketches the variation of the number of ants at the end of each cycle for Network 1. From the analysis of the figure, we can conclude that in few cycles a good trail of pheromone is created, implying an important diminishing in the number of ants that have not reached destination, that is, less ants are "lost" as the number of cycles increases. That number suggest a stabilization above the number of ants created in the last cycles that have not enough ticks to reach its destination node.

—Euclidean distance—

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	4	-	5	$3\sqrt{2}$	$5\sqrt{2}$	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	5	-	-	5	-	-	$\sqrt{37}$	-	-	-	-	-	-	-	-	-	-
4	-	$3\sqrt{2}$	-	-	$2\sqrt{5}$	-	$\sqrt{34}$	-	-	-	-	-	-	-	-	-	-	-
5	-	$5\sqrt{2}$	5	$2\sqrt{5}$	-	$\sqrt{5}$	-	-	-	4	-	-	-	-	-	-	-	-
6	-	-	-	-	$\sqrt{5}$	-	3	-	-	-	$\sqrt{17}$	-	-	-	-	-	-	-
7	-	-	-	$\sqrt{34}$	-	3	-	-	-	-	-	$\sqrt{65}$	-	-	-	-	-	-
8	-	-	$\sqrt{37}$	-	-	-	-	-	$\sqrt{5}$	-	-	-	-	-	5	$\sqrt{65}$	-	-
9	-	-	-	-	-	-	-	$\sqrt{5}$	-	3	-	-	-	-	-	-	-	-
10	-	-	-	-	4	-	-	-	3	-	$\sqrt{10}$	-	-	$\sqrt{37}$	-	-	-	-
11	-	-	-	-	-	$\sqrt{17}$	-	-	-	$\sqrt{10}$	-	5	5	-	-	-	-	-
12	-	-	-	-	-	-	$\sqrt{65}$	-	-	-	5	-	-	-	-	-	-	$\sqrt{34}$
13	-	-	-	-	-	-	-	-	-	-	5	-	-	4	-	-	-	4
14	-	-	-	-	-	-	-	-	-	$\sqrt{37}$	-	-	4	-	$2\sqrt{5}$	-	$\sqrt{17}$	-
15	-	-	-	-	-	-	-	5	-	-	-	-	-	$2\sqrt{5}$	-	-	-	-
16	-	-	-	-	-	-	-	$\sqrt{65}$	-	-	-	-	-	-	-	-	$\sqrt{97}$	-
17	-	-	-	-	-	-	-	-	-	-	-	-	-	$\sqrt{17}$	-	$\sqrt{97}$	-	3
18	-	-	-	-	-	-	-	-	-	-	-	$\sqrt{34}$	4	-	-	-	3	-

—Edge velocity—

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	5	-	5	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	5	-	-	1	-	-	5	-	-	-	-	-	-	-	-	-	-
4	-	1	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-
5	-	1	1	1	-	1	-	-	-	1	-	-	-	-	-	-	-	-
6	-	-	-	-	1	-	1	-	-	-	1	-	-	-	-	-	-	-
7	-	-	-	1	-	1	-	-	-	-	-	1	-	-	-	-	-	-
8	-	-	5	-	-	-	-	-	1	-	-	-	-	-	5	1	-	-
9	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-
10	-	-	-	-	1	-	-	-	1	-	1	-	-	1	-	-	-	-
11	-	-	-	-	-	1	-	-	-	1	-	3	3	-	-	-	-	-
12	-	-	-	-	-	-	1	-	-	-	3	-	-	-	-	-	-	1
13	-	-	-	-	-	-	-	-	-	-	3	-	-	3	-	-	-	1
14	-	-	-	-	-	-	-	-	-	1	-	-	3	-	3	-	1	-
15	-	-	-	-	-	-	-	5	-	-	-	-	-	3	-	-	-	-
16	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	1	-
17	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	1
18	-	-	-	-	-	-	-	-	-	-	-	1	1	-	-	-	1	-

Table 5.2: Euclidean distances and edge velocity between nodes of Network 1.

In Figure 5.4 the pheromone levels toward a node t, represented by the thickness of the edges, are sketched for both networks, which gives us an idea of the possible choices made by the ants.

Relatively to the influence of the parameters, if we compare Network 1 in Figures 5.4 and 5.5, it is possible to observe differences in the line thicknesses of some edges. Those differences are associated to variations in the values of the parameter (in the first figure $\alpha_d > \alpha_t$ and in the second $\alpha_d < \alpha_t$). For example, in Figure 5.5 the thicker lines correspond to the edges that conduct to t in a small amount of ticks, while in Figure 5.4 are the edges that conduct to t through a shortest path (see Table 5.2

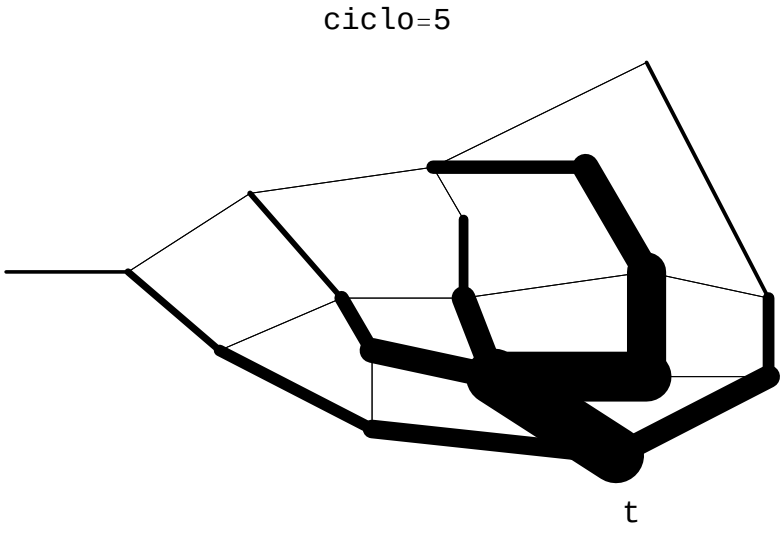


Figure 5.5: Pheromone trails relative to ticks weights for node t - the thickness of the edges corresponds to the quantity of pheromone present in cycle 5 ($\alpha_t = 2$ and $\alpha_d = \alpha_h = 1$).

for the Euclidean distances and edges velocities). Therefore, depending on the alpha parameters, the pheromone trails reflect the importance of the weights associated to those factors.

5.2.2 Application to the Multiple Objective Travelling Salesman Person Problem

A first approach to the Multiple Objective Travelling Salesman Person problem using MONACOis proposed in [Cardoso *et al.*, 2003b]. Maintaining the same idea, as in the flows simulator presented in the previous section, it uses as many pheromone trails as the number of weights. In this case, the construction of the Hamiltonian cycle is made in a constructive manner, such that, if the last node that was selected is u then the

probability of adding edge e_{uv} to the cycle is given by

$$p_{uv} = \begin{cases} \frac{\Pi_{k=1}^{m}(\tau_k(e_{uv}))^{\alpha_k}(w_k(e_{uv}))^{-\beta_k}}{\sum_{w \in \mathcal{U}} \Pi_{k=1}^{m}(\tau_k(e_{uv}))^{\alpha_k}(w_k(e_{uv}))^{-\beta_k}} & \text{if} \quad v \in \mathcal{U} \\ 0 & \text{if} \quad v \notin \mathcal{U} \end{cases}, \tag{5.5}$$

where $\mathcal{U}$ is the set of nodes that still do not belong to the cycle, $\tau_k(e_{uv})$ is the quantity of k-pheromone in edge e_{uv}, $w_k(e_{uv})$ is the k weight of edge e_{uv}, and $\alpha_1, \alpha_2, \ldots, \alpha_m$ and $\beta_1, \beta_2, \ldots, \beta_m$ are parameters that, respectively, emphasis the weights and the local greedy heuristic importance, which favours the selection of nodes near to node u. Each cycle was completed through the application of a 2-opt non-stochastic optimizer. In essence, the 2-opt optimizer starts with the cycle provided by MONACO. Then the algorithm improves the tour repeatedly by exchanging two edges at a time. These exchanges are performed if the cycle fitness is improved and done until no better solution can be derived.

A set of tests were run with parameters

$$\alpha_1, \alpha_2, \beta_1, \beta_2 \in \{1, 5, 10\}, \rho_1 = \rho_2 = 0.1 \tag{5.6}$$

(the pheromone updating formulas are similar to the ones presented in formulas (5.3) and (5.4)) and only the elements of the approximation set contribute to the variation of the pheromone trail.

Figure 5.6 sketches the approximation sets obtained with MONACO (continuous line) and MOGLS [Jaszkiewicz, 2002] (dashed line) for the *kroab*50 and *kroab*100 problems. The *kroab*50 problem is obtained through the combination of the *kroa*50 and the *krob*50 available at [Jaszkiewicz, 2006b; TSPLib, 2003] (analogous for *kroab*100).

Others implementations

For the same problem, other authors adapted/implemented the MONACO's concept and compared the results with the outcomes of other multiple objective optimization Swarm

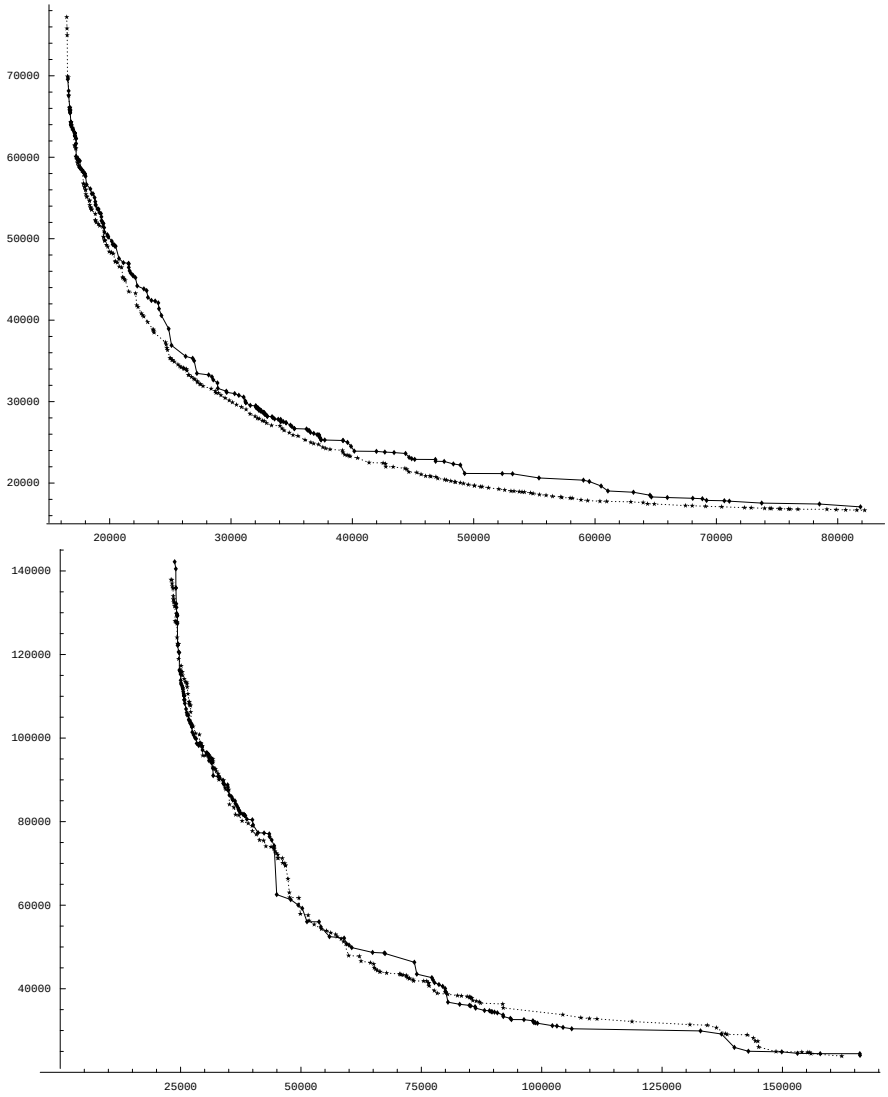


Figure 5.6: Results from MONACO for *kroab*50 and *kroab*100.

Algorithms [García-Martínez *et al.*, 2004a]: Multiple Objective Ant-Q Algorithm, Ant Algorithm for Bi-criterion Optimization Problems, Multi Colony for Bi-criterion Optimization Problems, Pareto Ant Colony Optimization (P-ACO), Multiple Ant Colony System for Vehicle Routing Problem with Time Windows, Multiple Ant Colony System, COMPETants, Multiple Objective Ant Colony Optimization Metaheuristic, Ant Colony Optimization Approach to Multiple Objectives and SACO (in [García-Martínez *et al.*, 2004b] is a more detailed report from same authors). Their test used six bi-objective problem instances: *kroab*50, *krocd*50,*kroab*100, *kroad*100, *krobc*100, and

*krocd*100. Furthermore, two Multiple Objective Genetic Algorithms were considered as baselines for the Swarm Algorithms performance: the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [Zitzler *et al.*, 2001, 2002a] and the Nondominated Sorting Genetic Algorithm II (NSGAII) [Deb *et al.*, 2000].

As a result, their implementation of MONACO method was considered to

> "(...) we can see that both MOGAs [Multiple Objective Genetic Algorithms], NSGA-II and SPEA2, can usually perform much more iterations and evaluations than the MOACO [Multiple Objective Ant Colony] algorithms considered in the same fixed run time. Hence, this shows how MOACO algorithms are somehow slower than MOGAs in the current problem(...)
>
> (...) Non-dominated solution sets returned by both MOGAs, NSGA-II and SPEA2, are poor in comparison with most of those returned by MOACO algorithms. These sets can only dominate some solutions returned by COMPETants and MOAQ when applied to small instances of size 50(...)
>
> (...) P-ACO and MONACO algorithms return very good solutions in the central part of the Pareto front but they are not able to generate any solution at all in the extents of the Pareto front(...)
>
> (...) Pareto fronts returned by P-ACO and MONACO are practically not dominated by those obtained from the remaining algorithms. So we could say that P-ACO and MONACO are the algorithms with the best performance according to this metric [C metric], because their Pareto fronts are not dominated (...)
>
> (...)The only case we found where the algorithms of a family behave in the same way is that of P-ACO and MONACO, characterized by two

> pheromone matrices and a single heuristic matrix. Both algorithms achieve a good convergence to the central parts of the Pareto fronts, avoiding the generation of solutions in the extremes of the front.(...)"
>
> [García-Martínez *et al.*, 2004b]

Globally, the authors concluded that MONACO and P-ACO achieve very good solutions in central parts of the Pareto front but could not generate a spread set of solutions all along the Pareto front, as it is desirable. However, in their approach the parameters were set as $\alpha_1 = \alpha_2 = 1$, $\beta_1 = \beta_2 = 2$, and $\rho_1 = \rho_2 = 0.2$, and kept constant along the process. A different approach, as the one presented earlier, where the parameters assume different weights according to formula (5.6), or a more greedy strategy as the one that will be presented in Section 5.3.2, would probably favour a wide exploration of the Pareto front.

5.2.3 Application to the Multiple Objective Minimum Spanning Trees Problem

`MONACO` was later adapted to the Multiple Objective Minimum Spanning Trees problem [Cardoso *et al.*, 2004, 2005b].

In this approach, the construction of a spanning tree by `MONACO` is divided in two phases: in the first phase a set of disjoint subtrees is built and in the second phase those subtrees are joined to form a final tree.

In both phases the process uses m pheromone trails, τ_k $(k = 1, 2, \ldots, m)$, where each trail is associated to a weight, and a local heuristic that favours the inclusion of edges with lower weights.

Phase 1: Construction of the Disjoint Subtrees

The process begins by determining a set of disjoint subtrees. It starts by randomly choose a node, u_0, that does not belong to any subtree. From that node it is built a path, which is a subtree, by the means of the ordered addition of nodes, selected with probability $p_{u_i u_{i+1}}$ given by formula:

$$p_{u_i u_{i+1}} = \frac{\prod_{k=1}^{m} \tau_k(e_{u_i u_{i+1}})^{\alpha_k} w_k(e_{u_i u_{i+1}})^{-\beta_k}}{\sum_{z \in \{v : e_{u_i v} \in \mathcal{E}\}} \prod_{k=1}^{m} \tau_k(e_{u_i z})^{\alpha_k} w_k(e_{u_i z})^{-\beta_k}}, e_{u_i u_{i+1}} \in \mathcal{E}, i = 0, 1, 2, \ldots, \quad (5.7)$$

where $\alpha_k, \beta_k \in \mathbb{R}_0^+ (k = 1, 2, \ldots, m)$ are parameters associated to the relative importance of weight k and the local heuristic, $w_k(e_{u_i v})^{-\beta_k}$, respectively. This local heuristic, as already referred, favours the addition of the more promising edges, with lower weights, which *de per se* improves the convergence to the Pareto front.

The subtree construction stops when the selected node already belongs to some subtree. When it happens, two things can occur:

- The last selected node does not belong to the tree in construction. In this case, it is obtained a subtree by joining the subtree under construction with the intersected, which is kept for latter merge in Phase 2.

- The node that was selected already belongs to the subtree in construction that is disjoint from any other. In this case, the chosen edge is discarded, to avoid cycles, and this subtree is also kept and later joined with the others constructed subtrees.

The described process is repeated until all nodes belong to some subtree. In Algorithm 14 is a high level description of the process and in Example 5.2 (steps (a)-(f)) we can see a typical run for a 8 node network.

Algorithm 14 Low-level description of the subtrees construction.

input: $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z})$, α_k, $\beta_k (k = 1, 2, \ldots, m)$ ▷ *Network and algorithm parameters*

output: Set of disjoint subtrees, $\mathcal{T}$.

1: $\mathcal{T} \leftarrow \emptyset$ ▷ *set of disjoint subtrees*
2: $\mathcal{L} \leftarrow \emptyset$ ▷ *node taboo list*
3: **while** $\mathcal{L} \neq \mathcal{V}$ **do**
4: $T \leftarrow \emptyset$
5: Randomly select a node u from $\mathcal{V} - \mathcal{L}$
6: $\mathcal{L} \leftarrow \mathcal{L} \cup \{u\}$,
7: $out \leftarrow false$
8: **repeat**
9: Choose v with probability p_{uv}, using formula(5.7)
10: **if** $v \notin \mathcal{L}$ **then** ▷ *v does not belong to any subtree*
11: $T \leftarrow T \cup \{e_{uv}\}$
12: $\mathcal{L} \leftarrow \mathcal{L} \cup \{v\}$,
13: $u \leftarrow v$
14: **else** ▷ *v already belongs to some subtree, T'*
15: **if** $v \notin T$ and $v \in T'$ **then** ▷ $T' \neq T$
16: $\mathcal{T} \leftarrow \mathcal{T} - \{T'\}$
17: $T \leftarrow T \cup T' \cup \{e_{u,v}\}$ ▷ *Trees T and T' are joined*
18: **end if**
19: $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$,
20: $out \leftarrow true$
21: **end if**
22: **until** $out = true$ or $\mathcal{L} = \mathcal{V}$
23: **end while**
24: **return** $\mathcal{T}$

Phase 2: Merge of the Subtrees

In the first phase, a set of disjoint subtrees $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ were generated. Now, the process start by picking one subtree from $\mathcal{T}$, T_i, and follows by searching for a good connection to one of the others, $\mathcal{T} - \{T_i\}$. That connection is made by pseudo-randomly choosing an edge e_{uv} such that $u \in T_i$ and $v \notin T_i$. The probability of one admissible edge being inserted into the tree is given by:

Algorithm 15 Low-level description of the subtrees fusion.

input: $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{Z})$, $\mathcal{T} = \{T_1, T_2, \ldots, T_p\}$, α_k, $\beta_k (k = 1, 2, \ldots, m)$ ▷ *Network, set of disjoint trees and algorithm parameters*

output: T ▷ *spanning tree*

1: **while** $|\mathcal{T}| > 1$ **do**
2: Randomly choose a subtree, T_i, from $\mathcal{T}$
3: Using formula (5.8) randomly select e_{uv} such that $u \in T_i$ and $v \in T_j$ with $i \neq j$
4: $\mathcal{T} \leftarrow \mathcal{T} - \{T_i, T_j\}$
5: $\mathcal{T} \leftarrow \mathcal{T} \cup \{T_i \cup T_j \cup \{e_{uv}\}\}$
6: **end while**
7: **return** $T \in \mathcal{T}$ ▷ $|\mathcal{T}| = 1$

$$p_{uv} = \frac{\prod_{k=1}^{m} \tau_k(e_{uv})^{\alpha_k} w_k(e_{uv})^{-\beta_k}}{\sum_{f \in \mathcal{Q}} [\prod_{k=1}^{m} \tau_k(f)^{\alpha_k} w_k(f)^{-\beta_k}]}, e_{uv} \in \mathcal{Q}, \tag{5.8}$$

where $\mathcal{Q} = \{e_{xy} \in \mathcal{E} : x \in \mathcal{V}_{T_i} \wedge y \notin \mathcal{V}_{T_i}\}$. This process is repeated until it remains a single tree in $\mathcal{T}$.

Algorithm 15 has a high level description of the merge part and Example 5.2 (steps (g)-(i)) presents a complete description of this process.

Example 5.2: Consider the network in Figure 5.7 with associated edges weights (w_1, w_2). Suppose that the process has been running for some time, with parameters $\alpha_1 = 1$, $\alpha_2 = 2$, $\beta_1 = 2$ and $\beta_2 = 2$, and the pheromone trails have values given by (τ_1, τ_2), resumed in Table 5.3.

The process to build a tree can be described as follows (in Figure 5.7 is also depicted the process, where the continuous lines are the edges already included and the dashed lines are the candidate edges for entering in the tree):

(a) Randomly select a node. Supposing that it is node 1, the probabilities of adding

—$\mathcal{Z}$—								
	1	2	3	4	5	6	7	8
1		(1,2)	(3,3)	(2,2)	(1,2)	(2,3)	(5,4)	(7,5)
2	(1,2)			(1,1)	(3,3)			
3	(3,3)					(2,3)		(1,2)
4	(2,2)	(1,1)				(2,1)		
5	(1,2)	(3,3)					(1,2)	
6	(2,3)		(2,3)	(2,1)				
7	(5,4)				(1,2)			(1,3)
8	(7,5)		(1,2)				(1,3)	

.

—τ—								
	1	2	3	4	5	6	7	8
1		(5,4)	(1,2)	(4,4)	(5,4)	(2,2)	(1,1)	(1,2)
2	(5,4)			(3,4)	(2,3)			
3	(1,2)					(4,4)		(4,5)
4	(4,4)	(3,4)				(6,5)		
5	(5,4)	(2,3)					(5,5)	
6	(2,2)		(4,4)	(6,5)				
7	(1,1)				(5,5)			(2,3)
8	(1,2)		(4,5)				(2,3)	

Table 5.3: Edges weights and pheromone trails.

each of the edges with this starting node are

$$p_{1,2} = \frac{\left(\frac{1}{1}\right)^2 5 \left(\frac{1}{2}\right)^2 4^2}{\left(\frac{1}{1}\right)^2 5 \left(\frac{1}{2}\right)^2 4^2 + \left(\frac{1}{3}\right)^2 1 \left(\frac{1}{3}\right)^2 2^2 + \cdots + \left(\frac{1}{7}\right)^2 1 \left(\frac{1}{5}\right)^2 2^2} = 0.452,$$

$p_{1,3} = 0.001$, $p_{1,4} = 0.090$, $p_{1,5} = 0.452$, $p_{1,6} = 0.005$, $p_{1,7} = 0.0$ and $p_{1,8} = 0.0$. Edge e_{12} is selected, Figure 5.7 (a)-(b);

(b) From node 2 the probabilities are $p_{2,1} = 0.293$, $p_{2,4} = 0.704$ and $p_{2,5} = 0.003$. Edge e_{24} is added, Figure 5.7 (b)-(c);

(c) From node 4 the probabilities are $p_{4,1} = 0.044$, $p_{4,2} = 0.536$ and $p_{4,6} = 0.419$. Edge e_{42} is selected. Since it would generate a cycle the construction of this subtree is

stopped. Other node is randomly choose to restart the process. Node 5 is selected, Figure 5.7 (c)-(d);

(d) From node 5 the probabilities are $p_{5,1} = 0.389$, $p_{5,2} = 0.004$ and $p_{5,7} = 0.607$. Edge e_{51} is added, Figure 5.7 (d)-(e);

(e) Since node 1 belongs to other subtree, the two subtrees (the one in construction and the one to which node 1 belongs) are joined;

(f) The process in steps (a) to (e) is repeated until all nodes belong to one subtree, Figure 5.7 (f);

(g) Randomly choose a subtree. Subtree T_1 is selected and will be connected to another subtree, according to formula (5.8), with probabilities $p_{1,3} = 0.001$, $p_{1,6} = 0.003$, $p_{1,7} = 0.0$, $p_{1,8} = 0.0$, $p_{4,6} = 0.543$ and $p_{5,7} = 0.453$. Edge e_{57} is selected, Figure 5.7 (g)-(h);

(h) Repeat the last step until all subtrees are connected;

(i) Final tree, Figure 5.7 (i).

Tests and Results

Computational Environment A set of instances obtained from the MOST repository (Section 4) was used to test MONACO with the Multiple Objective Minimum Spanning Trees problem. The obtained result are compared with two other algorithms: Brute Force method and Weighted Sum method.

Brute Force Method The results for the Brute Force method were obtained through an implementation of the algorithm proposed by Ramos *et al.* [1998]. The tests

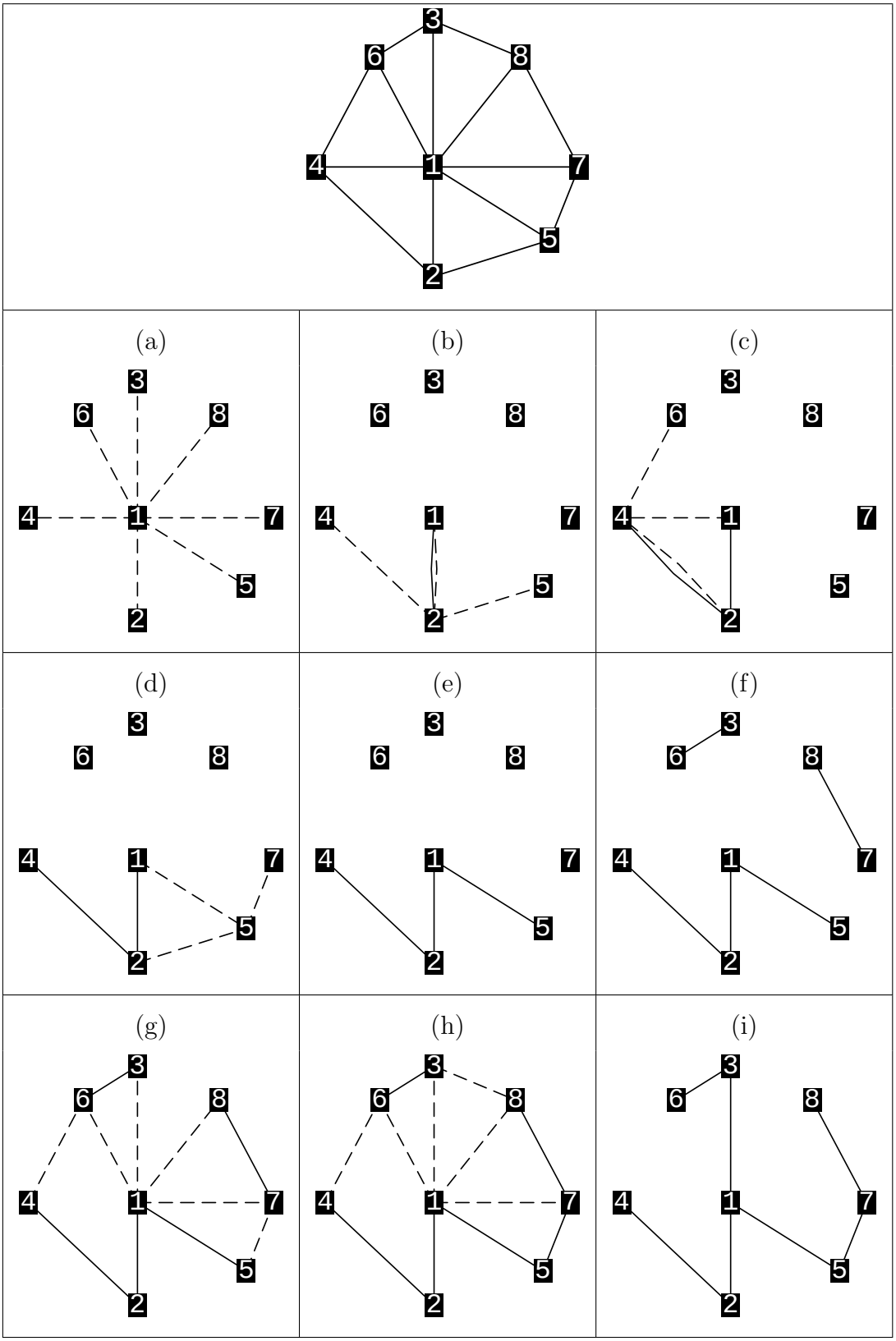


Figure 5.7: Simulation of the building tree process.

were run on a LINUX OS workstations with Intel Pentium$^{\text{TM}}$III $533Mhz$ processors and $128Mb$ of RAM. Due to its time complexity, the use of this method was restricted to the cases with less than 10^{11} distinct networks. A quadtree data structure was used to maintain the Pareto set [Mostaghim *et al.*, 2002].

Weighted Sum Method The results for the weighted sum approach (Section 2.3.2) were obtained using 1000 distinct pondering weights:

$$\Lambda = \left\{ (\lambda_1, \lambda_2) : \lambda_1 \in \left\{ \frac{k}{1001} : k = 1, 2, \ldots, 1000 \right\} \text{ and } \lambda_2 = 1 - \lambda_1 \right\}. \tag{5.9}$$

MONACO The tests were run on the same environment of Brute Force case. The running time was limited to $|\mathcal{E}| \log |\mathcal{E}|$ seconds, and it was also kept the last update time of the approximation set for reference. To maintain the approximation set it was used a quadtree data structure. In each cycle were used $\lfloor |\mathcal{N}|^{1/2} \rfloor$ ants per cycle and 1 cycle for each pair of (α_1, α_2) parameters, with extra cycles whenever the approximation set was improved. Additional parameters were $\rho_k = 0.9$, α_1 varied in $\{0, 0.25, 0.5, \ldots, 3.0\}$, $\alpha_2 = 3.0 - \alpha_1$ and $\beta_k = \frac{\alpha_k}{2} (k = 1, 2)$.

To take advantage of the quadtree data structure and the pheromone vectors updating strategy, a fast (time limited) set of iterations, with $\rho = 0$ and α_1 following the sequence $1.5, 1.25, \ldots, 0, 1.75, 2.0, \ldots, 3.0$, were run. This provides a first approximation to the Pareto front, which will be used in the pheromone vectors update strategy. It has the advantage to spread the solutions all over the branches of the quadtree, avoiding a possible quasi-linear distribution.

Furthermore, after each cycle the pheromone trails were updated using formula

$$\tau_k(e) = \rho_k \tau_k(e) + \Delta\tau_k(e), \tag{5.10}$$

		$R1$			$R3$		
		(hf, ws)	(hf, aco)	(ws, aco)	(hf, ws)	(hf, aco)	(ws, aco)
20	Mean(μ)	0.7548†	0.5338†	0.2933	-0.0005	-0.0003	-0.0003
	S.d.(σ)	0.1594†	0.0826†	0.1911	0.0007	0.0018	0.0023
50	Mean(μ)	–	–	0.5085	–	–	-0.0101
	S.d.(σ)	–	–	0.3393	–	–	0.0271
100	Mean(μ)	–	–	0.7266	–	–	-0.0879
	S.d.(σ)	–	–	0.3145	–	–	0.2805
all	Mean(μ)	–	–	0.4749	–	–	-0.0278
	S.d.(σ)	–	–	0.3291	–	–	0.1543

Table 5.4: Mean and standard deviation for the $R1$ and $R3$ metrics obtained for 20, 50 and 100 nodes networks († - results for 130 networks for which the Brute Force method was run).

where $k \in \{1, 2, \ldots, m\}$, $e \in \mathcal{E}$,

$$\Delta\tau_k(e) = \sum_{T \in \mathcal{T}_e} \frac{Q}{w_k(T)}, \tag{5.11}$$

$\mathcal{T}_e$ is the set of solutions generated during the cycle containing edge e, and Q is a value in the same magnitude of the solutions.

Results

For a more detailed analysis, the results presented here are divided according to the size of the problems and the methods used to approximate them.

20 Nodes Networks The Brute Force method was applied to 130 networks with less then 10^{11} distinct spanning trees, from a total of 249 networks with 20 nodes.

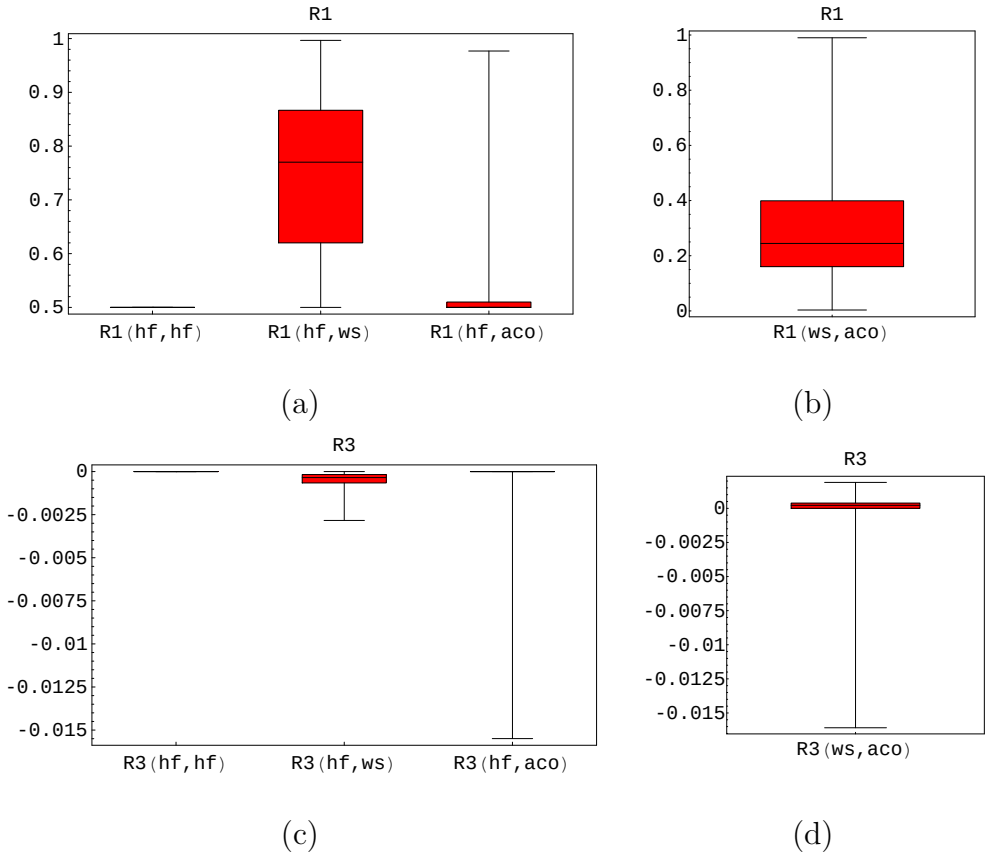


Figure 5.8: Box-and-whisker plots for: (a) and (c) – $R1$ and $R3$ metrics comparing Brute Force Method (hf) with the Weighted Sum (ws) and MONACO (aco) for the 130 networks of 20 nodes; (b) and (d) – $R1$ and $R3$ metrics comparing the Weighted Sum and MONACO methods for all 249 networks of 20 nodes.

Figures 5.8 shows the box-and-whisker plots for the $R1$ and $R3$ metrics, respectively, comparing Brute Force method (hf) with the Weighted Sum (ws) and MONACO (aco) methods. Table 5.4 resumes the values of the mean and the standard deviation for those metrics values.

For the studied problems, MONACO's results are similar to the results returned by Brute Force method (note that the Brute Force method returns the exact Pareto set, which implies that $R1(hf, \mathcal{P}_1) \geq \frac{1}{2}$ and $R3(hf, \mathcal{P}_1) \geq 0$ for any approximation $\mathcal{P}_1$). Therefore, the 0.5338 value for the mean of $R1(hf, aco)$ indicates that MONACO return similar values to the ones of Brute Force method. This observation, is corroborated by

50 nodes								
	CWG		$\rho - CWG$ $\rho \geq 0$		$\rho - CWG$ $\rho < 0$		RWG	
	μ	σ	μ	σ	μ	σ	μ	σ
$R1$	0.1293	0.1810	0.4700	0.2382	0.7855	0.2573	0.5078	0.3400
$R3$	0.0440	0.1830	-0.1040	0.2050	-0.1382	0.1368	-0.0736	0.1435
#	32		52		50		20	
100 nodes								
	CWG		$\rho - CWG$ $\rho \geq 0$		$\rho - CWG$ $\rho < 0$		RWG	
	μ	σ	μ	σ	μ	σ	μ	σ
$R1$	0.3185	0.2585	0.7736	0.2380	0.8877	0.2269	0.8957	0.1060
$R3$	-0.0379	0.2487	-0.3537	0.3412	-0.4244	0.4648	-0.1573	0.1649
#	36		48		50		25	

Table 5.5: Mean (μ) and standard deviation (σ) values for the $R1(ws, aco)$ and $R3(ws, aco)$ metric values according to the network size and weight generator (# – number of networks in each class).

the fact that $R1(hf, aco)$ returned 0.5 (and $R3(hf, aco) = 0$) in 95 of the 130 cases.

In general, for the $R1$ and $R3$ metrics, `MONACO` performed better than the Weighted Sum method in a great number of networks ($\mu_{R1(ws,aco)} = 0.2933$), but with similar approximation sets ($\mu_{R3(ws,aco)} = -0.0003$). In Figure 5.8 (d) we can see the box-and-whisker plots for the observed results.

50 Nodes Networks For the 50 nodes networks case were considered 154 problems. Figures 5.9 (a)-(b) sketch the box-and-whisker plots for $R1$ and $R3$ metrics, and the mean and standard deviation values of $R1$ and $R3$ according to the weight generators are resumed in Table 5.5.

In this case, the Weighted Sum method and MONACO have similar solutions, according to the $R1$ and $R3$ metrics, since $\mu_{R1(ws,aco)} = 0.5085$ and $\mu_{R3(ws,aco)} = -0.0101$ (see Table 5.4). However, if the problems are divided into the classes presented in Table 5.5, it is possible to conclude that when compared to the Weighted Sum, MONACO achieves better values for the $R1$ and $R3$ metrics when applied to networks generated using the CWG. On the other hand, the worst cases are obtained for the $\rho - CWG$ with $\rho < 0$ networks, where $\mu_{R1(ws,aco)} = 0.7855$ and $\mu_{R3(ws,aco)} = -0.1382$.

100 Nodes Networks For the 100 nodes case were considered 159 networks. Table 5.4 presents the statistics for the values of the $R1$ and $R3$ metrics and Figures 5.9 (c)-(d) shows the box-and-whisker plots for those same metrics. The mean and standard deviation values of $R1$ and $R3$ according to four weight classes of network generators are resumed in Table 5.5.

In this case the Weighted Sum method outperforms MONACO in most of the problems with $R1(ws, aco) = 0.7266$ and $R3(ws, aco) = -0.0879$. The best results for the $R1$ and $R3$ metrics are achieved for the networks obtained with the CWG where $\mu_{R1(ws,aco)} = 0.3185$.

Overall Values For the overall 562 networks Figure 5.10(a)-(b) shows the box-and-whisker for $R1$ and $R3$ metrics and Tables 5.4 and 5.6 resume the values for the mean and standard deviation of the $R1$ and $R3$ metrics.

MONACO is sightly better than Weighted Sum for the $R1$ metric ($\mu_{R1(ws,aco)} = 0.4749$), although the large value of the standard deviation. $R3$ returned a small variation with

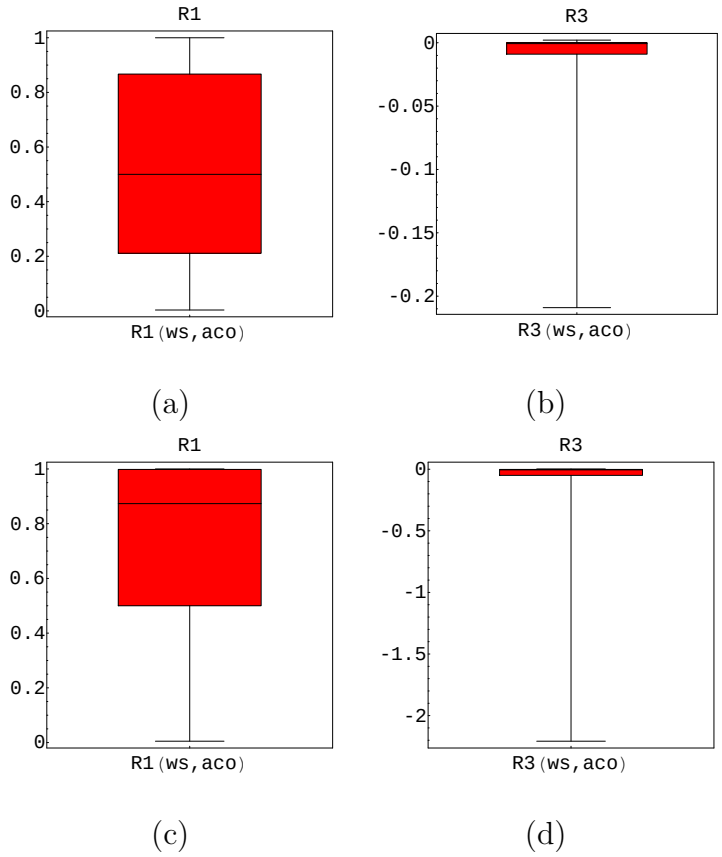


Figure 5.9: Box-and-whisker plots for $R1$ and $R3$ metrics comparing Weighted Sum (ws) and MONACO (aco) for network with: (a)–(b) 50 nodes, and (c)–(d) 100 nodes.

the interquartile value range, equal to 0.0056. According to Table 5.6, if we restrict the analysis to the 104 networks that used the CWG then $\mu_{R1(ws,aco)} = 0.1731$ and $\mu_{R3(ws,aco)} = 0.0324$.

In general, $\mu_{R3(ws,aco)} = -0.0278$ and $\sigma_{R3(ws,aco)} = 0.1543$ values indicate that the approximation sets returned by MONACO are relatively similar to the ones produced by the Weighted Sum method, which only returns Pareto solutions. However, it is possible to observe that quality of the results returned by MONACO diminish as the size of the networks increase, which indicates that possible alternatives should be considered, besides the possible extra computational time.

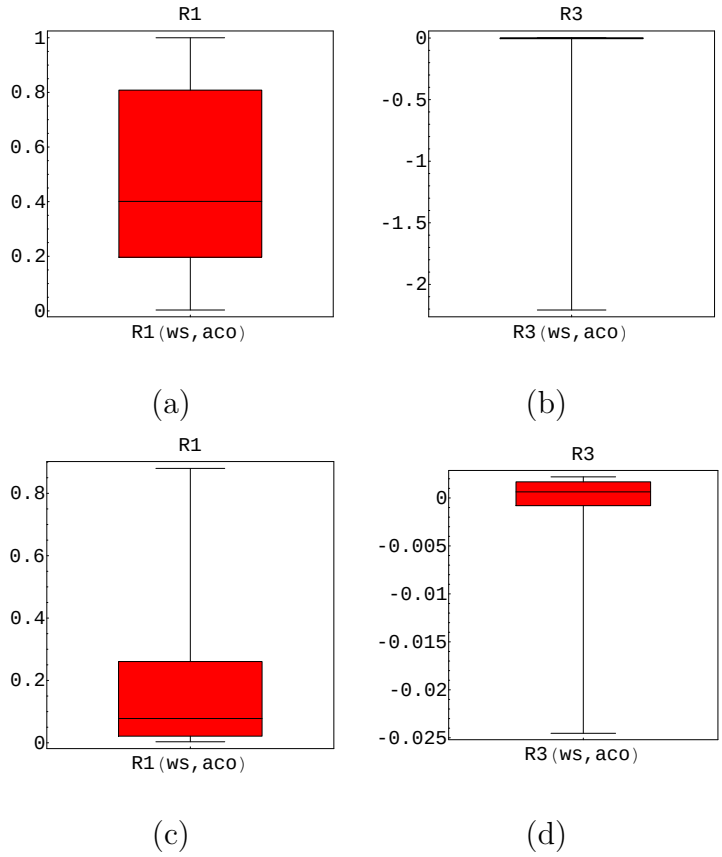


Figure 5.10: Box-and-whisker plots for the $R1$ and $R3$ metrics comparing Weighted Sum (ws) and MONACO(aco) for: (a)-(b) all 562 networks; (c)-(d) the 104 networks that used the CWG.

5.3 ϵ-DANTE Algorithm

The previous sections analysed the results returned by MONACO. It was possible to conclude that, although the approximations sets returned by MONACO were, in most of the cases, comparable to the used reference sets, alternatives to improve the quality of the solutions should be thought.

As in most of the Swarm Intelligence processes, MONACO builds a solution, evaluates it, performs the pheromone updating process and discards it (recall Algorithm 1). This strategy does not allow a proper local exploration, overlooking the computational effort that was necessary to produce it. Furthermore, as pointed by Dorigo & Stützle [1999]

	CWG		$\rho - CWG$ $\rho \geq 0$		$\rho - CWG$ $\rho < 0$		RWG	
	μ	σ	μ	σ	μ	σ	μ	σ
$R1$	0.1731	0.2137	0.4749	0.2589	0.6036	0.3476	0.5706	0.3399
$R3$	0.0324	0.1888	- 0.1052	0.2433	- 0.1476	0.3107	- 0.0723	0.1454
#	104		205		184		69	

Table 5.6: Mean (μ) and standard deviation (σ) values for the $R1$ and $R3$ metrics for all networks according to the weights generators (# – number of networks in each class).

> "(...) construction algorithms are typically the fastest approximate methods, but the solutions they generate often are not of a very high quality and they are not guaranteed to be optimal with respect to small changes; the results produced by constructive heuristics can often be improved by local search algorithms.(...)"

In fact, to further explore the achieved solutions, it is common to use hybrid algorithms that apply local optimizers to the solutions obtained with the construction algorithms, as

- The 2-opt or 3-opt in the Travelling Salesman Person [Dorigo & Stutzle, 2004];
- The SOP-3-exchange for the Sequential Ordering Problem [Gambardella & Dorigo, 2000]; or
- The iterated local search for the Bin Packing Problem [Levine & Ducatelle, 2004].

The ϵ-`DANTE` process was developed with the objective of exploring the more promising constructed solutions. The method is characterized by a restricted depth search

Algorithm 16 ϵ-DANTE Algorithm

```
1: Initialize the pheromone trail.
2: while stopping criterion is not met do
3:     for all ants do
4:         Construct a new solution, S, using the current pheromone trail.
5:         if the distance of S to the approximation set is inferior to ε or S improves
               the approxiamtion set then
6:             Apply a depth search procedure from that solution.
7:         end if
8:     end for
9:     Update the pheromone trail.                                  ▷ Optional
10: end while
```

based on the pheromone vectors. In other words, whenever a solution is attractive, it is performed a depth search procedure limited in the number of branches by level and/or in the available computational requirements.

Algorithm 16 presents a high level description of the method. The main procedure is composed by a set of iterations. In each of these iterations, a set of agents construct solutions using the current pheromone trails. If the solutions improve the approximation set or its distance to that set is not superior to a ϵ criterion parameter then it is applied the restricted depth search method. Steps 4 - 6 of the algorithm will be described with more detail in the next section. Note that step 9 is optional, since the pheromone update can be performed in the previous steps, as we will see later.

5.3.1 Depth Search Exploration

As referred, the fitness of each generated solution, S, is compared with the fitness of the elements in the approximation set that is in construction. Then the limited depth search is made to

Level D – If S improves $\mathcal{P}$, that is, S is not dominated by any element of $\mathcal{P}$ ($\nexists_{T \in \mathcal{P}} : T \prec S$); or

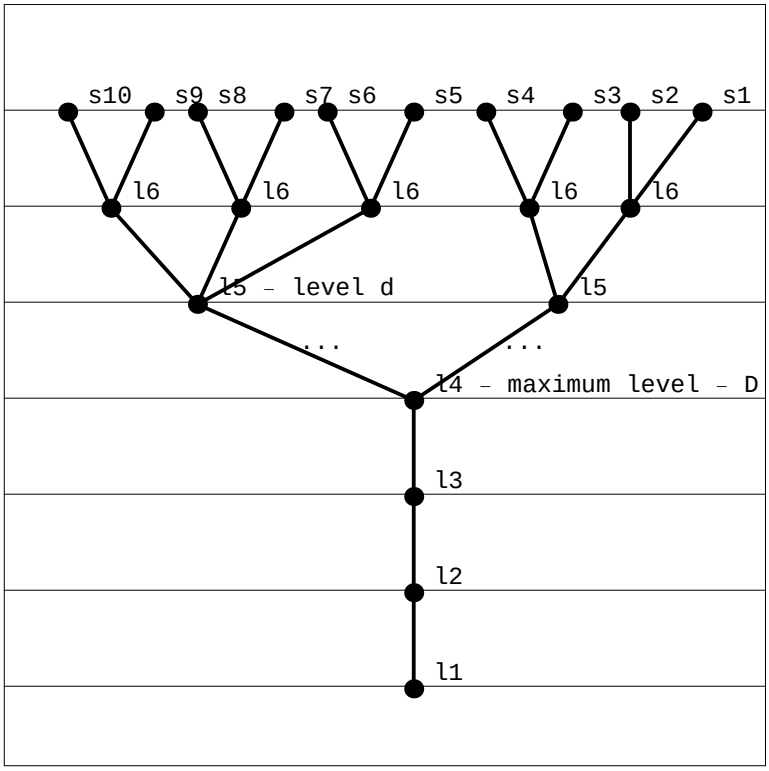


Figure 5.11: Example of search tree.

Level d – If S is dominated by some element T of $\mathcal{P}$ but its relative distance to the approximation set is smaller than ϵ, which includes the case where $\mathcal{W}(T) = \mathcal{W}(S)$.

Here d and D are algorithm parameters and should verify $d > D$. Furthermore, the number of branches used in each level of the depth search, M, it is also an algorithm parameter.

Figure 5.11 delineates the search tree for a process with $M = 1$, $d = 5$ and $D = 4$. Each edge of the tree represents the addition of one component of the solution (ϵ-DANTE is a construction algorithms) and the leaves, labelled as s_k, are the achieved solutions.

Algorithm 17 presents a high level description of the process where

- The algorithm contains a tabu list for each level, $tabu_l$, that is initialized as the empty set. This tabu list avoids that the same solutions are rebuilt in the depth search, by restricting the addition of the same edges in the same level. The use

of the tabu list is version dependent and should be thought according to step 31, since it also restricts the construction of new solutions containing those edges.

- In step 4 the computed distance from T to the approximation set $\mathcal{P}$ should return a negative value if T improves $\mathcal{P}$; and
- In step 6 the update procedure consists in inserting T in $\mathcal{P}$ and removing the elements of $\mathcal{P}$ dominated by T.
- Steps 7 and 10 are optional since, if all solutions contribute to the variation of the pheromones, the result is a noisy trail with consequent lost of performance.

5.3.2 Angle-Pheromones Update Strategy

With ϵ-DANTE it is also proposed a new strategy for updating the pheromone vectors. This update is particular to the multiple objective problems since, as before, it is supposed that to each edge is associated a m-vector of values and each component of the vector represents the worthy of that edge in the construction of good solutions, considering a particular weight.

The Angle-Pheromone Update strategy can be considered greedy in the sense that it only uses elements of the approximation set. The objective is to explore small regions of the search space by using, in the pheromone updating formula, only a subset of the solutions contained in that set. This idea is motivated by the fact that, in most of the cases, the number of elements in the approximation set becomes very large. Empirical tests proved that if all the solutions in the approximation set were used, the pheromone based selection becomes very noisy, which delays the convergence toward the Pareto set.

Algorithm 17 ϵ-DANTE's solutions exploration.

1: **function** ϵ-DANTE_SOLUTIONS(T)
2: $l \leftarrow |T|$ ▷ *defines the level by the number of added edges*
3: **if** T is a solution **then**
4: Set Δ as the relative distance from T to $\mathcal{P}$
5: **if** $\Delta < 0$ **then** ▷ T improves $\mathcal{P}$
6: Update the approximation set with T
7: Update Δ_k $(k = 1, 2, \ldots, m)$ with T ▷ *Optional*
8: **return** D
9: **else**
10: **if** $\Delta < \epsilon$ **then** Update Δ_k $(k = 1, 2, \ldots, m)$ with T ▷ *Optional*
11: **return** d
12: **else return** 0
13: **end if**
14: **end if**
15: **else**
16: $SearchLevel \leftarrow 0$, $NumberOfBranches \leftarrow M$
17: **for** $k = 1$ to $NumberOfBranches$ **do**
18: **if** $\exists_{e \in \mathcal{E} - tabu_l} : T \cup \{e\}$ is admissible **then**
19: Choose an edge, e, from $\mathcal{E} - tabu_l$ ▷ *Problem specific*
20: $Tabu_l \leftarrow Tabu_l \cup \{e\}$ ▷ *Optional*
21: $T \leftarrow T \cup \{e\}$
22: $L \leftarrow \epsilon$-DANTE_SOLUTIONS(T) ▷ *Recursive call*
23: **if** $L > 0$ **then**
24: $NumberOfBranches \leftarrow NumberOfBranches + M$
25: $SearchLevel \leftarrow \max\{SearchLevel, L\}$
26: **end if**
27: $T \leftarrow T - \{e\}$
28: **else** Go to line 31
29: **end if**
30: **end for**
31: $Tabu_l \leftarrow \emptyset$ ▷ *Optional (clean the tabu list)*
32: **return** $\max\{SearchLevel - 1, 0\}$
33: **end if**
34: **end function**

Therefore, the pheromone vector update is made according to formula

$$\tau(e) = \rho\tau(e) + \Delta(e), e \in \mathcal{E}, \tag{5.12}$$

where

- $\tau(e)$ is pheromone vector associated to edge e;
- $\rho \in [0, 1]$ is called the persistence factor ($1 - \rho$ is the evaporation factor). The smaller the values of ρ are, the smaller quantity of information, used in one cycle, is transmitted to following cycle;
- $\Delta(e) = (\Delta_1(e), \Delta_2(e), \ldots, \Delta_m(e))$ is the reinforcement pheromone vector associated to edge e and is computed using the elements of the approximation set, $\mathcal{P}$, and formula

$$\Delta_k(e) = \sum_{T \in \mathcal{P}_e} \frac{Q}{w_k(T)}, \tag{5.13}$$

where

  - Q is a value with the same magnitude of the solutions. For example, if the weights are balanced it can be used the average of the minimum weights,

$$\frac{1}{m} \sum_{k=1}^{m} \min_{T \in \mathcal{P}} w_k(T); \tag{5.14}$$

  - $\mathcal{P}_e$ are the elements of the approximation set that contain edge e and lie in a subangle of the angle defined by the origin and the extreme solutions (of weights k and $k + 1$), that is,

$$\mathcal{P}_e = \left\{ T \in \mathcal{P} : e \in T \wedge \phi_k^{\min} + I_k \phi_k^h \leq \phi_k(T) \leq \phi_k^{\min} + (I_k + 1)\phi_k^h \right\} \tag{5.15}$$

where

$$\phi_k(T) = \arctan\left(\frac{w_{k+1}(T)}{w_k(T)}\right), \tag{5.16}$$

$$\left\{\begin{array}{lcl} \phi_k^{\min} & = & \min_{T\in\mathcal{P}} \phi_k(T) \\ \phi_k^{\max} & = & \max_{T\in\mathcal{P}} \phi_k(T) \end{array}\right. , k = 1, 2, \ldots, m-1, \tag{5.17}$$

ϕ_k^h is the step interval and I_k is a value related to the region to be explored which is controlled by the main process.

For example, Figure 5.12 sketches one of those angles for the bi-objective case. In this figure, the black dots are the elements of the approximation set $\mathcal{P}$, the light gray zone is the region between angles $\phi_k^{\min}$ and $\phi_k^{\max}$, and the dark gray zone is the region between

$$\phi_k^{\min} + I_k\phi_k^h \leq \phi_k(T) \leq \phi_k^{\min} + (I_k + 1)\phi_k^h. \tag{5.18}$$

The elements of $\mathcal{P}_e \subset \mathcal{P}$ in the dark gray region are the elements that will be used in equation (5.13).

5.3.3 Test Cases

ϵ-DANTE algorithm was tested with three multiple objective problems: the Minimum Spanning Tree, the k-Degree Minimum Spanning Trees and the Travelling Salesman Person problems. The first problem is studied in detail in the next chapter. The choice of the other two problems is justified by the existence of results for sets of instances, obtained by other authors with other evolutionary techniques, which will allow to compare the performance of the proposed algorithm.

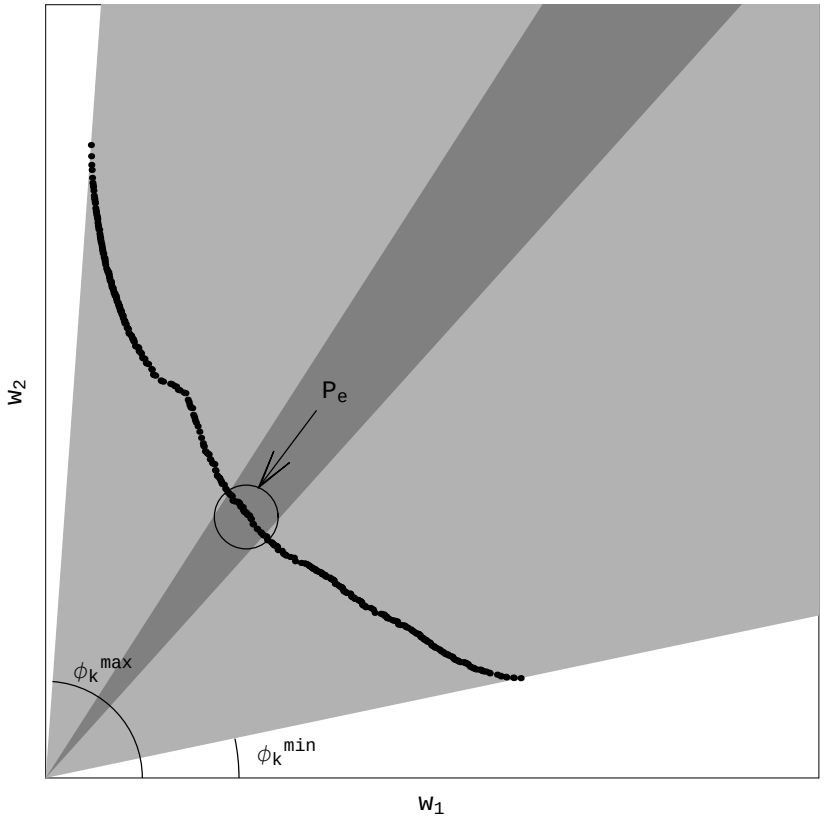


Figure 5.12: Pheromone vectors update strategy.

Multiple Objective k-Degree Minimum Spanning Trees problem

A k-Degree Minimum Spanning Tree is a minimal weight spanning tree such that the maximum degree of any node is k. In the proposed process, to build the spanning tree, feasible edges are successively added until a solution is formed. Before the addition of any of those edges it must be verified that such edge will not form a cycle neither the nodes maximum degree condition is violated.

More precisely, the process starts by randomly selecting a node from $\mathcal{V}$, s, and setting $T_N = \{s\}$ (T_N is the set of the nodes already included it the tree). Then sequentially are added $n-1$ admissible edges from

$$\mathcal{A} = \{e_{uv} \in \mathcal{E} : u \in T_N \wedge \delta(u) < k \wedge v \in \mathcal{V} - T_N\}, \tag{5.19}$$

where δ is the degree of node u in the subtree that is being constructed and k the

maximum degree allowed. The selection of the edges is pseudo-randomly made using formula

$$e_{st} = \begin{cases} \arg\max_{e_{s't'} \in \mathcal{A}} \left(\prod_{j=1}^{m} \tau_j(e_{s't'})^{\alpha_j} w_j(e_{s't'})^{-\beta_j} \right) & \text{if } q \leq q_0 \\ e & \text{if } q > q_0, \end{cases} \tag{5.20}$$

where

- $\tau_j(e)$ is the pheromone value associated to the j weight in edge e;
- $w_j(e)$ is the j-weight of edge e;
- α_j is an algorithm parameter associated to the relevance of weight j;
- β_j is an algorithm parameter associated to the local heuristic that favours edges with lower j-weight;
- $e \in \mathcal{A}$ is an edge pseudo-randomly chosen with probability

$$p(e) = \frac{\prod_{j=1}^{m} \tau_j(e)^{\alpha_j} w_j(e)^{-\beta_j}}{\sum_{f \in \mathcal{A}} \prod_{j=1}^{m} \tau_j(f)^{\alpha_j} w_j(f)^{-\beta_j}}. \tag{5.21}$$

- q is a uniform random value in $[0, 1]$; and
- $q_0 \in [0, 1]$ is a parameter that influences which branch of (5.20) is used more often: a smaller value of q_0 produces a more exploratory search, since it implies the use of the pseudo-random formula (5.21) with higher probability. When q_0 is near 1, the feasible edge with larger probability of entering the tree is used with greater frequency, which suggest an exploiting search.

Example 5.3 sketches some steps of the ϵ-DANTE process applied to an instance of the problem in study.

Example 5.3: Consider the same premises of Example 5.2, that is, the considered network is represented in Figure 5.7 and its weights and pheromone values are in Table 5.3. Let us also suppose that $\alpha_1 = 1$, $\alpha_2 = 2$, $\beta_1 = 2$, $\beta_2 = 2$, $q_0 = 0$, and the maximum degree of the nodes is $k = 3$.

The construction and exploration process of a set of trees, by ϵ-DANTE, can be described as follows (in Figure 5.7 is also depicted the process, where the continuous lines are the edges already included in the tree and the dashed lines are the candidate edges to enter the tree):

(a) Randomly select a node. Node 1 is selected. The probabilities of adding each of the edges with starting node 1 are

$$p_{1,2} = \frac{\left(\frac{1}{1}\right)^2 5 \left(\frac{1}{2}\right)^2 4^2}{\left(\frac{1}{1}\right)^2 5 \left(\frac{1}{2}\right)^2 4^2 + \left(\frac{1}{3}\right)^2 1 \left(\frac{1}{3}\right)^2 2^2 + \cdots + \left(\frac{1}{7}\right)^2 1 \left(\frac{1}{5}\right)^2 2^2} = 0.452,$$

$p_{1,3} = 0.001$, $p_{1,4} = 0.090$, $p_{1,5} = 0.452$, $p_{1,6} = 0.005$, $p_{1,7} = 0.0$ and $p_{1,8} = 0.0$. Edge e_{12} is added, Figure 5.13 (a)-(b);

(b) From nodes in $\{1, 2\}$ the probabilities are $p_{1,3} = 0.001, p_{1,4} = 0.055, p_{1,5} = 0.276, p_{1,6} = 0.003, p_{1,7} = 0, p_{1,8} = 0, p_{2,4} = 0.662$, and $p_{2,5} = 0.003$. Edge e_{24} is selected, Figure 5.13 (b)-(c);

(c) From nodes in $\{1, 2, 4\}$ the probabilities are $p_{1,3} = 0.001$, $p_{1,5} = 0.345$, $p_{1,6} = 0.004$, $p_{1,7} = 0$, $p_{1,8} = 0$, $p_{2,5} = 0.004$, and $p_{4,6} = 0.647$. Edge e_{46} is selected. Note that edge e_{14} is not allowed since if it was added it would create a cycle, Figure 5.13 (c)-(d);

(d) From nodes $\{1, 2, 4, 6\}$ the probabilities are $p_{1,3} = 0.002, p_{1,5} = 0.907, p_{1,7} = 0, p_{1,8} = 0, p_{2,5} = 0.01$, and $p_{6,3} = 0.081$. Edge e_{15} is added, Figure 5.13 (d)-(e);

(e) From nodes $\{1, 2, 4, 5, 6\}$ the probabilities are $p_{1,3} = 0.001$, $p_{1,8} = 0$, $p_{5,7} = 0.945$,

$p_{1,7} = 0$, and $p_{6,3} = 0.054$. Edge e_{57} is selected, Figure 5.13 (e)-(f);

(f) From nodes $\{1, 2, 4, 5, 6, 7\}$ the probabilities are $p_{1,3} = 0.013$, $p_{1,8} = 0.001$, $p_{3,6} = 0.464$, and $p_{7,8} = 0.522$. Edge e_{18} is selected, Figure 5.13 (f)-(g);

(g) From nodes $\{1, 2, 4, 5, 6, 7, 8\}$ the probabilities are $p_{3,6} = 0.066$ and $p_{3,8} = 0.934$. Edge e_{38} is added. Note that edge e_{13} is not considered since the degree of node 1 is already $k = 3$, Figure 5.13 (g)-(h);

(h) The construction of the spanning tree is complete, Figure 5.13 (h). If this tree enters the approximation set, then it is made the depth search process as follows;

(i) Edge e_{38} is removed. From nodes $\{1, 2, 4, 5, 6, 7, 8\}$ only the edge e_{36} can be added since edge e_{38} has already been used in this level and therefore is in the tabu list, Figure 5.13 (i);

(j) It is obtained a new tree, Figure 5.13 (j);

(k) Edge e_{36} is removed, but there are no other feasible edge to be added, Figure 5.13 (k);

(l) Edge e_{18} is removed. From nodes $\{1, 2, 4, 5, 6, 7\}$ the probabilities are $p_{1,3} = 0.024$ and $p_{7,8} = 0.976$. Note that edge e_{13} can now be used since the degree of node 1 is two and edge e_{18} cannot be added since it is tabu, Figure 5.13 (l).

The process would continue until some stopping criteria was met like the non-existence of feasible edges to continue the search, the full exploration of the depth levels or some other computational restriction.

The algorithm was implemented in $C++$ and tests were run on a PC with Intel Pentium^{TM}IV $3Ghz$ processor, $512Mb$ of RAM and Windows XP OS. For each problem the method was run 15 times with parameters in Table 5.7.

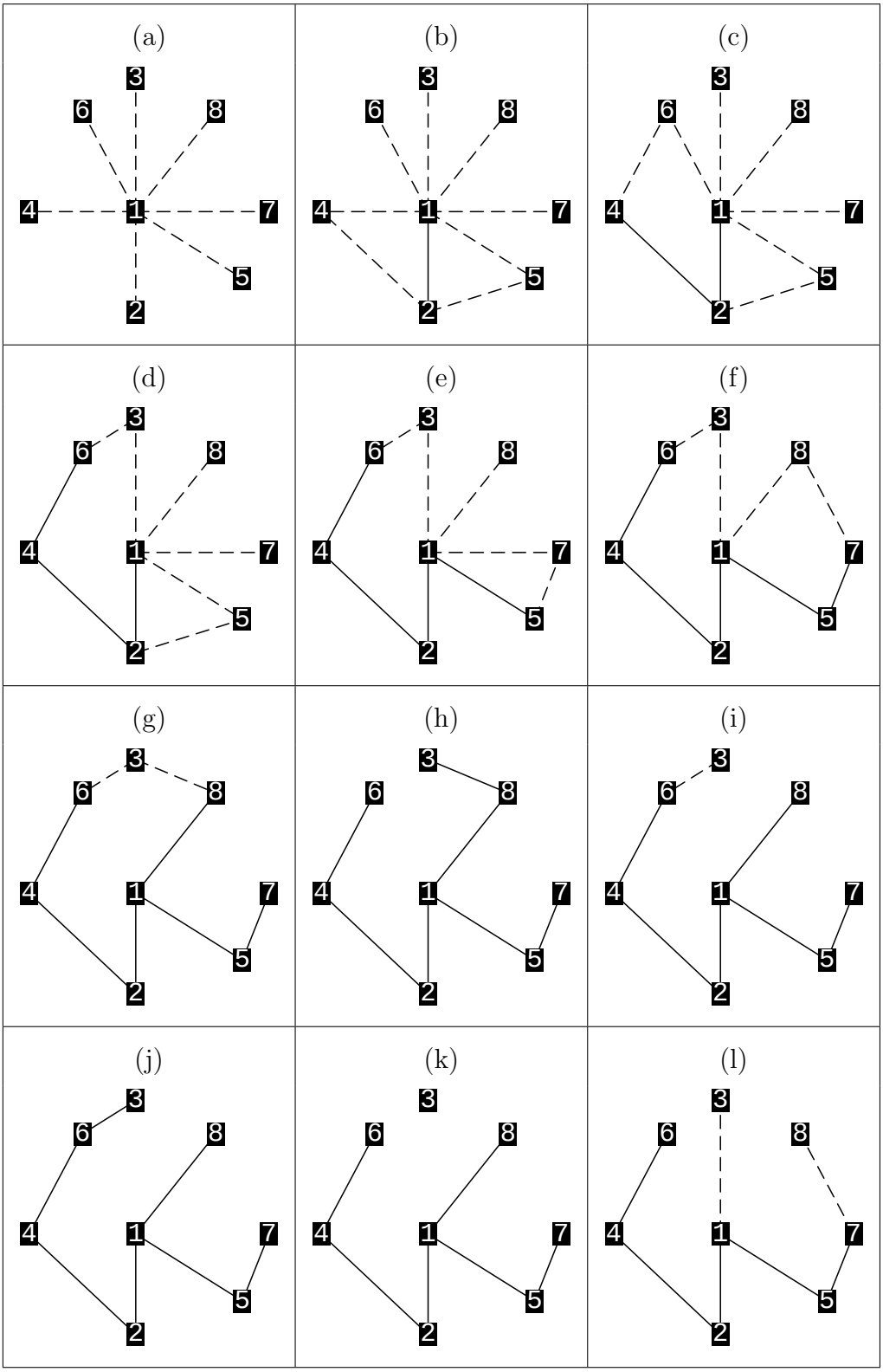


Figure 5.13: Simulation of the building trees process.

Parameter	Values	Comments
D	1	Maximum depth
d	$\lfloor \frac{3\|\mathcal{V}\|}{4} \rfloor$	Intermediate depth
M	2	Number of branches
ϵ	0	
α_i, β_i	$\{0, 0.03, 0.06, \ldots, 3.0\}$	
ρ	0.1	
q_0	$\{0.5, 0.6, 0.7, 0.8, 0.9\}$	Set randomly for each ant
k	3	Maximum of the nodes degrees
Number of ants per cycle	$\|\mathcal{V}\|$	
Number of cycles	2	
Maximum run time	$\min\{60\|\mathcal{V}\| \log \|\mathcal{V}\|, 36000\}$	Seconds

Table 5.7: Used parameters.

The problems instances that were used to test the implementation were defined by Knowles & Corne [2001a] and the ϵ-DANTE solutions are compared with the solutions obtained with the Genetic Algorithms presented by the same authors. More specifically, it was used as reference set the one composed by the non-dominated elements of the union of the 30 runs made with their Genetic Algorithm (except for the instances with 10 nodes where it was used the exact Pareto set, marked with a "†").

Table 5.8 presents a resume of the results for the tested instances over the 15 runs. This table contains information about the average values of the metrics $R1$ and $R3$, the reference point used to compute the values of $R1$ and $R3$, the average cardinality of the approximation set built with ϵ-DANTE (in parentheses the cardinal of the reference set), the average time of the last update of the approximation set and a sketch of a

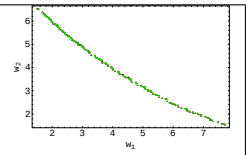
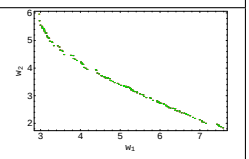
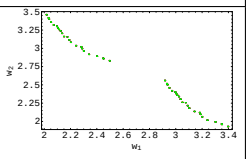
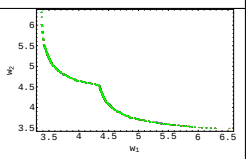
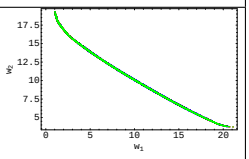
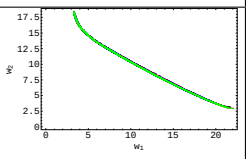
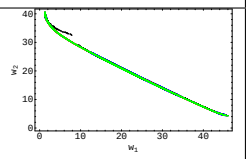
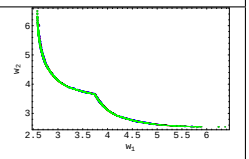
Problems	μ_{R1}	μ_{R3}	Reference point	$\mu_{\lvert\mathcal{P}\rvert}$ ($\mu_{\lvert\mathcal{P}_{ref}\rvert}$)	μ_{Time}	Sketch of a typical front
10vac	0.57	0.00	(1.47793, 1.51809)	183 (191†)	7.6	
10v-m-c	0.5	0.00	(2.92446, 1.81862)	129 (129†)	628	
10vconc	0.54	0.00	(1.99411, 1.90807)	128 (134†)	8	
25vac	0.47	0.00	(3.33481, 3.45579)	517 (439)	3754	
25vc	0.39	0.00	(0.964108, 3.64902)	2496.6 (1480)	4351.	
25v-m-c	0.13	0.01	(3.23334, 2.92848)	2168.7 (820)	4743	
50vac	0.40	-0.05	(1.16971, 4.18926)	6328.1 (1436)	11960	
50vconc1	0.48	0.00	(3.33481, 3.45579)	1748.6 (894)	11803	

Table 5.8: Resume of the results for the k-Degree Minimum Spanning Tree problem.

typical front. From the same table it is possible to observe that in most of the cases, ϵ-DANTE improves the reference set since $R1 < 0.5$ and $R3 \approx 0$. The exceptions were the 10 nodes networks where in one case it has achieved the exact Pareto set (obtained

with the Brute Force method) and in the other two cases it has obtained 128 and 183 of the 134 and 191 solutions, respectively.

The time evolution of the $R1$ and $R3$ metrics are depicted in Figures 5.14 and 5.15, respectively. It is possible to observe that the convergence of the solutions toward the reference set is made quite fast since $R3$ quickly takes values near to zero. Nevertheless, the local refinement takes some extra time which, as already referred, is a characteristic common to most of the Ant Colony based algorithms. If some extra local optimizers are applied to the solutions the convergence of the approximation set toward the Pareto set would probably be achieved more rapidly.

Multiple Objective Travelling Salesman Person Problem

In the implemented process each agent is initially placed at random in one of the nodes. Then each of those agents build a Hamiltonian cycle by the successive inclusion of nodes from the ones that were not yet visited, and are adjacent to the last one inserted. If the distance from the constructed solution to the approximation set is not superior to a ϵ parameter or the solution improves the approximation set then, it is performed the pheromone oriented depth search procedure as described in Section 5.3.1. In any of the cases, the order in which the nodes are chosen is made pseudo randomly according to the pheromone trails and a local greedy heuristic that gives preference to the "nearest" nodes.

Mathematically, if the current node is s and T_N is the set of nodes already included in the path that is being constructed, then an edge e_{st} is chosen to integrate the path (and consequently node t is added) accordingly to

$$e_{st} = \begin{cases} \arg\max_{e_{st'} \in \mathcal{A}_s} \left(\prod_{k=1}^{m} \tau_k(e_{st'})^{\alpha_k} w_k(e_{st'})^{-\beta_k} \right) & \text{if } q \leq q_0 \\ e & \text{if } q > q_0 \end{cases} \tag{5.22}$$

where

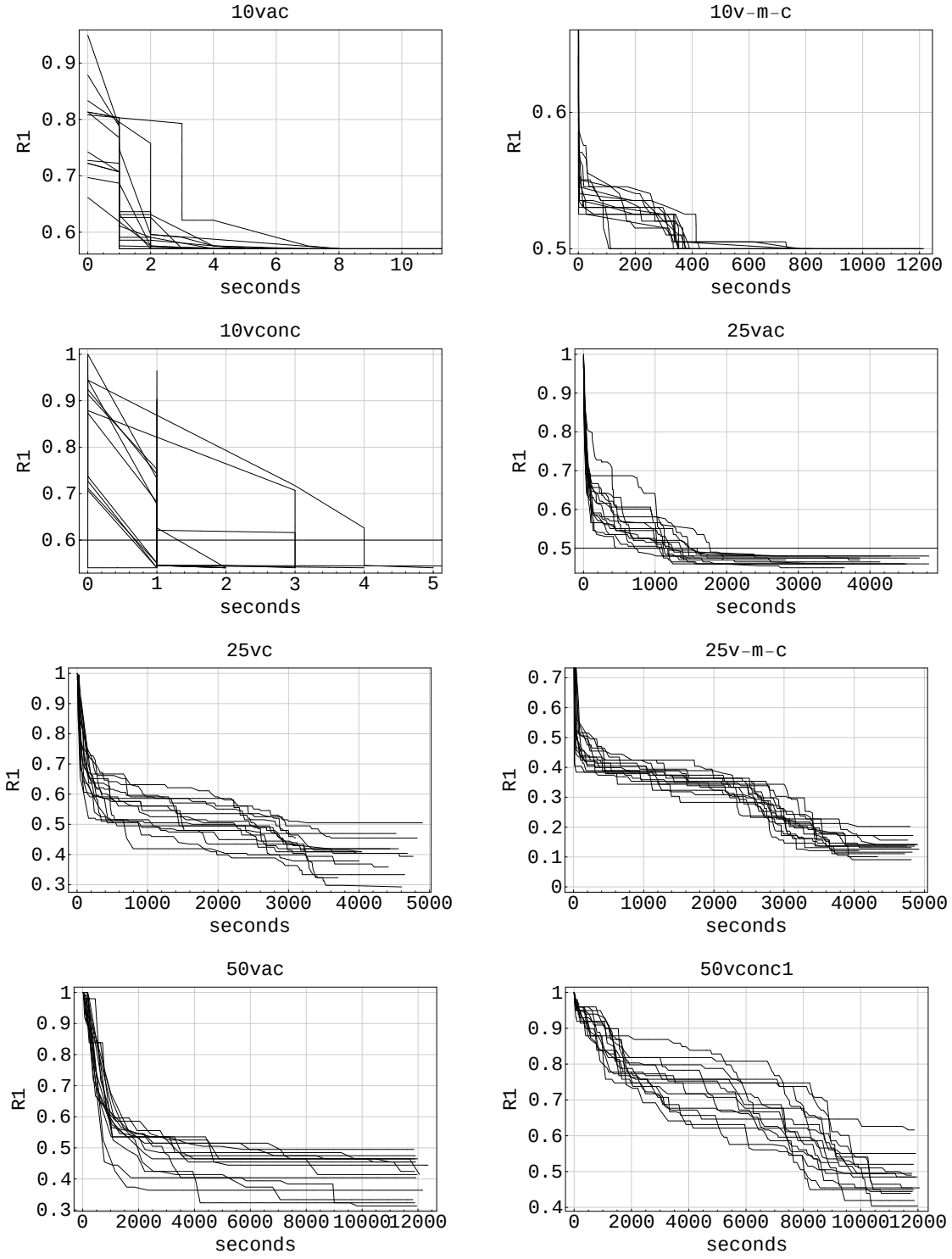


Figure 5.14: Time evolution for $R1$ metric over the 15 runs.

- $\mathcal{A}_s = \{e_{st'} \in \mathcal{E} : t' \notin T_N\}$

- t is the next node to be inserted in the path;

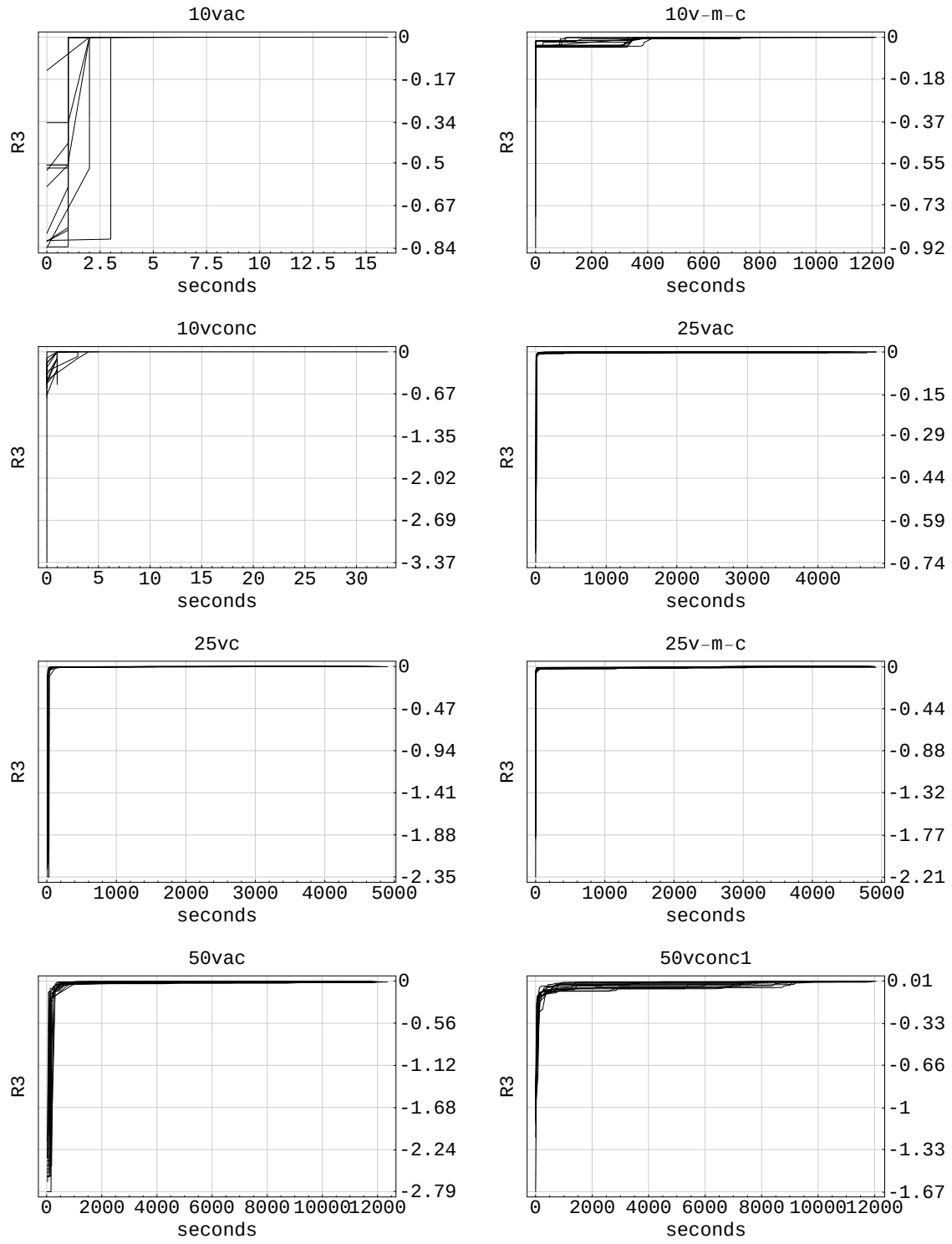


Figure 5.15: Time evolution for $R3$ metric over the 15 runs.

- $\tau_k(e)$ is the pheromone value associated to the k weight in edge e;
- $w_k(e)$ is the k-weight of edge e;

- α_k is an algorithm parameter associated to the relevance of weight k;
- β_k is an algorithm parameter associated to the local heuristic that favors edges with lower k-weight;
- $e \in \mathcal{A}_s$ is an edge pseudo-randomly chosen with probability

$$p(e) = \frac{\prod_{k=1}^{m} \tau_k(e)^{\alpha_k} w_k(e)^{-\beta_k}}{\sum_{f \in \mathcal{A}_s} \prod_{k=1}^{m} \tau_k(f)^{\alpha_k} w_k(f)^{-\beta_k}}; \tag{5.23}$$

- q is a uniform random value in $[0, 1]$; and
- $q_0 \in [0, 1]$ is a parameter that favours the exploration of the search space (for smaller values of q_0) or the exploitation of that same search space (for larger values of q_0).

This process is repeated $n - 1$ times. The Euclidean cycle is concluded by the addition of the edge defined by the first and last nodes inserted in the path.

The algorithm was developed in $C++$ and a set of runs were made over a PC with Intel Pentium$^{\text{TM}}$IV $3Ghz$ processor, $512Mb$ of RAM and Windows XP OS. For each problem the method was run 12 times with parameters showed in Table 5.9.

Table 5.10 resumes the results for the 12 runs where

- μ_{R1}, μ_{R3} are the mean values of the $R1$ and $R3$ metrics;
- The reference point column contains the points used in (2.57) and (2.59) to compute the values of $R1$ and $R3$;
- $\mu_{|\mathcal{P}|}$ is the mean value of the cardinality of the approximation set;
- $\mu_{|\mathcal{P}_{ref}|}$ is the cardinality of the reference set; and
- μ_{Time} is the average time (in seconds) of last update of the approximation set.

Parameter	Values	Comments
D	$\lfloor \frac{\lvert\mathcal{V}\rvert}{2} \rfloor$	Maximum depth
d	$\lfloor \frac{3\lvert\mathcal{V}\rvert}{4} \rfloor$	Intermediate depth
M	1	Number of branches
ϵ	0	
α_i, β_i	$\{0, 0.03, 0.06, \ldots, 3.0\}$	
ρ	0.1	
q_0	$\{0.5, 0.6, 0.7, 0.8, 0.9\}$	Set randomly for each ant
Number of ants per cycle	$\lvert\mathcal{V}\rvert$	
Number of cycles	10	
Maximum run time	10	hours

Table 5.9: Used parameters.

The instance problems considered were *kroab*50, *kroac*50, *kroad*50, *kroae*50, *krobc*50, *krobd*50, *krobe*50, *kroab*50, *krocd*50, *krocd*50 and *krode*50 (all available at [Jaszkiewicz, 2006b; TSPLib, 2003]). The used reference set was computed as the non dominated set of the union of the results obtained with 5 runs for each of the Multiple Objective Genetic Local Search (MOGLS), Multiple Objective Simulated Annealing (MOSA) and MOSA-like MOGLS algorithms [Jaszkiewicz, 2002, 2006a; Shibuchi & Murata, 1998].

The values presented in Table 5.10, where $R1 < 0.5$ and $R3 \geq 0$, indicate that ϵ-`DANTE` improved the reference sets. Furthermore, the cardinality of the approximation set is always superior to the reference set.

Some examples of the time evolution of the $R1$ and $R3$ metrics for the problems in study are sketched in Figures 5.16 and 5.17. It is observable that most of them attain a $R1$ value near to 0.5 after approximately one hour of computation. Furthermore, Figure 5.17 shows that the convergence to the reference set is much faster since $R3$

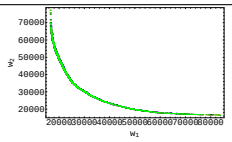
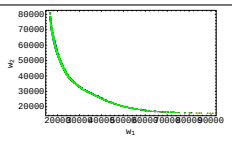
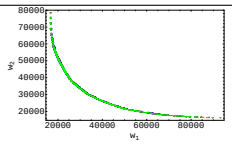
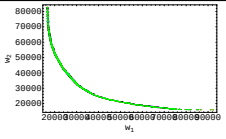
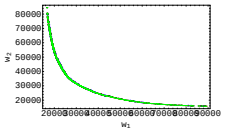
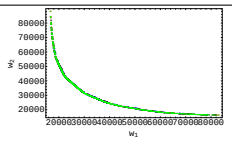
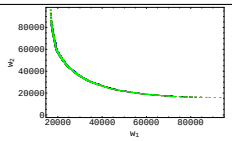
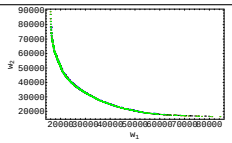
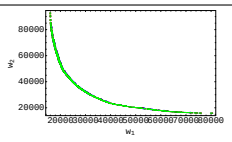
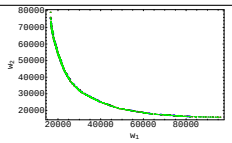
Problems	μ_{R1}	μ_{R3}	Reference point	$\mu_{\lvert\mathcal{P}\rvert}$ ($\mu_{\lvert\mathcal{P}_{ref}\rvert}$)	μ_{Time} (sec)	Sketch of a typical front
*kroab*50	0.20	0.001	(16296.4, 16354.8)	811 (414)	33294	
*kroac*50	0.28	0.002	(16296.4, 15614.3)	677 (291)	34799	
*kroad*50	0.32	0.001	(16296.4, 16155.8)	520 (363)	34980	
*kroae*50	0.35	0.001	(16296.4, 15751.9)	573 (342)	33979	
*krobc*50	0.40	0.000	(16354.8, 15614.3)	622 (369)	35134	
*krobd*50	0.31	0.001	(16354.8, 16155.8)	678 (410)	349336	
*krobe*50	0.40	0.000	(16354.8, 15751.9)	529 (330)	344046	
*krocd*50	0.30	0.001	(15614.3, 16155.8)	736 (403)	35429	
*kroce*50	0.25	0.002	(15614.3, 15751.9)	624 (328)	34880	
*krode*50	0.37	0.001	(16155.8, 15751.9)	499 (313)	35121	

Table 5.10: Resume of the results for the Travelling Salesman Person problem.

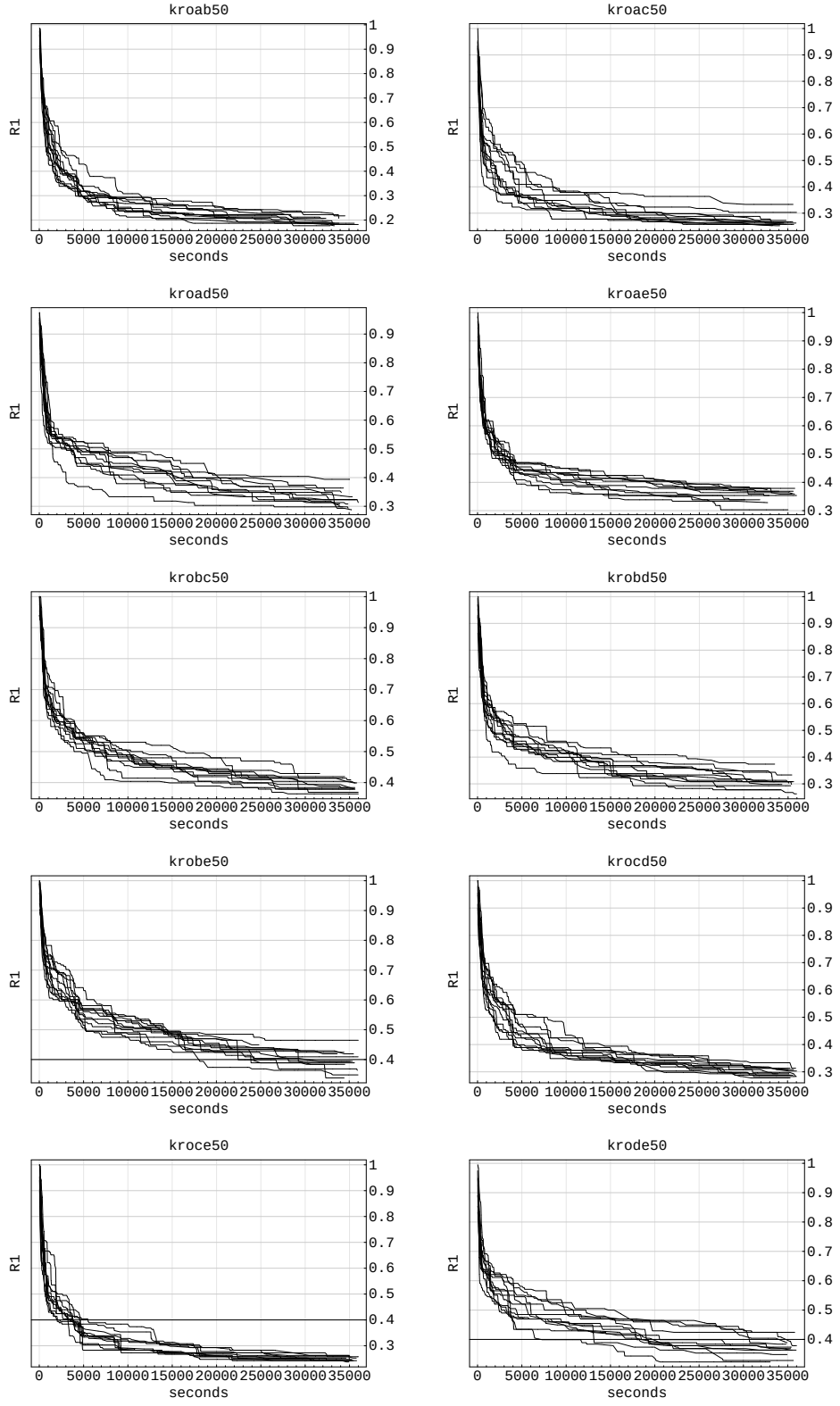


Figure 5.16: Time evolution for $R1$ metric over the 12 runs.

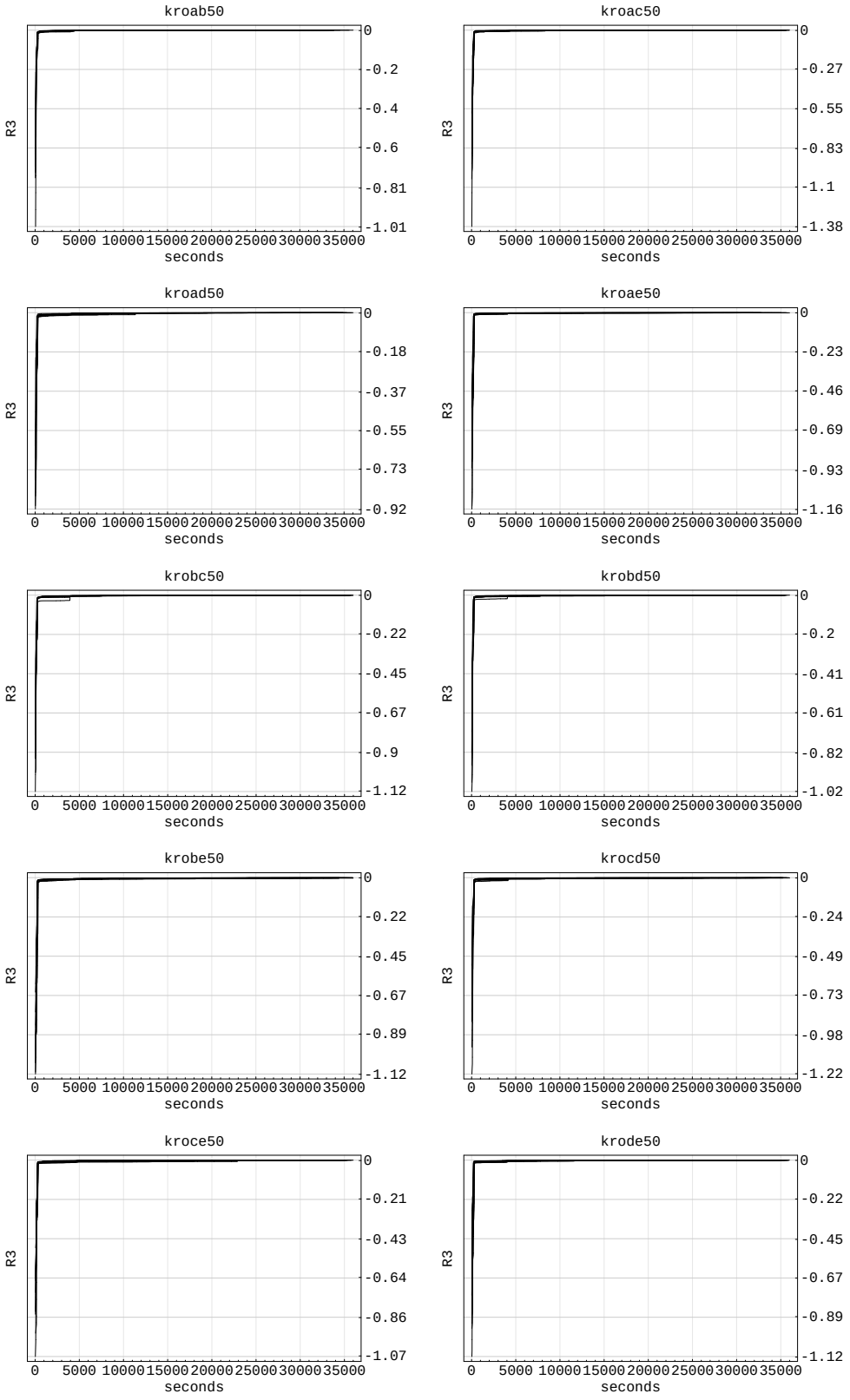


Figure 5.17: Time evolution for $R3$ metric over the 12 runs.

rapidly approximates 0.

Figure 5.18 sketches some examples of the graphics for

$$C(\mathcal{P}_1, \mathcal{P}_2, u_{\lambda,r}, r), \tag{5.24}$$

and

$$\frac{u^*_{\lambda,r}(\mathcal{P}_1) - u^*_{\lambda,r}(\mathcal{P}_2)}{u^*_{\lambda,r}(\mathcal{P}_1)}, \tag{5.25}$$

used to compute the values of $R1$ and $R3$ with $\lambda \in \{0.01, 0.02, \ldots, 0.99\}$ (refer to formulas (2.57) and (2.59)). The $R1$ graphics allow to infer where the solutions returned by ϵ-DANTE were better or worse than the solutions from the reference set, which is dependent of the utility function parameter, λ. On the other hand, the $R3$ gives the idea by how much one of the algorithms is better or worse than the others.

For example, for the λ parameter where $C(\mathcal{P}_1, \mathcal{P}_2, u_{\lambda,r}, r) = 0.5$ the (Tchebycheff) utility function returned equal utility values, and a better utility value for ϵ-DANTE when $C(\mathcal{P}_1, \mathcal{P}_2, u_{\lambda,r}, r) = 0$.

5.3.4 Algorithm Complexity

The overall complexity of the algorithm is strictly connected to the problem and the associated implementation. If Algorithm 16 has N_C cycles, each cycle uses N_A ants and, N_S is the computational requirements to compute a solution, then the process has

$$\mathcal{O}(N_C \times N_A \times N_S). \tag{5.26}$$

The computational complexity associated to the computation of a solution for the Multiple objective k-Degree Minimum Spanning Trees problem and the Multiple Objective Travelling Salesman Person problem are studied in the next two theorems.

Theorem 5.1. *Consider the implementation, in Section 5.3.3, for the Multiple Objective k-Degree Minimum Spanning Trees problem over a complete network instance.*

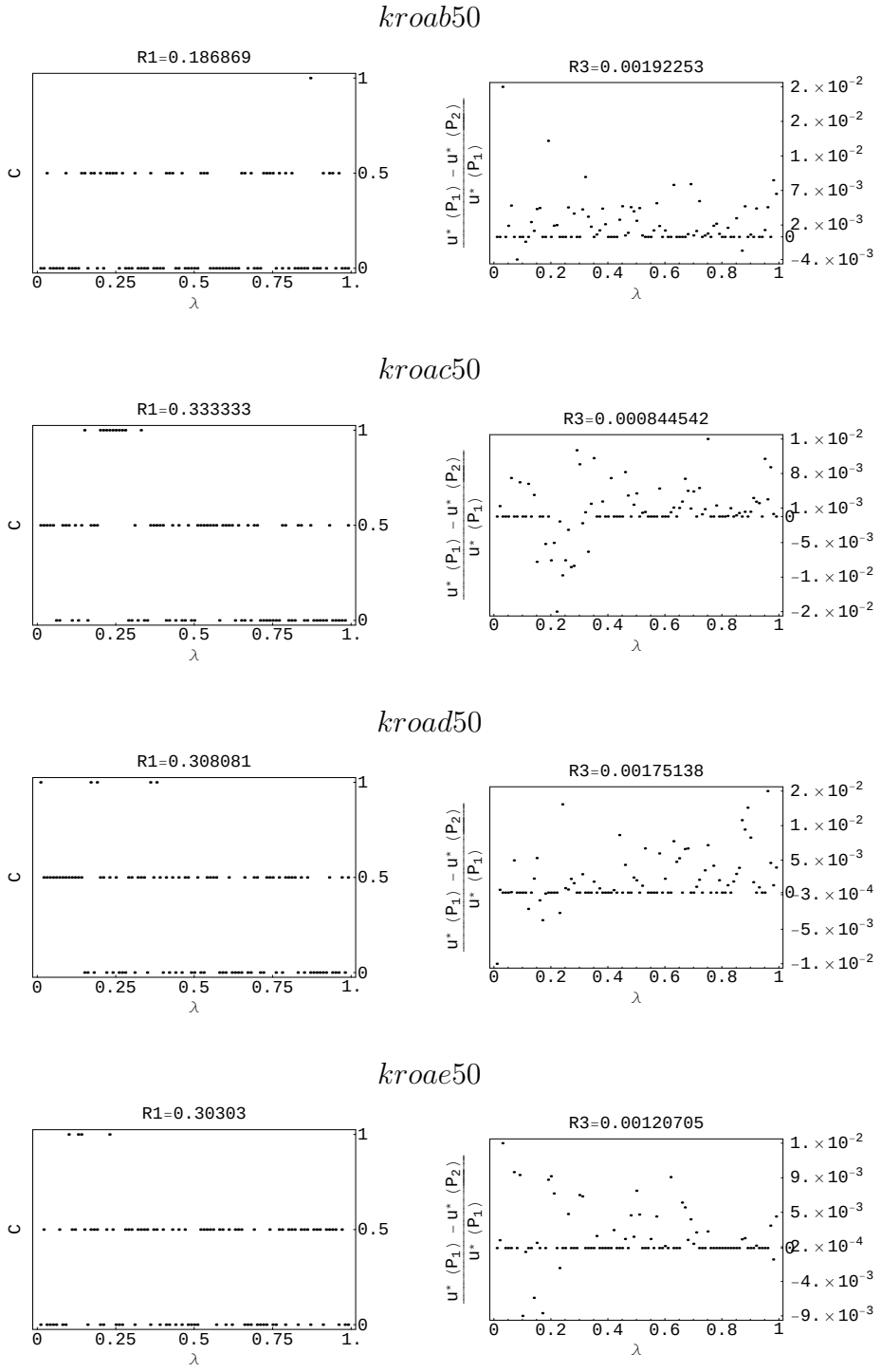


Figure 5.18: Graphics of $C(\mathcal{P}_{ref}, \mathcal{P}, u_{\lambda,r}, r)$ and $\frac{u^*_{\lambda,r}(\mathcal{P}_{ref}) - u^*_{\lambda,r}(\mathcal{P})}{u^*_{\lambda,r}(\mathcal{P}_{ref})}$ (functions used to compute the $R1$ and $R3$ metrics).

Suppose that k is large enough so that the k-degree restriction is never applied. Then the construction of tree has

$$\mathcal{O}\left(mn^2 + q_0 n^3 + (1-q_0)n^4\right) \tag{5.27}$$

complexity.

Proof. The values of the products present in (5.20) and (5.21) are calculated as a pre-computation, which takes $\mathcal{O}(mn^2)$. The main construction process start by randomly picking one node, n_1, and then it is chosen one edge, from the possible $n-1$. Using formula (5.20) it takes $\mathcal{O}(q_0(n-1) + (1-q_0)(n-1)^2)$ since

- The first branch of (5.20) is used with probability q_0 and takes $\mathcal{O}(n-1)$;
- The second branch, used with probability $(1-q_0)$, takes $\mathcal{O}((n-1)^2)$ since it is necessary to compute the sum of $n-1$ parcels in formula (5.21), for $n-1$ alternative edges.

Analogously, if k nodes were already inserted, from each of those nodes there are $n-k$ possible choices, which applied to the first and second branch of (5.20) would take $\mathcal{O}((n-k)k)$ and $\mathcal{O}((n-k)^2 k)$, respectively. Therefore, the construction of the tree presumes

$$\mathcal{O}\left(mn^2 + \sum_{k=1}^{n-1}\left[q_0(n-k)k + (1-q_0)(n-k)^2 k\right]\right) \tag{5.28}$$

or simply

$$\mathcal{O}\left(mn^2 + q_0 n^3 + (1-q_0)n^4\right). \tag{5.29}$$

□

A similar analysis can be made for the Travelling Salesman Person case as follows.

Theorem 5.2. *Consider the implementation, in Section 5.3.3, for the Multiple Objective Travelling Salesman Person problem over a complete network instance. The construction of a solution has*

$$\mathcal{O}(mn^2 + q_0 n^2 + (1 - q_0)n^3) \tag{5.30}$$

complexity.

Proof. The values of the products present in (5.20) and (5.21) are calculated as a pre-computation, which takes $\mathcal{O}(mn^2)$. The main construction process starts by picking one node. From that node exists $n-1$ possible edges which implies a $\mathcal{O}(q_0(n-1)+(1-q_0)(n-1)^2)$. If it were already inserted $k-1$ edges (k nodes in $e_{n_1n_2}, e_{n_2n_3}, \ldots, e_{n_{k-1}n_k}$), then the construction process searches from the $n-k$ nodes adjacent to node n_k, which for the first branch of (5.23) takes $\mathcal{O}(n-k)$ and for the second one takes $\mathcal{O}((n-k)^2)$. Therefore the construction of a solution is made in

$$\mathcal{O}\left(mn^2 + \sum_{k=1}^{n-1}\left[q_0(n-k) + (1-q_0)(n-k)^2)\right]\right) \tag{5.31}$$

which can be simplified to

$$\mathcal{O}(mn^2 + q_0 n^2 + (1 - q_0)n^3). \tag{5.32}$$

□

In none of the analysis is considered the complexity of the depth search phase, whenever it exists. However some things can be made to diminish the computational time, like

- Use values of q_0 near to 1, which has the risk of not providing enough exploration of the search space;
- In the k-Degree Minimum Spanning Trees case select a subset of the nodes in the subtree instead of the all subtree, diminishing the number of times that (5.21) is computed;

- Use a candidate list (as in [Dorigo & Stutzle, 2004]), which through a pre-computation allows to diminish the cardinality of $\mathcal{A}$ in formulas (5.21) and (5.23), by restricting the number of choices to be considered in each construction step.

Nevertheless, the computational requirements to compute a single solution are high and if that solution has a good fitness then it should not be simply evaluated and discarded. In fact, the use of a local search, like the 2-opt, 3-opt, or the depth search process proposed with ϵ-DANTE, allows to further explore the solutions, possible achieving new ones to improve the approximation set with, less computational effort.

5.4 Summary

This chapter has been devoted to the analysis of two methods proposed for the resolution of discrete multiple objective optimization problems: MONACO and ϵ-DANTE. Both methods, based in the Ant Colony Optimization paradigm, use as many layers of pheromones as the number of weights of the multiple objective problems.

In particular, MONACO was applied to three problems, namely: a multiple objective flows simulator, an optimizer for the Multiple Objective Travelling Salesman Person, and the Multiple objective Minimum Spanning Trees Problems. MONACO shows potential to be a working tool for optimization and simulation problems. However, it suffers from the common to most of the Ant Colony Optimization algorithms pitfall, that is, the refinement of the solutions can require an extra computational effort, possibly diminished with the use of post-optimizers. Moreover, after the evaluation of a solution two things can happen:

- The solution enters the approximation set and information is propagated through the pheromone vectors; or

- The solution is discarded which happens even if the solution is "attractive", that

is, it is near to the approximations set or has the same fitness of one of its elements. In this case it can be lost all the computational effort necessary to build the solution.

The ϵ-DANTE algorithm appears as an effort to provide an effective way of further exploring the solutions that improve the approximation set or the solutions that are considered "attractive" (based in their fitness). More precisely, whenever a solution is inserted into the approximation set or satisfies an ϵ distance to the approximation set criterion, it is performed a limited depth search using the pheromone values to guide the search.

A version of ϵ-DANTE was implemented and applied to two multiple objective problems for which exits a set of publish results: the k-Degree Minimum Spanning Trees and the Travelling Salesman Person. In both cases, it was verified that the method rapidly converges toward the fronts achieved by other authors and ends up by improving, in general, their results.

In this chapter has also been proposed a pheromone update strategy that uses the elements of the approximation set. In the Ant Colony algorithms is common to use greedy strategies, where only the best performing solutions are used to update the pheromone value. One possible procedure uses all the elements of the approximation set to update the pheromone trails. However, the cardinality of this set is usually very large, which introduces noisy pheromone trails, reducing the performance of the algorithm. The proposed update strategy uses only small subsets of the approximation set, which successively favours the exploration of small part of the search space.

Next chapter will be devoted to the analysis of the results obtained with ϵ-DANTE when applied to the Multiple Objective Minimum Spanning Trees problem for a large set of instances obtained through the application of the network generators proposed in Chapter 4.

Insanity: doing the same thing over and over again and expecting different results.

Albert Einstein

6

Computational Results

Contents

6.1 Overview

This chapter makes an analysis of the results obtained with an ϵ-DANTE implementation when applied to several instances of the Multiple Objective Minimum Spanning Trees problem. Previously, ϵ-DANTE algorithm was described and compared with some high performing methods. For the tested instances, the obtained results showed that this method produces quality solutions, which were, at least, comparable to the other implementations.

Concerning the problem that is in study in this chapter, the construction of the spanning trees starts by randomly picking one node from $\mathcal{V}$. Then, while the solution is not complete, the edges are added to the tree in construction using formula

$$e_{st} = \begin{cases} \arg\max_{e_{s't'} \in \mathcal{A}} \left(\prod_{j=1}^{m} \tau_j(e_{s't'})^{\alpha_j} w_j(e_{s't'})^{-\beta_j} \right) & \text{if } q \leq q_0 \\ e & \text{if } q > q_0 \end{cases}, \tag{6.1}$$

where $\mathcal{A} = \{e_{uv} \in \mathcal{E} : u \in T_N \wedge v \in \mathcal{V} - T_N\}$ and the remaining values are explained in section 5.3.3.

To test the ϵ-DANTE implementation a total of 404 networks with 20, 50 and 100 nodes were considered. Those networks were obtained with the generators described in Chapter 4. Further details related to the dimensions and classes of the networks that were used can be found in [Cardoso *et al.*, 2006b].

For each network instance the method was run once and its results are compared with results produced by an Brute Force and/or a Weighted Sum method, according to the dimensions of the problem.

The computational environment used to run ϵ-DANTE was a set of PCs with Intel Pentium$^{\text{TM}}$IV processors at $3Ghz$, $512Mb$ of RAM and Windows XP OS. The used parameters are resumed in Table 6.1. On the other hand, as in Section 5.2.2, the results for the Brute Force method were obtained through an implementation of the algorithm proposed by Ramos *et al.* [1998], in a LINUX OS workstation with Intel Pentium$^{\text{TM}}$III $533Mhz$ processor and $128Mb$ of RAM. For the Weighted Sum approach the results were obtained using 1000 distinct pondering weights (equation (5.9)). The solutions obtained with these implementations were compared using a set of reference metrics, described in Section 2.4, like the Error Ratio, the Set Coverage, and the $R1$, $R2$ and $R3$ metrics.

The approximation and the Pareto sets, whenever they exist, obtained with the implementations are sketched in Appendix B.

Parameter	Values	Comments
D	1	Maximum depth
d	$\lfloor \frac{\lvert\mathcal{V}\rvert}{2} \rfloor$	Intermediate depth
M	2	Number of branches
ϵ	0	
α_i, β_i	$\{0, 0.03, 0.06, \ldots, 3.0\}$	
ρ	0.1	
q_0	$\{0.5, 0.6, 0.7, 0.8, 0.9\}$	Set randomly for each ant
Number if ants per cycle	$\lvert\mathcal{V}\rvert$	
Number of cycles	2	
Maximum run time	$\min\{60\lvert\mathcal{V}\rvert \log \lvert\mathcal{V}\rvert, 36000\}$	Seconds

Table 6.1: Used parameters for ϵ-DANTE in the Multiple Objective Minimum Spanning Trees construction.

Therefore, the results for the complete set of tests are examined in the remaining chapter. The analysis is divided according to the dimensions of the networks and the classes of their generators.

6.2 Performance Analysis

6.2.1 20 Nodes Networks

The ϵ-DANTE implementation was applied to 143 networks with 20 nodes. For 79 of those networks, networks that define less than 10^{11} distinct spanning trees, it was applied a Brute Force method to compute the exact Pareto set. Table 6.2 resumes the mean and standard deviation values of the metrics results obtained with ϵ-DANTE for

	Network with 20 nodes and less that 10^{11} distinct spanning trees	
	Mean (μ)	Standard Deviation (σ)
$\lvert\mathcal{P}_{bf}\rvert$	54,9	53,4
$\lvert\mathcal{P}_{\epsilon}\rvert$	54,9	53,4
$ER(\mathcal{P}_{bf}, \mathcal{P}_{\epsilon})$	0,0	0,0
$C(\mathcal{P}_{bf}, \mathcal{P}_{\epsilon})$	1,0	0,0
$C(\mathcal{P}_{\epsilon}, \mathcal{P}_{bf})$	1,0	0,0
$R1(\mathcal{P}_{bf}, \mathcal{P}_{\epsilon})$	0,5	0,0
$R2(\mathcal{P}_{bf}, \mathcal{P}_{\epsilon})$	0,0	0,0
$R3(\mathcal{P}_{bf}, \mathcal{P}_{\epsilon})$	0,0	0,0
Time (sec)	18,8	64,7

Table 6.2: Comparison of the Brute Force method with ϵ-DANTE.

those 79 networks, where $\mathcal{P}_{hf}$ and $\mathcal{P}_{\epsilon}$ are the sets of solutions returned by the Brute Force and the ϵ-DANTE methods, respectively. From the analysis of the table values, it is possible to conclude that ϵ-DANTE was able to compute the exact Pareto set in all the cases. Figure 6.1 sketches the box-and-whisker plot of the distribution of the last time that the Pareto set was updated by the process. In this case, for 25% of the problems the optimum set is achieved in less than 2 seconds and for 75% of them after 17 seconds.

Table 6.3 presents a summary of the values obtained with ϵ-DANTE for the 143 networks with 20 nodes (see also Figure 6.2), where $\mathcal{P}_{ws}$ and $\mathcal{P}_{\epsilon}$ are the sets of solutions returned by the Weighted Sum and, as before, the ϵ-DANTE , respectively.

From the analysis of the values some conclusions can be made:

- In average, 28.14% of the elements of $\mathcal{P}_{\epsilon}$ are weakly dominated by at least one of

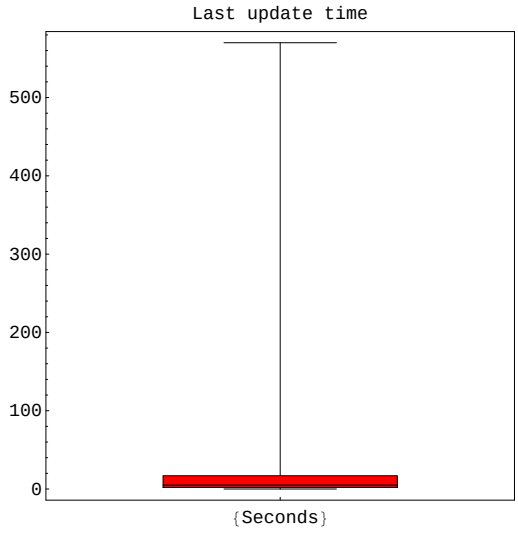


Figure 6.1: Box-and-whisker for the time of the last update performed by ϵ-DANTE in the 79 networks for which Brute Force Method was run.

	20 nodes networks	
	Mean (μ)	Standard Deviation (σ)
$\lvert\mathcal{P}_{ws}\rvert$	14,6	10,0
$\lvert\mathcal{P}_{\epsilon}\rvert$	129,9	246,2
$C(\mathcal{P}_{ws}, \mathcal{P}_{\epsilon})$	0,2814	0,2196
$C(\mathcal{P}_{\epsilon}, \mathcal{P}_{ws})$	0,9884	0,0978
$R1(\mathcal{P}_{ws}, \mathcal{P}_{\epsilon})$	0,1932	0,1369
$R2(\mathcal{P}_{ws}, \mathcal{P}_{\epsilon})$	5,8672	111,9400
$R3(\mathcal{P}_{ws}, \mathcal{P}_{\epsilon})$	0,0356	0,1320
Time (sec)	277,9	819,0

Table 6.3: Mean and standard deviation for the metrics obtained for 20 nodes networks.

the elements of $\mathcal{P}_{ws}$, which means that the remaining 71.86% are not dominated by any of the elements of $\mathcal{P}_{ws}$.

- Also in average, 98.84% of the solutions obtained with the Weighted Sum method

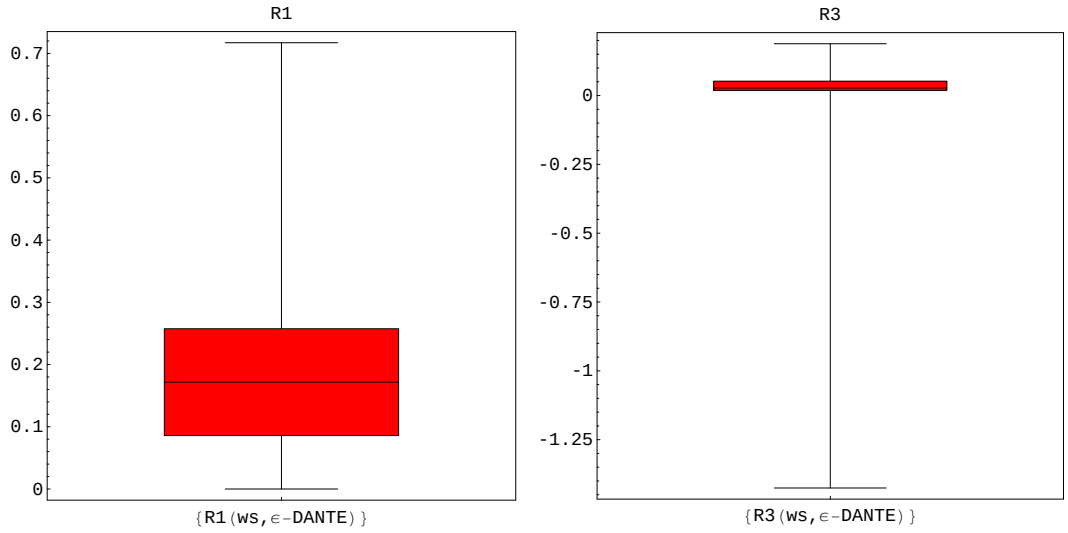


Figure 6.2: Box-and-whisker plots for $R1$ and $R3$ metrics comparing the Weighted Sum (ws) with ϵ-DANTE for the 143 networks with 20 nodes.

are weakly dominated by the solutions that were retuned by ϵ-DANTE and only 1.16% of the elements of $\mathcal{P}_{ws}$ are not weakly dominated by the elements of $\mathcal{P}_{\epsilon}$.

- $\mu_{R1(\mathcal{P}_{ws},\mathcal{P}_{\epsilon})} = 0.1932$ and $\mu_{R3(\mathcal{P}_{ws},\mathcal{P}_{\epsilon})} = 0.0356$ evidence that the results obtained with ϵ-DANTE outperform the results achieved by the Weighted Sum implementation.

6.2.2 50 Nodes Networks

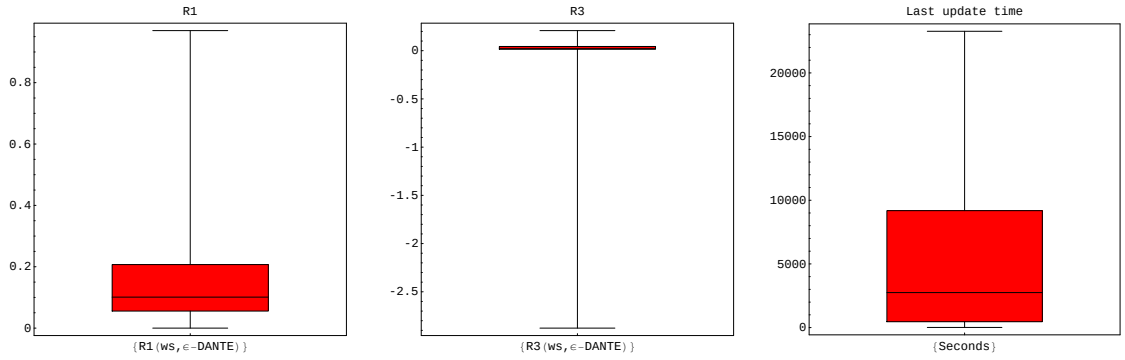
The implementation of ϵ-DANTE was applied to 119 networks with 50 nodes. Table 6.4 resumes the statistical results for the same metrics of the previous section, and the box-and-whisker diagrams for the observed values of $R1$, $R3$ and time of the last update of the approximation set are sketched in Figure 6.3.

From the analysis of the table values, some observations can be made:

- 20.09% is the percentage of elements of $\mathcal{P}_{\epsilon}$ that are weakly dominated by the elements of $\mathcal{P}_{ws}$. This implies that 79.91% of the elements of $\mathcal{P}_{\epsilon}$ are not weakly

	50 nodes networks	
	Mean (μ)	Standard Deviation (σ)
$\lvert\mathcal{P}_{ws}\rvert$	33,7	24,3
$\lvert\mathcal{P}_{\epsilon}\rvert$	344,7	387,1
$C(\mathcal{P}_{ws}, \mathcal{P}_{\epsilon})$	0,2009	0,1731
$C(\mathcal{P}_{\epsilon}, \mathcal{P}_{ws})$	0,8219	0,3381
$R1(\mathcal{P}_{ws}, \mathcal{P}_{\epsilon})$	0,1670	0,1812
$R2(\mathcal{P}_{ws}, \mathcal{P}_{\epsilon})$	2,5530	112,0223
$R3(\mathcal{P}_{ws}, \mathcal{P}_{\epsilon})$	0,0432	0,0657
Time (sec)	4940,1	5168,0

Table 6.4: Mean and standard deviation for the metrics obtained for 50 nodes networks.



(a) (b) (c)

Figure 6.3: Box-and-whisker plots for (a)-(b) $R1$ and $R3$ metrics comparing the Weighted Sum (ws) with ϵ-DANTE, and (c) time of the last update of the approximation set, for the 119 networks with 50 nodes.

dominated by any of the elements of $\mathcal{P}_{ws}$.

- 82.19% of the elements of $\mathcal{P}_{ws}$ are weakly dominated by the elements of $\mathcal{P}_{\epsilon}$.
- $\mu_{R1(\mathcal{P}_{ws},\mathcal{P}_{\epsilon})} = 0.1670$ and $\mu_{R3(\mathcal{P}_{ws},\mathcal{P}_{\epsilon})} = 0.0432$ also evidences that the results returned by ϵ-DANTE improve the results that were returned by the Weighted

Sum implementation.

- The cardinal of $\mathcal{P}_{ws}$ is approximately one tenth of the cardinal of $\mathcal{P}_{\epsilon}$.

6.2.3 100 Nodes Networks

For the 100 nodes cases it were considered 142 networks. Table 6.5 resumes the average and standard deviation of the metrics values that were obtained, and Figure 6.4 depicts the box-and-whisker plot for the $R1$ and $R3$ metrics distribution. In this case,

- $\mu_{R1(\mathcal{P}_{ws},\mathcal{P}_{\epsilon})} = 0.2401$, which indicates that often $\mathcal{P}_{\epsilon}$ has a better utility function value than $\mathcal{P}_{ws}$.

- $\mu_{R3(\mathcal{P}_{ws},\mathcal{P}_{\epsilon})} = -0,0798 < 0$ evidence that, in average, the utility function values of $\mathcal{P}_{ws}$ were better than the values of $\mathcal{P}_{\epsilon}$. However, observing the box-and-whisker plot for $R3$ (Figure 6.4) for which the quartile values are $\{-0.0123, 0.0091, 0.0249\}$, it is possible to conclude that more than 50% of the networks have $\mu_{R3(\mathcal{P}_{ws},\mathcal{P}_{\epsilon})} > 0$ (in fact, 97 of the 142, which correspond to 68.3%).

- 63.73% of the elements of $\mathcal{P}_{ws}$ are weakly dominated by the elements of $\mathcal{P}_{\epsilon}$.

- 75.35% of the elements of $\mathcal{P}_{\epsilon}$ are not weakly dominated by any element of $\mathcal{P}_{ws}$.

6.2.4 Global Results

Table 6.6 presents a statistical resume of the metric values for the 404 network instances and the box-and-whisker plots for the values of the $R1$ and $R3$ metrics are sketched in Figure 6.5. In general, it is possible to conclude that

- ϵ-`DANTE` returned approximately ten times the number of supported solutions computed by the Weighted Sum method, $\mu_{|\mathcal{P}_{\epsilon}|} \approx 10\mu_{|\mathcal{P}_{ws}|}$;

	100 nodes networks	
	Mean (μ)	Standard Deviation (σ)
$\|ws\|$	64,5	51,2
$\|\mathcal{P}_\epsilon\|$	674,7	539,7
$C(\mathcal{P}_{ws}, \mathcal{P}_\epsilon)$	0,2465	0,2518
$C(\mathcal{P}_\epsilon, \mathcal{P}_{ws})$	0,6373	0,3339
$R1(\mathcal{P}_{ws}, \mathcal{P}_\epsilon)$	0,2401	0,2653
$R2(\mathcal{P}_{ws}, \mathcal{P}_\epsilon)$	-347,3236	1704,5600
$R3(\mathcal{P}_{ws}, \mathcal{P}_\epsilon)$	-0,0798	0,4492
Time (sec)	20650,8	11651,6

Table 6.5: Mean and standard deviation for the metrics obtained for 100 nodes networks.

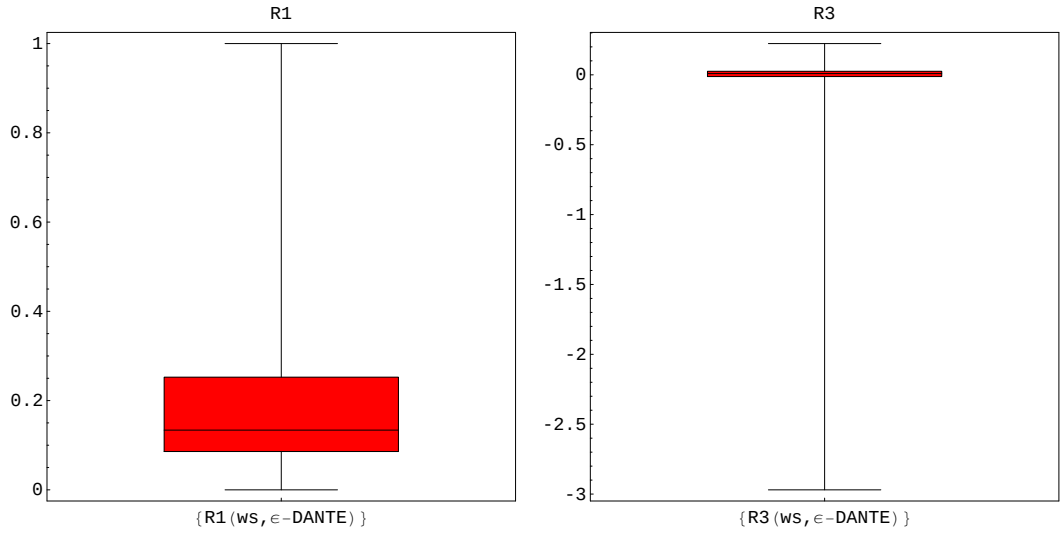


Figure 6.4: Box-and-whisker plots for $R1$ and $R3$ metrics comparing the Weighted Sum (ws) with ϵ-DANTE for the 142 networks with 100 nodes.

- In average 81.6% of the solutions returned by the Weighted Sum method are weakly dominated by the solutions returned by ϵ-DANTE. This value should be

	20, 50 and 100 nodes networks	
	Mean (μ)	Standard Deviation (σ)
$\|ws\|$	37,8	39,7
$\|\mathcal{P}_\epsilon\|$	384,6	469,3
$C(\mathcal{P}_{ws}, \mathcal{P}_\epsilon)$	0,2454	0,2213
$C(\mathcal{P}_\epsilon, \mathcal{P}_{ws})$	0,8160	0,3125
$R1(\mathcal{P}_{ws}, \mathcal{P}_\epsilon)$	0,2020	0,2043
$R2(\mathcal{P}_{ws}, \mathcal{P}_\epsilon)$	-119,2504	1026,1224
$R3(\mathcal{P}_{ws}, \mathcal{P}_\epsilon)$	-0,0027	0,2850
Time (sec)	8812,0	11628,1

Table 6.6: Mean and standard deviation for the metrics values obtained for the 404 networks.

considered together with the fact that the solutions in the $\mathcal{P}_{ws}$ sets are Pareto solutions. On the other hand, in average only 24.54% of the solutions obtained with ϵ-DANTE are weakly dominated by the solutions produced by the Weighted Sum method, which include the ones that are dominated and the ones that have equal objective values;

- From the statistical values $\mu_{R1(\mathcal{P}_{ws},\mathcal{P}_\epsilon)} = 0,2020$ and $\mu_{R3(\mathcal{P}_{ws},\mathcal{P}_\epsilon)} = -0,0027$ and Figure 6.5 it is inferable that ϵ-DANTE , in the majority of the problems, returned fronts that are better than $\mathcal{P}_{ws}$ set.

In order to make a more detailed analysis based in the classes of generators, Table 6.7 (with values depicted in Figures 6.6) presents the mean and standard deviation of $R1$ and $R3$ per classes of network weights generators.

In general, ϵ-DANTE returned approximation sets such that $\mu_{R1(\mathcal{P}_{ws},\mathcal{P}_\epsilon)}$ is inferior to 0.2646, with minimum value among the classes equal to 0.0541 for the CWG networks.

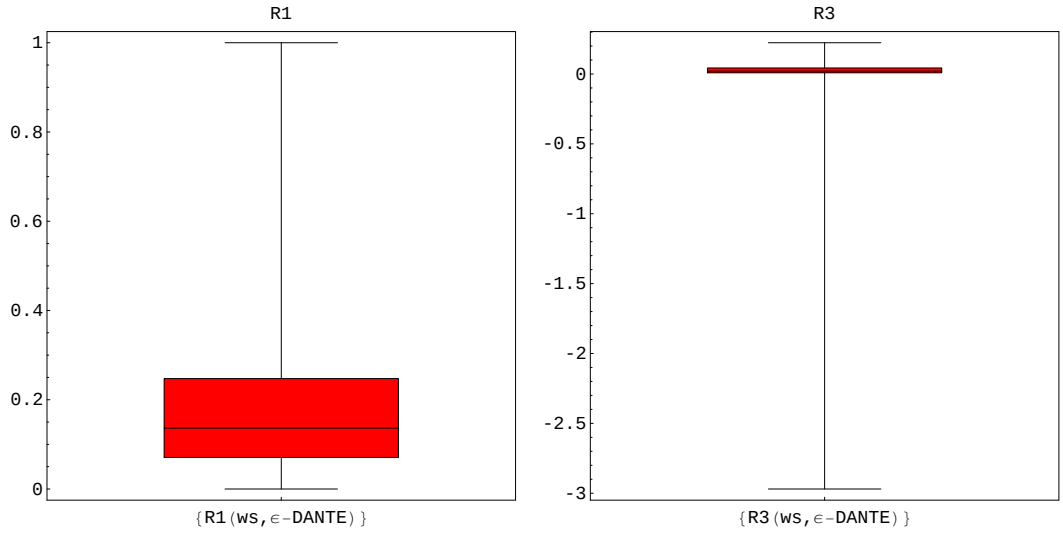


Figure 6.5: Box-and-whisker plots for $R1$ and $R3$ metrics comparing the Weighted Sum and ϵ-DANTE for the 404 networks with 20, 50 and 100 nodes.

	$R1(\mathcal{P}_{ws}, \mathcal{P}_{\epsilon})$		$R3(\mathcal{P}_{ws}, \mathcal{P}_{\epsilon})$		
	Mean (μ)	Standard Deviation (σ)	Mean (μ)	Standard Deviation (σ)	Number of networks in each class
RWG	0,2051	0,2042	0,2042	0,0224	53
CWG	0,0541	0,0617	0,1207	0,0721	81
$\rho - CWG$ with $\rho < 0$	0,2315	0,2625	-0,0976	0,4722	134
$\rho - CWG$ with $\rho > 0$	0,2646	0,1461	0,0095	0,0336	136

Table 6.7: Mean and standard deviation values for the $R1$ and $R3$ metrics for networks divided according to their generators class.

Relatively to the $\mu_{R3(\mathcal{P}_{ws},\mathcal{P}_{\epsilon})}$ the worse case is observed for the class $\rho - CWG$ with $\rho < 0$, although it has a $\mu_{R1(\mathcal{P}_{ws},\mathcal{P}_{\epsilon})}$ equal to 0.2315. A more detailed observation of the

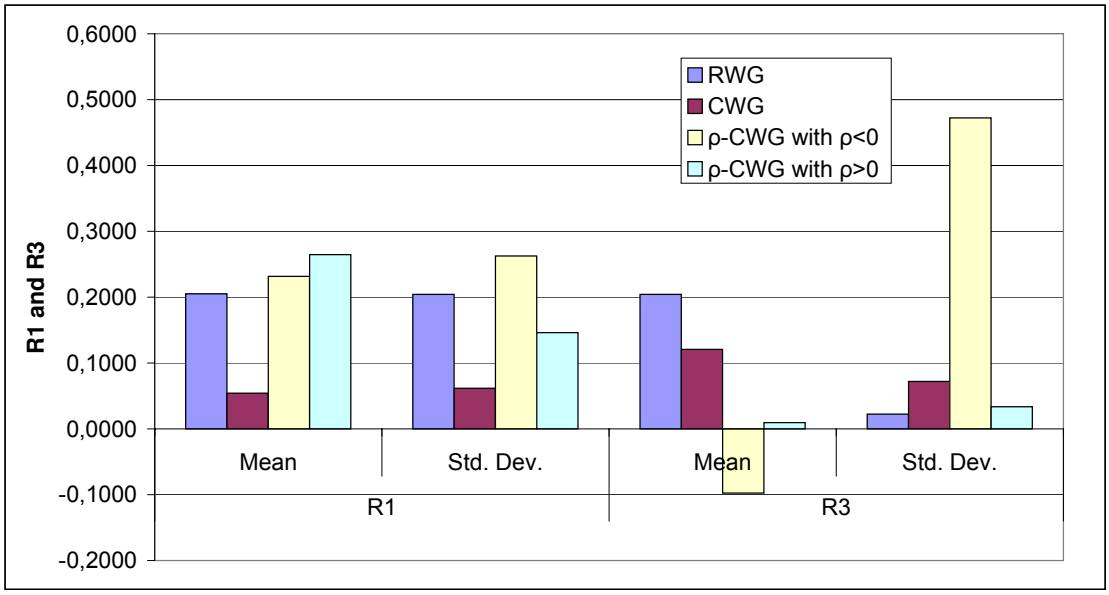


Figure 6.6: Mean and standard deviation values for the $R1$ and $R3$ metrics for networks divided according to their generators class.

distributions of the $R1$ and $R3$ values can be made using the box-and-whisker plots in Figures 6.7, and 6.8.

Figure 6.9 simultaneously depicts the pairs of values ("Time of the last update of the approximation set", $R1$) and ("Time of the last update of the approximation set", $R3$) for all the 404 tests. It is possible to observe that the worse cases, that is, with smaller $R3$ value, occur when the last update is made earlier. This seams to indicate a small exploration of the search space possibly motivated by a premature local convergence, which can be justified by the greedy Angle-Pheromone Update strategy.

6.3 Summary

An analysis of the results obtained with ϵ-`DANTE` for the Multiple Objective Minimum Spanning Trees problem has been made in this chapter. The results were compared with the exact Pareto set for the smaller networks. In the larger ones, it was used a

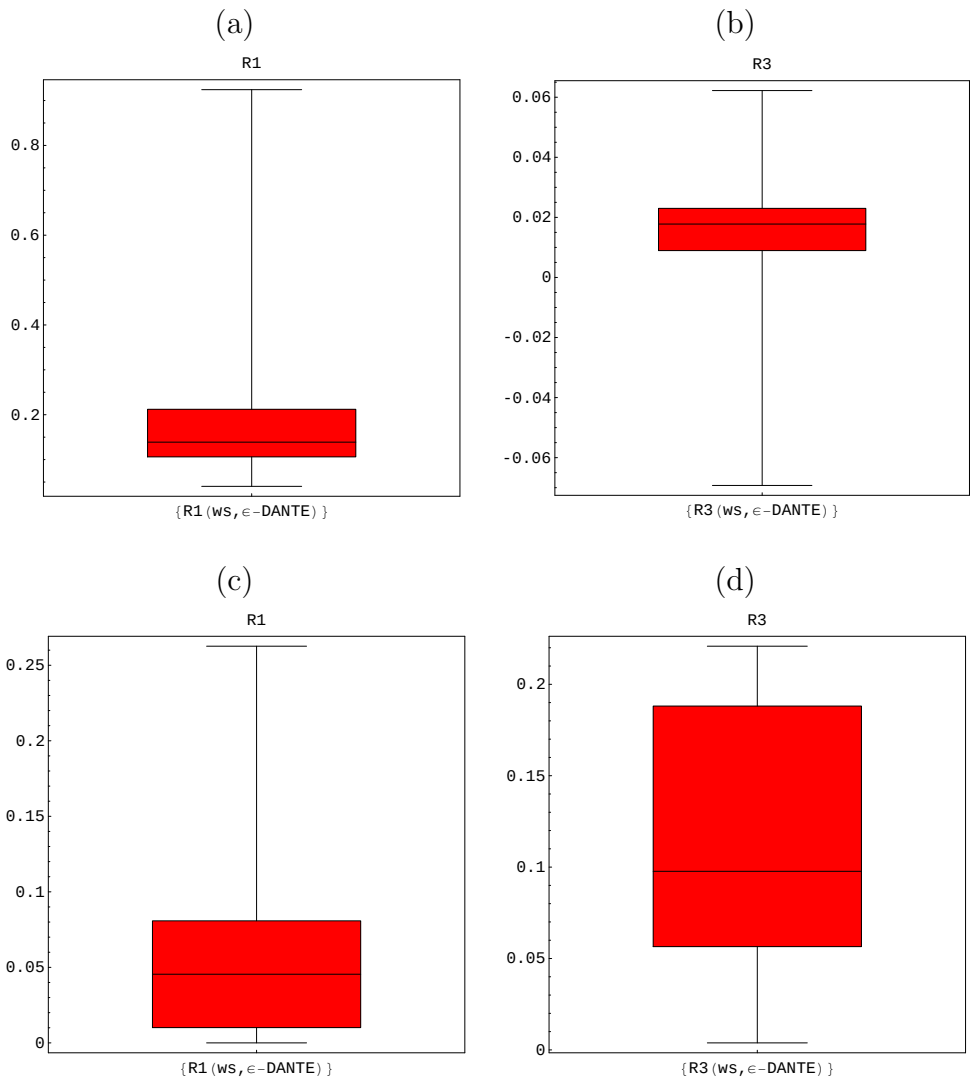


Figure 6.7: Box-and-whisker plots of the $R1$ and $R3$ metrics for the networks generated with: (a)-(b) the RWG; and (c)-(d) the CWG.

subset of the Pareto set obtained with the Weights Sum method. The main idea was not to compare methods, like ϵ-DANTE with the Weighted Sum, since they were not tested in the same condition. In fact, the study made in this chapter is motivated by other objectives, like

- The analysis of the reliability, accuracy and versatility of the ϵ-DANTE method; and

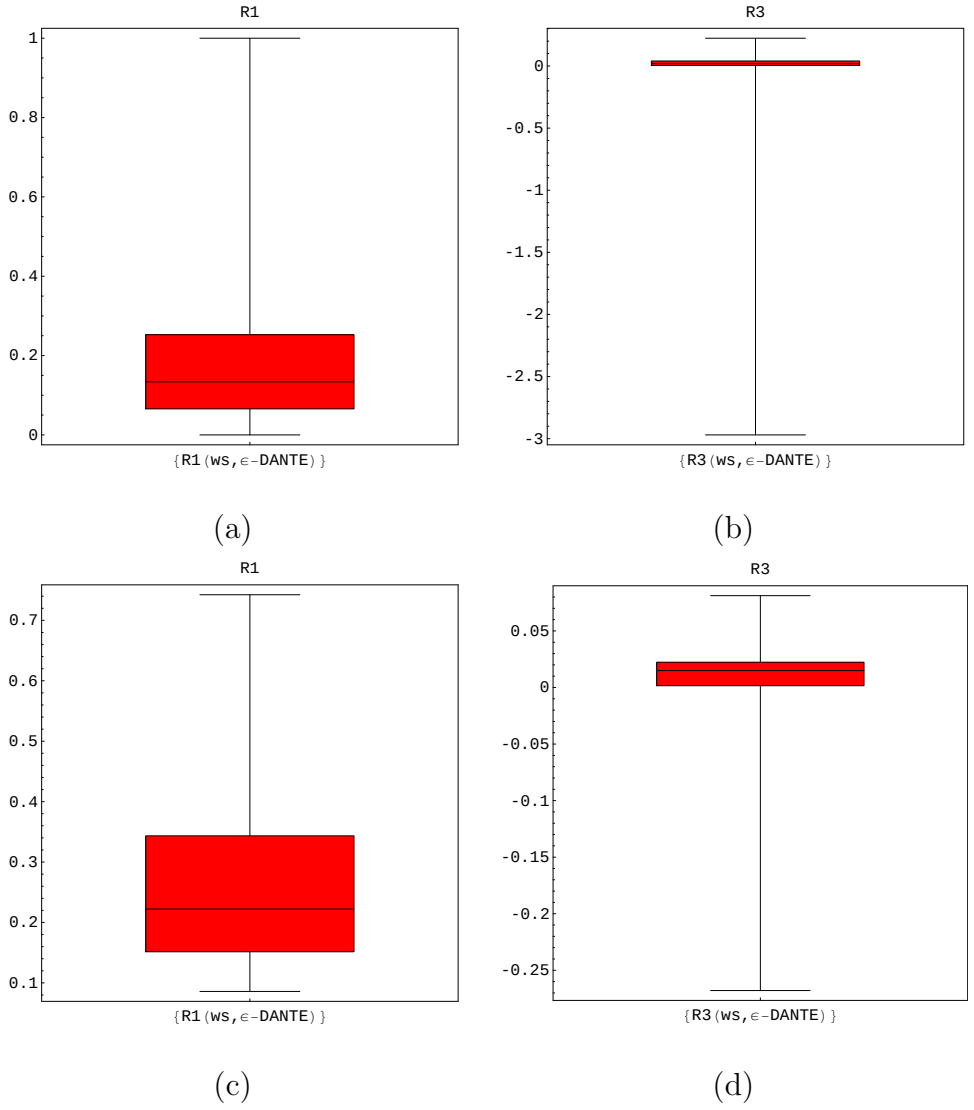


Figure 6.8: Box-and-whisker plots of the $R1$ and $R3$ metrics for networks generated with $\rho - CWG$: (a)-(b) with $\rho < 0$ and (c)-(d) with $\rho > 0$.

- To prove that ϵ-DANTE can be used as a high performance tool in optimization problems, namely in problems for which their intrinsic complexity is recognized.

In general, it was possible to conclude that ϵ-DANTE procedure produces high quality solutions and seems to be an alternative method to the previously presented cases.

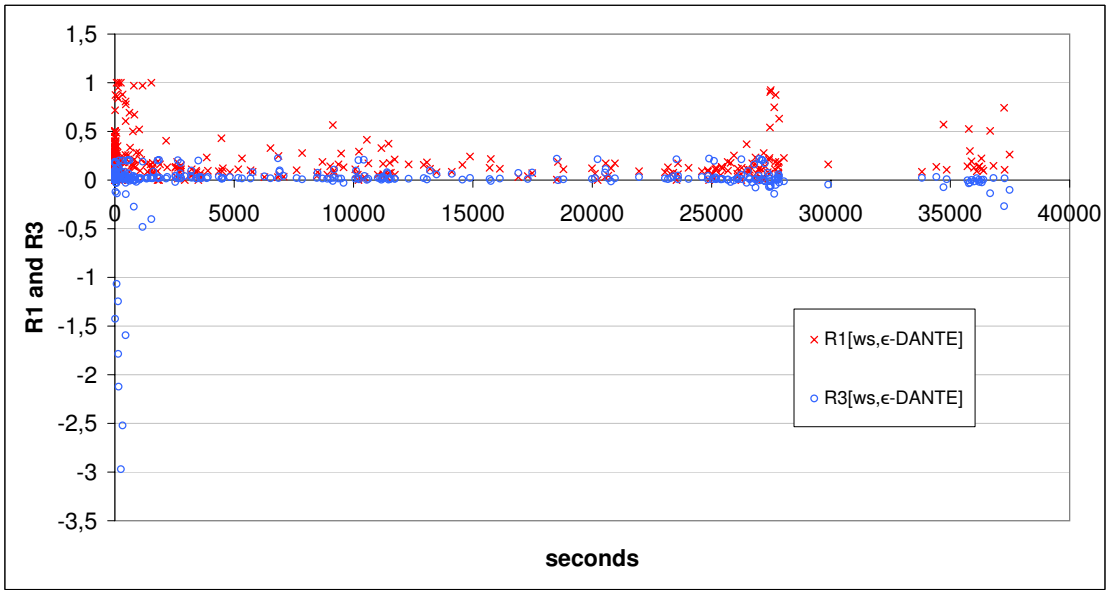


Figure 6.9: Distributions of the pairs ("Time of the last update of the approximation set",$R1(\mathcal{P}_{ws}, \mathcal{P}_\epsilon)$) and ("Time of the last update of the approximation set",$R3(\mathcal{P}_{ws}, \mathcal{P}_\epsilon)$) for the 404 networks.

7

Conclusions and Future Work

Contents

The analysis and presentation of practical solutions for the Multiple Objective Minimum Spanning Trees problem was one of the main aims of this thesis. Like in the generality of the multiple objective problems, the computational complexity of this problem is very high, which implies that almost always is impossible to exactly determine the Pareto set. Therefore, alternatives should be thought that can produce pragmatic answers, that is, good approximations to the Pareto set in reasonable time. The next sections will present general conclusions about the developed work and possible future research.

7.1 Multiple Objective Minimum Spanning Trees Problem

This thesis devoted the entire third chapter, along with some other sections, to the study of the Multiple Objective Minimum Spanning Trees problem. Related to this problem, earlier proved to be $\mathcal{NP}$-complete, were presented a set of theoretical results, which, as discussed, the *per se* can hardly be efficiently applied to the construction of approximation algorithms. Nevertheless, it was possible to observe that, in some conditions, the search space could be restricted either through the guaranty of the inclusion of certain edges in all the efficient spanning trees or through the exclusion of edges that could not belong to any efficient solution. However, the process necessary to establish these restrictions has a very high computational overhead, which limits its application.

In the same third chapter it is made a survey of the previous methods that were used by other authors to compute the approximation sets. In their majority, these methods rely in the use of Weighted Sum strategies, which take advantage of the existence of efficient algorithms to solve the single objective problem that is associated. From those algorithms, the best performing are two phase methods that after the computation of the supported solutions, using the Weighted Sum method, continue the process applying other optimization strategies, which allows looking for other efficient/approximation solutions. Nevertheless, it were also sketched a set of limitations for those algorithms, which exhort for the development of alternative strategies.

7.2 Repository of Network Problems

The test of alternative methods has to deal with the absence of libraries for the generality of the Multiple Objective Minimum Spanning Tree problems. Before this insufficiency, in Chapter 4 it was presented a set of network generators. Mainly directed for

practical network types, the generation of a network is divided in three sub-generating steps: nodes, edges and (edge) weights. The combination of the sub-generators allow to define several network topologies and associated Pareto fronts, which induce different problems in the computation of the approximation sets. With those generators it was built an extensive repository of network instances, which were used latter to test the proposed algorithms.

7.3 MONACO and ϵ-DANTE

Chapter 5 proposes two Swarm Intelligence algorithms, which are particularly adapted to the optimization of multiple objective problems. The first part of the chapter describes the MONACO algorithm. This method is based in m layers of pheromones, each one related to a weight, implemented in the form of m-ary vectors associated to the edges. The components of those vectors are used to stochastically guide the search, allowing to explore distinct zones of the search space. This exploration is mainly done by adjusting the values of the algorithm parameters, which permits to alternatively explore the search/objective space in different directions. MONACO's algorithm was applied in the optimization of three multiple objective networks problems. First it was used in the simulation of a network of flows, where the pheromone trails were used to guide the flows through the edges. Latter it was adapted to the Multiple Objective Travelling Salesman Person and the results were compared with the results published by other authors. Returning to the main problem, the Multiple Objective Minimum Spanning Trees, it was made another implementation of MONACO, allowing to verify that the method could be considered as an alternative to the processes that were previously used. For this problem, an extensive set of tests was conducted over the networks repository previously referred. Nevertheless, it was verified that MONACO's process, as most of the constructive algorithm, does not exploit the computational effort that was

made to build the solutions.

In the second part of the chapter it is proposed another new method called ϵ-DANTE. This method is an evolution of the MONACO's procedure, which does a more effective exploitation of the computational effort that was made to build the solutions. This is achieved through the application of a pheromone oriented depth search procedure based in the more promising solutions. That depth search is limited both in the depth and the number of branches of the search tree.

The ϵ-DANTE method was first applied to two multiple objective problems for which are known approximation sets: the Travelling Salesman Person and the k-Degree Minimum Spanning Trees problem. In the generality of the cases, the known approximation sets, which were obtained with other evolutionary methods, were improved.

Furthermore, the results of the application of ϵ-DANTE to the Multiple Objective Minimum Spanning Trees problem were presented in Chapter 6. A statistical analysis over more than 400 networks with different topologies and weights allowed to conclude that, almost in 100% of the cases the ϵ-DANTE method improves the results obtained with the Weighted Sum process. In the cases where it was possible to run a Brute Force method, ϵ-DANTE was capable to compute the exact Pareto set.

7.4 Future Research

From the global analysis of the developed work, a set of future researches can be derived, like

- **Further develop the strategies proposed in MONACO and ϵ-DANTE.** Although the promising results obtained with both methods, their effective application is significantly limited by the size of the problems. Therefore, the inclusion in the search process of other optimization strategies and heuristics should be further

thought as a way to solve larger instances, to improve the quality of the solutions and to accelerate the search procedure.

- **Improve the theoretical results and apply them to the meta-heuristic search procedures in a more effective manner.** Other strategies to improve the computational results can be achieved recurring to pre/post-computation processes. This allows to restrict the search space when applied in the pre-computing stage, or to further exploit the computed solutions when applied in the post-computing case.

- **Extend the repository of networks.** One of the main concerns in the establishment of the network generators in Chapter 4 was to provide a large set of similar to real world problems, mainly directed for the spanning trees case. Therefore, the proposed repository should contain some other features, like undirected networks, networks with a larger number of objectives and nodes, different topologies and different instances of the spanning trees problem.

Additional future work include the

- **Apply the proposed optimization methods to other practical problems.** For example, most of the real networks are dynamic systems, like the traffic, the telecommunication or the emergency networks. The referred dynamism can be associated to all the components of the networks (nodes, edges or weights), introducing additional difficulties to the traditionally static problems. However, as already pointed out, the preliminary tests indicate that the adaptive methods that were proposed are particularly suitable to this types of problems, with some pitfalls related with the computational time.

- **Improvement of the pheromone updating strategies.** The pheromone updating step is fundamental in the search procedure. Inappropriate strategies,

like strategies that introduce noisy pheromone trail, significantly reduce the algorithms performance. Therefore, although the Angle-Pheromone Update tries to avoid this and other limitations, further improvements or possible alternatives should be thought.

- **Implementation of distributed and/or parallel systems** The application to practical problems require modelling system with large dimensions. The optimization of those systems demand the use of huge computational capacities that are not usually available in single processor machines. Therefore, in the future the proposed method should be adapted, so that they can be applied in distributed/parallel computational systems.

References

802.1D, 2004 (2004). *802.1D - IEEE Standard for Local and metropolitan area networks*. IEEE Computer Society.

Aarts, E. & Lenstra, J. (1997). *Local Search in Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Inc.

Abellanas, M., García, J., Hernández-Peñalver, G., Hurtado, F., Serra, O. & Urrutia, J. (1996). Onion polygonizations. *Inf. Process. Lett.*, **57**, 165–173.

Abuali, F., Schoenefeld, D. & Wainwright, R. (1994). Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees. In *SAC '94: Proceedings of the 1994 ACM symposium on Applied computing*, 242–246, ACM Press, New York, NY, USA.

Adami, C. & Mazure, A. (1999). The use of minimal spanning tree to characterize the 2D cluster galaxy distribution. *Astronomy & Astrophysics Supplement Series*, **134**, 393–400.

AHD, 2000 (2000). The american heritage dictionary of the english language.

Alam, S., Abbass, H. & Barlow, M. (2006). Multi-objective ant colony optimization for weather avoidance in a free flight environment. Tech. rep., The Artificial Life and Adaptive Robotics Laboratory, ALAR Technical Report Series.

Alves, M. & Clímaco, J. (2000). An interactive method for 0-1 multiobjective problems using simulated annealing and tabu search. *Journal of Heuristics*.

An, L., Xiang, Q.S. & Chavez, S. (2000). A fast implementation of the minimum spanning tree method for phase unwrapping. *IEEE, Transactions on Medical Imaging*, **19**.

Andersson, J. (2001). *Multiobjective Optimization in Engineering Design*. Ph.D. thesis, Linkoping, Sweden.

Atallah, M. (1999). *Algorithms and Theory of Computation Handbook*. CRC Press.

Ausiello, G., Marchetti-Spaccamela, A., Gambosi, G., Protasi, M., Crescenzi, P. & Kann, V. (1999). *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, Berlin.

Bagchi, T. (1999). *Multiobjective Scheduling by Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA.

Barr, R., Golden, B., Kelly, J., Resende, M. & Stewart, W. (1995). Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, **1**.

Bernal-Agustin, J. (1998). *Aplicación de Algoritmos Genéticos al Diseño Óptimo de Sistemas de Distribuición de Energía Eléctrica*. Ph.D. thesis, Universidad de Zaragoza.

Bishop, J., Nasuto, S. & De Meyer, K. (2002). *Dynamic Knowledge Representation in Connectionist Systems*, 308–313. No. 2415 in Lecture Notes in Computer Science, Springer.

Borůvka, O. (1926). O jistém problému minimálním (about a certain minimal problem). *Práca Moravské Přirodovědecké Společnosti*, **3**.

Botee, H. & Bonabeau, E. (1998). Evolving ant colony optimization. *Adv. Complex Systems*, **1**, 149–159.

Bovet, D. & Crescenzi, P. (1994). *Introduction to the theory of complexity*. Prentice Hall International (UK) Ltd., Hertfordshire, UK.

Bui, L., Essam, D., Abbass, H. & Green, D. (2001). Performance analysis of evolutionary multiobjective optimization methods in noisy environments. Tech. Rep. TR-ALAR-200504006, University of New South Wales, Australia.

Camerini, P., Galbiati, G. & Maffioli, F. (1980). Complexity of spanning tree problems: Part I. *European Journal Operation Research*, **5**, 346–352.

Camerini, P., Galbiati, G. & Maffioli, F. (1983). On the complexity of finding multi-constrained spanning trees. *Discrete Applied Mathematics*, **5**, 39–50.

Camerini, P., Galbiati, G. & Maffioli, F. (1984). The complexity of weighted multi-constrained spanning tree problems. *Colloquia Mathematica Societais Janos Bolyai*, **44**, 53–101.

Cardoso, P. & Jesus, M. (2004). Plataforma computacional para a representação e simulação de redes de fluxos. *Tecnovisão*.

Cardoso, P., Jesus, M. & Márquez, A. (2003a). MONACO - Multi-Objective Network

optimisation based on an ACO. In *X Encuentros de Geometria Computacional*, Universidad de Sevilla, Sevilla.

Cardoso, P., Jesus, M. & Márquez, A. (2003b). Multiple objective TSP based on ACO. In *III Encuentro Andaluz de Matemáticas Discretas*, Universidad de Almeria, Almeria.

Cardoso, P., Jesus, M. & Márquez, A. (2004). Determinação de minimum spanning trees multi-objectivos baseadas num ACO. In *Congresso de Métodos Computacionais em Engenharia*, LNEC, Lisboa.

Cardoso, P., Jesus, M. & Márquez, A. (2005a). MONACO applied to the multiple criteria minimum spanning trees problem. Tech. rep., EST-Universidade do Algarve.

Cardoso, P., Jesus, M. & Márquez, A. (2005b). Multiple criteria minimum spanning trees. In *XI Encuentros de Geometria Computacional*, Universidad de Santander, Santander, Spain.

Cardoso, P., Jesus, M. & Márquez, A. (2006a). MOST – multiple objective spanning trees repository project. *To be submited to the OR Society*.

Cardoso, P., Jesus, M. & Márquez, A. (2006b). MOST Project. `http://est.ualg.pt/adec/csc/most`.

Carvalho, P., Ferreira, L. & Barruncho, L. (2001). On spanning-tree recombination in evolutionary large-scale network problems - application to electrical distribution planning. *IEEE Transactions on Evolutionary Computation*, **5**, 623–630.

Chang, Y., Bergman, L., Castelli, V., Li, C., Lo, M. & Smith, J. (2000). The onion technique: indexing for linear optimization queries. *SIGMOD Rec.*, **29**, 391–402.

Chazelle, B. (2000). A minimum spanning tree algorithm with inverse-ackermann type complexity. *Journal of the ACM*, **47**, 1028–1047.

Chazelle, B., Rubinfeld, R. & Trevisan, L. (2005). Approximating the minimum spanning tree weight in sublinear time. *Society for Industrial and Applied Mathematics*, **34**, 1370–1379.

Cirasella, J., Johnson, D., McGeoch, L. & Zhang, W. (2001). The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. *Lecture Notes in Computer Science*, **2153**.

Cisco Systems, ed. (2005). *Cisco IOS Switching Services Configuration Guide*, chap. Overview of Routing between Virtual LANs. Cisco Systems.

Coello-Coello, C. (1996). *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design*. Ph.D. thesis, Tulane University.

Coello-Coello, C. (2000). An updated survey of GA-based multiobjective optimization techniques. *ACM Comput. Surv.*, **32**, 109–143.

Cong, J., He, L., Koh, C. & Madden, P. (1996). Performance optimization of VLSI interconnect layout. *Integration, the VLSI Journal*, **21**, 1–94.

Coppersmith, D. & Lotker, Z. (1996). On Cayley's formula for counting trees in nested interval graphs. *CHI'96*.

Corley, H. (1985). Efficient spanning trees. *Journal of optimization theory and applications*, **45**, 481–485.

Cortés, C., Marquez, A., Nakamoto, A. & Valenzuela, J. (2005). Quadrangulations and 2-colorations. In *21st European Workshop on Computational Geometry (EWCG 2005)*, Eindhoven.

de Berg, M., van Kreveld, M., Overmars, M. & Schwarzkopf, O. (1997). *Computational Geometry - Algorithms and Applications*. Springer-Verlag, Berlin.

De Meyer, K. (2003). *Foundations of Stochastic Diffusion Search*. Ph.D. thesis, University of Reading, UK.

Deb, K. (2001). *Multi-objective Optimization using Evolutionary Algorithms*. John Wiley & sons.

Deb, K., Agrawal, S., Pratap, A. & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo & H. Schwefel, eds., *Parallel Problem Solving from Nature – PPSN VI*, 849–858, Springer, Berlin.

Dell'Amico, M. & Maffioli, F. (1996). On some multicriteria arborescence problems: complexity and algorithms. *Discrete Applied Mathematics*, **65**, 191–206.

Dell'Amico, M. & Maffioli, F. (2000). Combining linear and non-linear objectives in spanning tree problems. *J. Comb. Optim.*, **4**, 253–269.

Doerner, K., Gutjahr, W., Hartl, R., Strauss, C. & Strummer, C. (2001). Ant colony optimization in multiobjective portfolio selection. In *Proceedings of the 4th Metaheuristics Intl. Conf., MIC'2001, July 16–20, Porto, Portugal*, 243–248.

Dorigo, M. & Stützle, T. (1999). The ant colony optimization metaheuristic: Algorithms. In F. Glover & G. Kochenberger., eds., *to appear in Handbook of Metaheuristics*.

Dorigo, M. & Stutzle, T. (2004). *Ant Colony Optimization*. MIT Press.

Dorigo, M., Maniezzo, V. & Colorni, A. (1996). The ant system: Optimization by a

colony of cooperating agents. *IEEE Transaction on Systems, Man and Cybernetics*, **26**, 29–41.

Dorigo, M., Bonabeau, E. & Theraulaz, G. (1999). *Swarm Intelligence: From Natural to artificial Systems*. Oxford University Press.

Eberhart, R. & Kennedy, J. (1995a). A new optimizer using particles swarm theory. In *Sixth International Symposium on Micro Machine and Human Science*, 39–43.

Eberhart, R. & Kennedy, J. (1995b). Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, 1942–1948.

Eberhart, R., Shi, Y. & Kennedy, J. (2001). *Swarm Intelligence*. Morgan Kaufmann.

Ehrgott, M. & Dandibleux, X. (2000). A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*.

Fadili, M., Melkemi, M. & ElMoataz, A. (2004). Non-convex onion-peeling using a shape hull algorithm. *Pattern Recognition Letters, Elsevier*, **25**, 1577–1585.

Fonseca, C. (1995). *Multiobjective Genetic Algorithms with Application to Control Engineering Problems*. Ph.D. thesis, Sheffield University.

Fonseca, V., Fonseca, C. & Hall, A. (2001). Inferential performance assessment of stochastic optimisers and the attainment function. In E. Zitzler, K. Deb, L. Thiele, C. Coello-Coello & D. Corne, eds., *Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001*, vol. 1993 of *Lecture Notes in Computer Science*, Springer-Verlag.

Gabow, H. & Myers, E. (1978). Finding all spanning trees of directed and undirected graphs. *SIAM J. Comput.*, **7**, 280–287.

Gabow, H., Galil, Z. & Spencer, T. (1989). Efficient implementation of graph algorithms using contraction. *Journal of the ACM*, **36**, 540–572.

Gambardella, L. & Dorigo, M. (2000). An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS, Journal on Computing*, **12**.

Gambardella, L., Taillard, E. & Agazzi, G. (1999). *New Ideas in Optimization*, chap. MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows, 63–79. McGraw-Hill, London, UK.

García-Martínez, C., Cordón, O. & Herrera, F. (2004a). An empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. *ANTS Workshop, Lecture Notes in Computer Science*, **3172**, 61–72.

García-Martínez, C., Cordón, O. & Herrera, F. (2004b). A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. Tech. rep., Dept. of Computer Science and Artificial Intelligence, E.T.S. de Ingeniería Informática. University of Granada, SPAIN.

Garey, M. & Johnson, D. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. Freeman & Co., New York, NY, USA.

Gen, M. & Li, Y.Z. (1998). Spanning tree-based genetic algorithm for bicriteria transportation problem. In *ICC&IE: Selected papers from the 22nd ICC&IE conference on Computers & industrial engineering*, 531–534, Elsevier Science Ltd., Oxford, UK.

Glover, F. & Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

Graham, R. & Hell, P. (1985). On the history of the minimum spanning tree problem. *Annals of the History of Computing*, **7**, 43–57.

Hamacher, H. & Ruhe, G. (1994). On spanning tree problems with multiple objective. *Annals of operation research*, 2309–230.

Hernández-Pérez, H. & Salazar-González, J. (2004). A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Appl. Math.*, **145**, 126–139.

Hodgson, P. (2002). Inside reality: Extreme immersion elevates virtual reality to a new level. *CSEG Recorder*, **27**, 57–59.

Holland, J. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor.

Hu, T. (1974). Optimum communication spanning trees. *Journal of Computation*, **3**.

Jarník, V. (1930). O jistém problému minimálním (in czech). *Prace Moravske Prirodovedecke Spolecnosti*, **6**, 57–63.

Jaszkiewicz, A. (2001). *Multiple Objective Metaheuristic Algorithms for Combinatorial Optimization*. Ph.D. thesis, Poznan University of Technology.

Jaszkiewicz, A. (2002). Genetic local search for multiple objective combinatorial optimization. *European Journal of Operational Research*, **137**, 50–71.

Jaszkiewicz, A. (2006a). A comparative experiment with multiple objective genetic local search algorithm on multi-objective travelling salesperson problem. `http://www-idss.cs.put.poznan.pl/~jaszkiewicz/motsp/index.htm`.

Jaszkiewicz, A. (2006b). http://www-idss.cs.put.poznan.pl/ ∼jaszkiewicz/.

Jesus, M. (2000). *Steiner Trees Approximations using Genetic Algorithms*. Ph.D. thesis, University of Seville, Spain.

Johnson, D. & McGeoch, L. (1997). *Local Search in Combinatorial Optimisation*, chap. The Traveling Salesman Problem: A Case Study in Local Optimization, 215–310. E. Aarts and J. Lenstra (editors), John Wiley and Sons Ltd.

Johnson, S. (2001). *Emergence. The connected lives of ants, brains, cities, and software*. Scribner.

Jungnickel, D. (1999). *Graphs, Networks and Algorithms*, vol. 5 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin.

Kirkpatrick, S., Gelatt, C. & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, **220, 4598**, 671–680.

Kleint, P. & Tarjan, R. (1994). A randomized linear-time algorithm for finding minimum spanning trees. *Journal of the ACM*.

Knowles, J. (2002). *Local-search and hybrid evolutionary algorithms for Pareto optimization*. Ph.D. thesis, University of Reading, United Kingdom.

Knowles, J. & Corne, D. (2001a). Benchmark problem generators and results for the multiobjective degree-constrained minimum spanning tree problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 424–431, Morgan Kaufmann Publishers, San Francisco, California.

Knowles, J. & Corne, D. (2001b). A comparative assessment of memetic, evolutionary, and constructive algorithms for the multiobjective d-MST problem. In *2nd Workshop On Memetic Algorithms, WOMA2001*.

Knowles, J. & Corne, D. (2002). On metrics for comparing non-dominated sets. In *Congress on Evolutionary Computation (CEC 2002)*, vol. 1.

Knowles, J., Corne, D. & Oates, M. (1999). A new evolutionary approach to the degree constrained minimum spanning tree problem. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith, eds., *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 1, 794, Morgan Kaufmann, Orlando, Florida, USA.

Knowles, J., Thiele, L. & Zitzler, E. (2005). A tutorial on the performance assessment of stochastic multiobjective optimizers. Tech. rep., Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich.

Knuth, D. (1973). *The Art of Computer Programming: Sorting and searching*, vol. 3. Addison-Wesley.

Kozanidis, G. & Melachrinoudis, E. (2004). A branch & bound algorithm for the 0-1 mixed integer knapsack problem with linear multiple choice constraints. *Comput. Oper. Res.*, **31**, 695–711.

Kruskal, J. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, vol. 7.

Laskari, E., Parsopoulos, K. & Vrahatis, M. (2002). Particle swarm optimization for minimax problems. In *IEEE 2002 Congress on Evolutionary Computation*, 1582–1587.

Leung, Y., Li, G. & Xu, Z. (1998). A genetic algorithm for the multiple destination routing problems. *IEEE Trans. on Evolutionary Computation*, **2**, 150–161.

Levine, J. & Ducatelle, F. (2004). Ant colony optimisation and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society, Special Issue on Local Search*, **55**.

Mariano-Romero, C. & Morales-Manzanares, E. (1999). MOAQ an Ant-Q Algorithm for Multiple Objective Optimization Problems. In W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela & R. Smith, eds., *Genetic and Evolutionary Computing Conference (GECCO 99)*, vol. 1, 894–901, Morgan Kaufmann, San Francisco, California.

Masbaum, G. (2002). Matrix-tree theorems and the alexander-conway polynomial. *GEOM.TOPOL.MONOGR.*, **4**, 201.

Middendorf, M., Reischle, F. & Schmeck, H. (2002). Multi colony ant algorithms. *Journal of Heuristics*, **8**, 305–320.

Miettinen, K. (1999). *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers.

Mitchell, J. (2002). *Handbook of Applied Optimization*, chap. Branch-and-Cut Algorithms for Combinatorial Optimization Problems. Oxford University Press.

Montemanni, R. & Gambardella, L. (2003). A branch and bound algorithm for the robust spanning tree problem with interval data. *European Journal of Operation Research*.

Mostaghim, S., Teich, J. & Tyagi, A. (2002). Comparison of data structures for storing Pareto-sets in MOEAs. In *Evolutionary Computation (CEC'2002)*, vol. 1, IEEE Service Center, Piscataway, New Jersey.

Murata, T. (1997). *Genetic Algortithms for Multi-Objective Optimization*. Ph.D. thesis, Osaka Prefecture University.

Nakrani, S. & Tovey, C. (2004). On honey bees and dynamic server allocation in

internet hosting centers. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, **12**, 223–240.

Nešetřil, J. (1997). A few remarks on the history of MST problem. *Archivum Mathematicum*, **33**, 15–22.

Norman, B. (1993). *Algebraic graph theory*. Cambridge Mathematical Library.

Parmee, I. (2001). *Evolutionary and Adaptive Computing in Engineering Design*. Springer-Verlag, London, iSNB:1-85233-029-5.

Pinto, D., Barán, B. & Fabregat, R. (2005). Multi-objective multicast routing based on Ant Colony Optimization. In *CCIA 2005*, L'Alguer.

Pirzadeh, H. (1999). *Computational Geometry with the Rotating Callipers*. Master's thesis, McGill University, Canada.

Poulos, M., Evangelou, A., Magkos, E. & Papavlasopoulos, S. (2004). Fingerprint verification based on image processing segmentation using an onion algorithm of computational geometry. In *Sixth Int. Conf. on Mathematics Methods in Scattering Theory and Biomedical Technology (BIOTECH'6)*.

Prim, R. (1957). Shortest connection network and some generalizations. *Bell System Tech.*, **36**, 1389–1401.

Raidl, G. & Julstrom, B. (2000). A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. In *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing*, 440–445, ACM Press, New York, NY, USA.

Ramos, R., Alonso, S., Sicilia, J. & González, C. (1998). The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, **111**, 617–628.

Ray, T. & Liew, K. (2002). A swarm metaphor for multiobjective design optimization. *Engineering Optimization*, **34**, 141–153.

Rechenberg, I. (1973). *Evolutionsstrategie – Optimierung technisher Systeme nach Prinzipien der biologischen Evolution*. Ph.D. thesis, Stuttgart, GER.

Resnick, M. (1996). Starlogo: An environment for decentralized modeling and decentralized thinking. *CHI'96*.

Resnick, M. (1999). *Turtles, Termites, and Traffic Jams: Exploration in Massively Parallel Microworlds*. MIT Press.

Resnick, M. (2004). StarLogo. `http://education.mit.edu/starlogo`.

Ribeiro, C. & Resende, M. (2001). Greedy randomized adaptive search procedures. Tech. rep., AT&T Labs Research, to appear in State of the Art Handbook in Metaheuristics, F. Glover and G. Kochenberger, eds., Kluwer, 2002.

Sack, J.R. & Urrutia, J. (2000). *Handbook of Computational Geometry*. Elsevier Science, North-Holland.

Sait, S. & Youseff, H. (1997). *Iterative Computer Algorithms with Applications in Engineering*. IEEE - Computer Society.

Saleh, H. & Dare, P. (2002). Heuristics for improved efficiency in the use of the global navigation satellite systems (GNSS) for establishing positioning networks. **2**.

Schott, J. (1995). *Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization*. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Schwefel, H.P. (1977). *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie*. Ph.D. thesis, Basel.

Shibuchi, H. & Murata, T. (1998). Multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, **3**.

Shioura, A., Tamura, A. & Uno, T. (1997). An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM Journal on Computing*, **26**.

Sinclair, M. (2001). *Evolutionary Algorithms for Optical Network Design: A Genetic-algorithm/Heuristic Hybrid Approach*. Ph.D. thesis, University of Essex.

Sourd, F., Spanjaard, O. & Perny, P. (2006). Multi-objective branch-and-bound. application to the bi-objective spanning tree problem. In *Int. Conf. on Multi-Objective Programming and Goal Programming*.

Steiner, S. & Radzik, T. (2003). Solving the biobjective minimum spanning tree problem using a k-best algorithm. Tech. rep., King's College London, United Kingdom.

Tarasewich, P. & McMullen, P. (2002). Swarm intelligence - power in the numbers. *Communications of the ACM*, **5**, 149–159.

Torres-Jimenez, J., Garcia-Romero, A. & Pinto-Elias, R. (1999). A simulated annealing approach for the capacitated minimum spanning tree problem. *ICCIMA*, **00**, 432.

Toussaint, G. (1983). Solving geometric problems with the rotating calipers. In *Proceedings of IEEE MELECON'83*, Athens, Greece.

Tovey, C. (2004). The honey bee algorithm: A biological inspired approach to internet server optimization. *Engineering enterpise*, 13–15.

TSPLib, 2003 (2003). TSPLib. `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95`.

van Veldhuizen, D. (1999). *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. Ph.D. thesis, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

van Veldhuizen, D. & Lamont, G. (1999). Multiobjective Evolutionary Algorithm Test Suites. In J. Carroll, H. Haddad, D. Oppenheim, B. Bryant & G. Lamont, eds., *Proceedings of the 1999 ACM Symposium on Applied Computing*, 351–357, ACM, San Antonio, Texas.

Vose, M. (1999). *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press,.

Yu, H., Das, S., Lim, Y. & Gerla, M. (2003). Efficient building method of multiple spanning tree for QoS and load balancing. *GLOBECOM 2003*.

Zhang, W. (1993). Truncated branch-and-bound: A case study on the asymmetric TSP. In *Working Note of AAAI 1993 Spring Symposium: AI and NP-Hard Problems*, 160–166.

Zhou, G. & Gen, M. (1999). Genetic Algorithm Approach on Multi-Criteria Minimum Spanning Tree Problem. *European Journal of Operational Research*, **114**.

Zitzler, E. (1999). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Ph.D. thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

Zitzler, E. & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary computation*, **3**.

Zitzler, E., Deb, K. & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, **8**, 173–195.

Zitzler, E., Laumanns, M. & Thiele, L. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Tech. Rep. 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland.

Zitzler, E., Laumanns, M. & Thiele, L. (2002a). SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papaliliou & T. Fogarty, eds., *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems. Proceedings of the EUROGEN2001 Conference, Athens, Greece, September 19-21, 2001*, 95–100, International Center for Numerical Methos in Engineering (CIMNE), Barcelona, Spain.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. & Fonseca, V. (2002b). Why quality assessment of multiobjective optimizers is difficult. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, 666–674.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. & Fonseca, V. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary computation*, **7**.

A

Binary Relations

A $\propto$-binary (endo-)relation is defined by a pair (X, G), where X is a set and G is a subset of the product $X \times X$. If $(x, y) \in G$ then x is $\propto$-related to y and is usually denoted as $x \propto y$.

Some important classes of binary relations over a set X are:

Reflexive $\forall_{x \in X} : x \propto x$.

Irreflexive $\forall_{x \in X} : x \not\propto x$.

Co-reflexive $\forall_{x,y \in X} : x \propto y \Rightarrow x = y$

Symmetric $\forall_{x,y \in X} : x \propto y \Rightarrow y \propto x$

Antisymmetric $\forall_{x,y \in X} : x \propto y \wedge y \propto x \Rightarrow x = y$

Transitive $\forall_{x,y,z \in X} : x \propto y \wedge y \propto z \Rightarrow x \propto z$

Total $\forall_{x,y \in X} : x \propto y \vee y \propto x$

Trichotomous $\forall_{x,y \in X} : x \propto y \dot{\vee} y \propto x \dot{\vee} x = y$

A relation is a(n)

(Weak) partial order if is reflexive, antisymmetric and transitive

Strict partial order if is irreflexive and transitive

Total order if is trichotomous and transitive

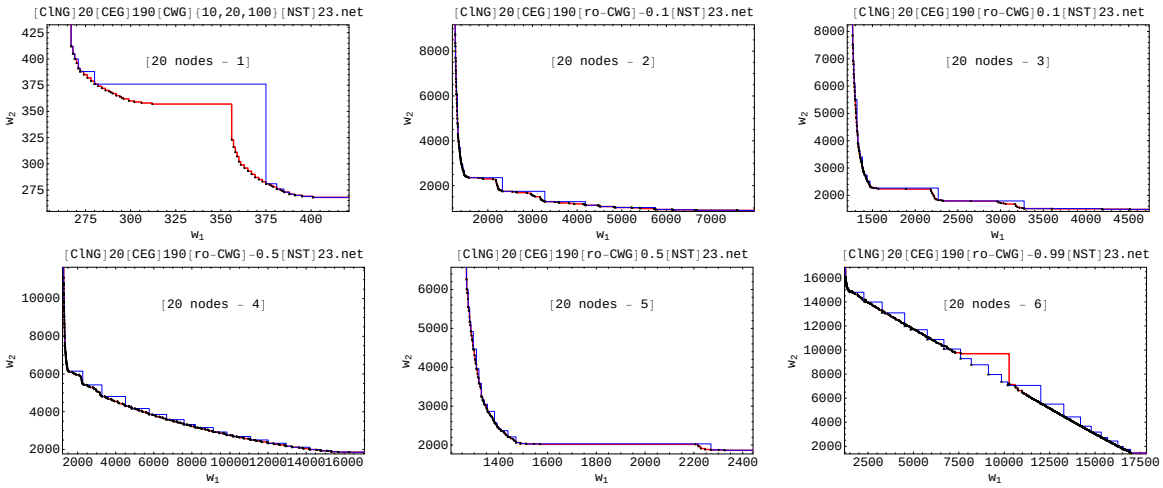
Equivalence if is reflexive, symmetric and transitive

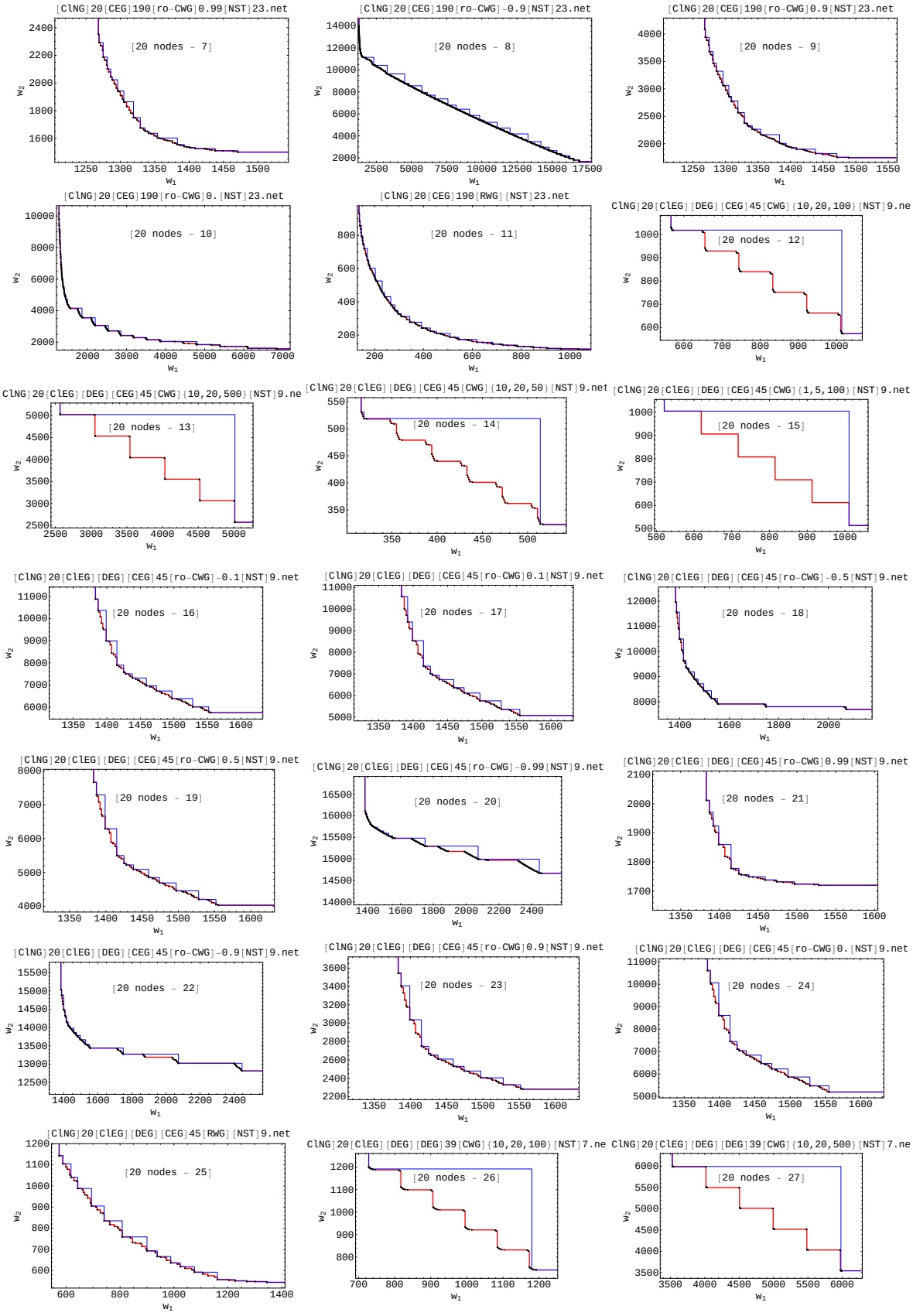
B

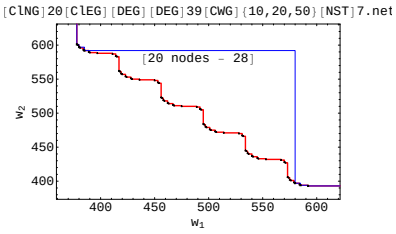
Pareto and Approximation Fronts

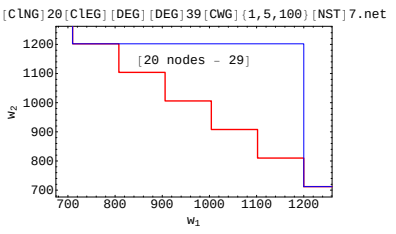
In this appendix are sketched the approximation sets obtained with the Weighted Sum (blue line) and the ϵ-DANTE (red line) methods.

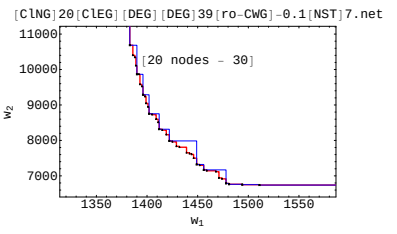
B.1 20 Nodes Networks

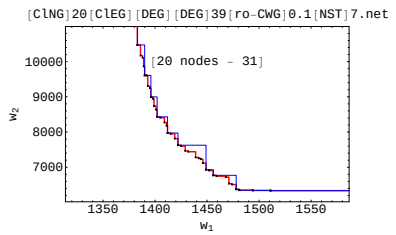


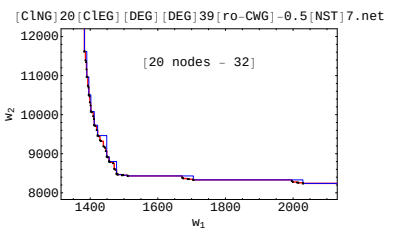

[ClNG]20[CEG]190[ro-CWG]0.99[NST]23.net
[20 nodes - 7]
[ClNG]20[CEG]190[ro-CWG]-0.9[NST]23.net
[20 nodes - 8]
[ClNG]20[CEG]190[ro-CWG]0.9[NST]23.net
[20 nodes - 9]
[ClNG]20[CEG]190[ro-CWG]0.[NST]23.net
[20 nodes - 10]
[ClNG]20[CEG]190[RWG][NST]23.net
[20 nodes - 11]
ClNG]20[ClEG][DEG][CEG]45[CWG]{10,20,100}[NST]9.ne
[20 nodes - 12]
ClNG]20[ClEG][DEG][CEG]45[CWG]{10,20,500}[NST]9.ne
[20 nodes - 13]
[ClNG]20[ClEG][DEG][CEG]45[CWG]{10,20,50}[NST]9.net
[20 nodes - 14]
[ClNG]20[ClEG][DEG][CEG]45[CWG]{1,5,100}[NST]9.net
[20 nodes - 15]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]-0.1[NST]9.net
[20 nodes - 16]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]0.1[NST]9.net
[20 nodes - 17]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]-0.5[NST]9.net
[20 nodes - 18]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]0.5[NST]9.net
[20 nodes - 19]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]-0.99[NST]9.net
[20 nodes - 20]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]0.99[NST]9.net
[20 nodes - 21]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]-0.9[NST]9.net
[20 nodes - 22]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]0.9[NST]9.net
[20 nodes - 23]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]0.[NST]9.net
[20 nodes - 24]
[ClNG]20[ClEG][DEG][CEG]45[RWG][NST]9.net
[20 nodes - 25]
ClNG]20[ClEG][DEG][DEG]39[CWG]{10,20,100}[NST]7.ne
[20 nodes - 26]
ClNG]20[ClEG][DEG][DEG]39[CWG]{10,20,500}[NST]7.ne
[20 nodes - 27]
w1
w2

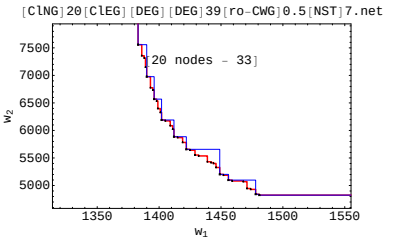

[ClNG]20[ClEG][DEG][DEG]39[CWG]{10,20,50}[NST]7.net
[20 nodes - 28]
w2
w1

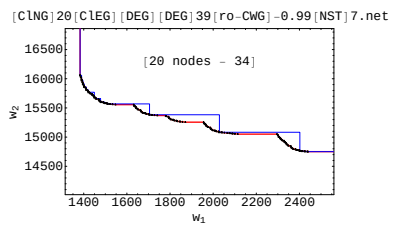

[ClNG]20[ClEG][DEG][DEG]39[CWG]{1,5,100}[NST]7.net
[20 nodes - 29]
w2
w1

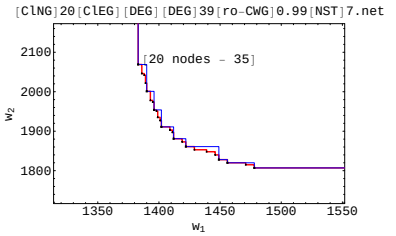

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]-0.1[NST]7.net
[20 nodes - 30]
w2
w1

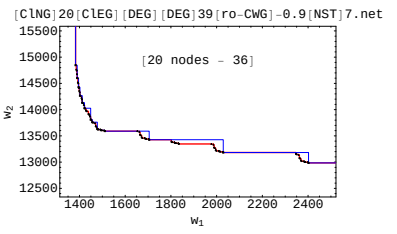

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]0.1[NST]7.net
[20 nodes - 31]
w2
w1

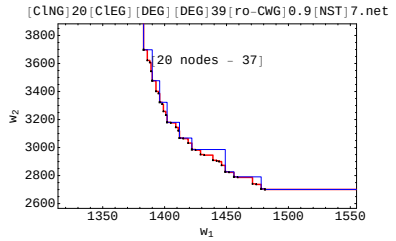

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]-0.5[NST]7.net
[20 nodes - 32]
w2
w1

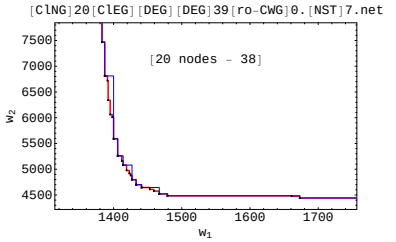

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]0.5[NST]7.net
[20 nodes - 33]
w2
w1

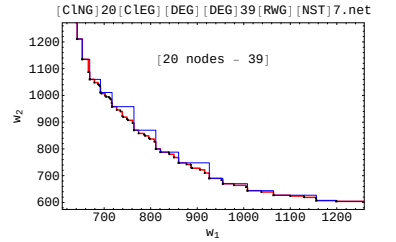

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]-0.99[NST]7.net
[20 nodes - 34]
w2
w1

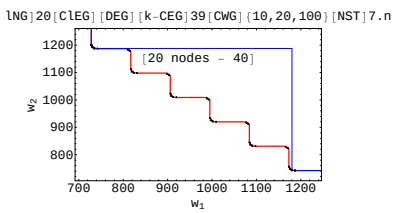

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]0.99[NST]7.net
[20 nodes - 35]
w2
w1

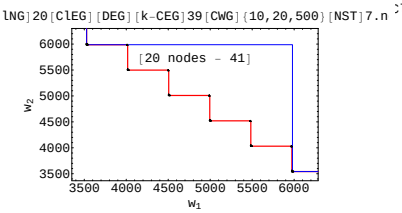

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]-0.9[NST]7.net
[20 nodes - 36]
w2
w1

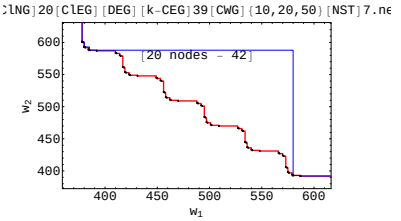

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]0.9[NST]7.net
[20 nodes - 37]
w2
w1

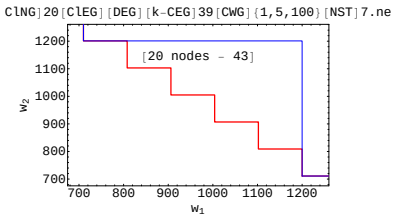

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]0.[NST]7.net
[20 nodes - 38]
w2
w1

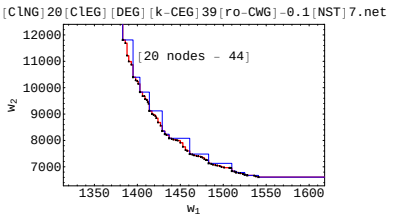

[ClNG]20[ClEG][DEG][DEG]39[RWG][NST]7.net
[20 nodes - 39]
w2
w1

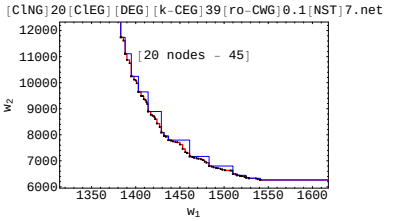

lNG]20[ClEG][DEG][k-CEG]39[CWG]{10,20,100}[NST]7.n
[20 nodes - 40]
w2
w1

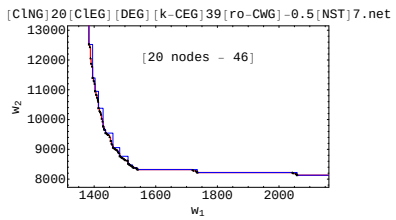

lNG]20[ClEG][DEG][k-CEG]39[CWG]{10,20,500}[NST]7.n
[20 nodes - 41]
w2
w1

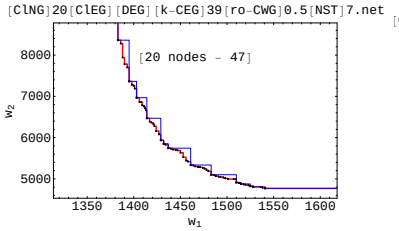

lNG]20[ClEG][DEG][k-CEG]39[CWG]{10,20,50}[NST]7.ne
[20 nodes - 42]
w2
w1

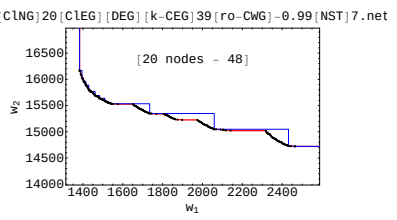

ClNG]20[ClEG][DEG][k-CEG]39[CWG]{1,5,100}[NST]7.ne
[20 nodes - 43]
w2
w1

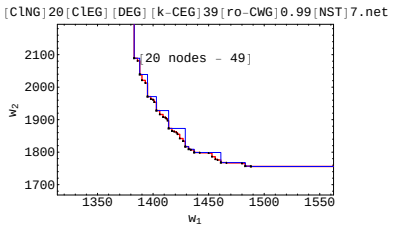

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]-0.1[NST]7.net
[20 nodes - 44]
w2
w1

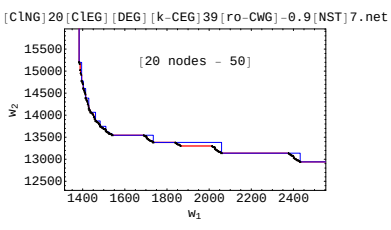

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]0.1[NST]7.net
[20 nodes - 45]
w2
w1

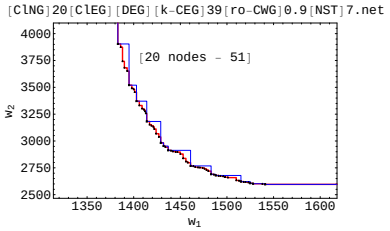

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]-0.5[NST]7.net
[20 nodes - 46]
w2
w1

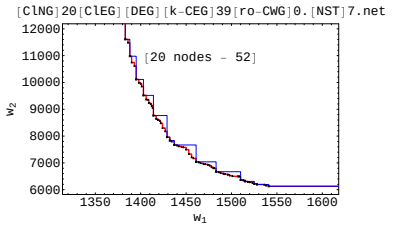

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]0.5[NST]7.net
[20 nodes - 47]
w2
w1

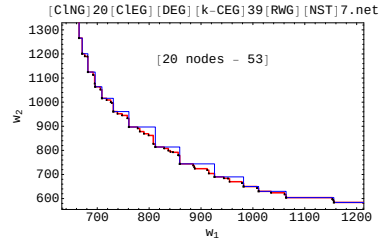

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]-0.99[NST]7.net
[20 nodes - 48]
w2
w1

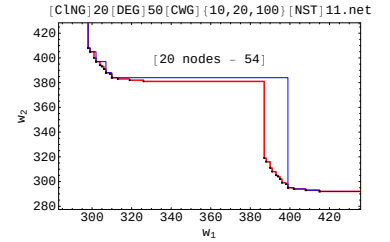

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]0.99[NST]7.net
[20 nodes - 49]
w2
w1

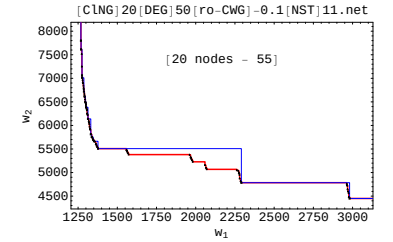

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]-0.9[NST]7.net
[20 nodes - 50]
w2
w1

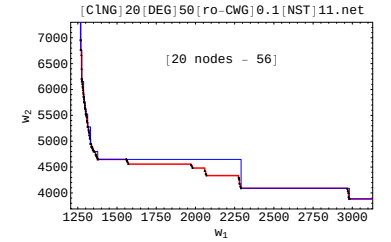

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]0.9[NST]7.net
[20 nodes - 51]
w2
w1

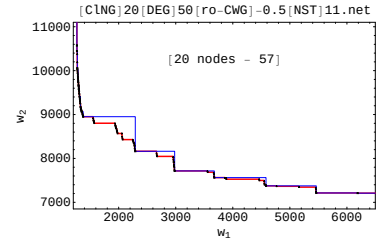

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]0.[NST]7.net
[20 nodes - 52]
w2
w1

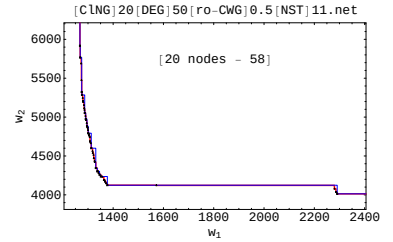

[ClNG]20[ClEG][DEG][k-CEG]39[RWG][NST]7.net
[20 nodes - 53]
w2
w1

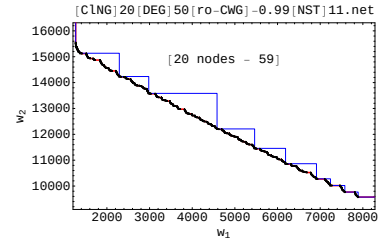

[ClNG]20[DEG]50[CWG]{10,20,100}[NST]11.net
[20 nodes - 54]
w2
w1

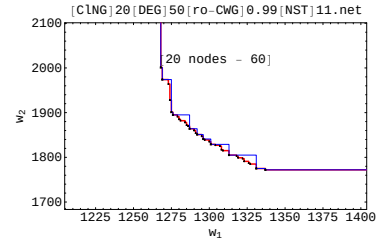

[ClNG]20[DEG]50[ro-CWG]-0.1[NST]11.net
[20 nodes - 55]
w2
w1

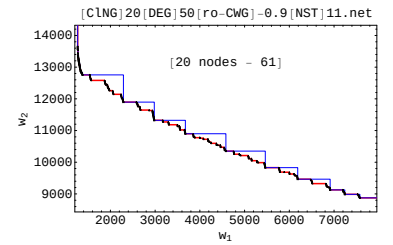

[ClNG]20[DEG]50[ro-CWG]0.1[NST]11.net
[20 nodes - 56]
w2
w1

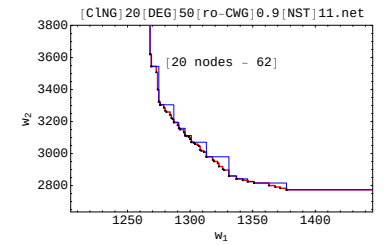

[ClNG]20[DEG]50[ro-CWG]-0.5[NST]11.net
[20 nodes - 57]
w2
w1

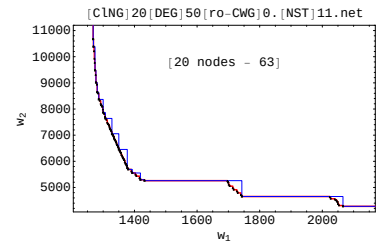

[ClNG]20[DEG]50[ro-CWG]0.5[NST]11.net
[20 nodes - 58]
w2
w1

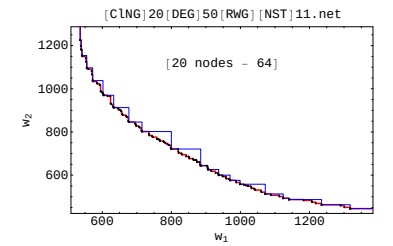

[ClNG]20[DEG]50[ro-CWG]-0.99[NST]11.net
[20 nodes - 59]
w2
w1

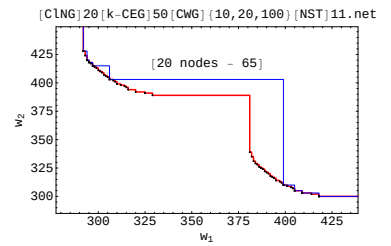

[ClNG]20[DEG]50[ro-CWG]0.99[NST]11.net
[20 nodes - 60]
w2
w1

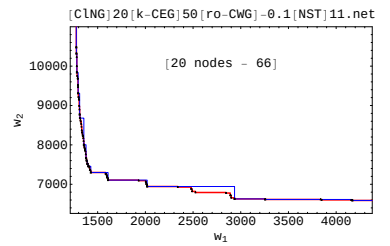

[ClNG]20[DEG]50[ro-CWG]-0.9[NST]11.net
[20 nodes - 61]
w2
w1

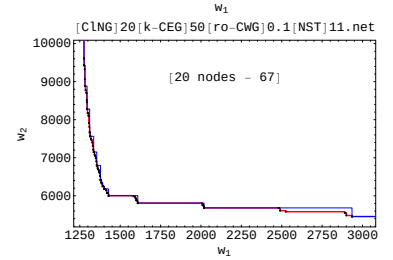

[ClNG]20[DEG]50[ro-CWG]0.9[NST]11.net
[20 nodes - 62]
w2
w1

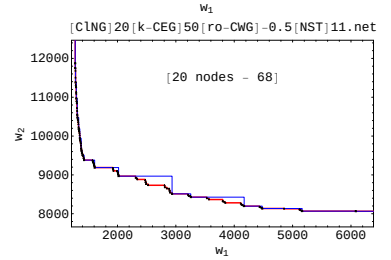

[ClNG]20[DEG]50[ro-CWG]0.[NST]11.net
[20 nodes - 63]
w2
w1

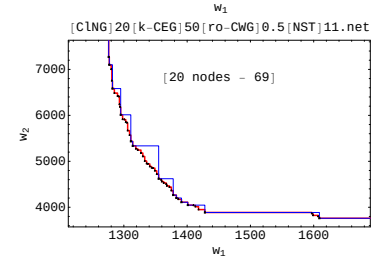

[ClNG]20[DEG]50[RWG][NST]11.net
[20 nodes - 64]
w2
w1

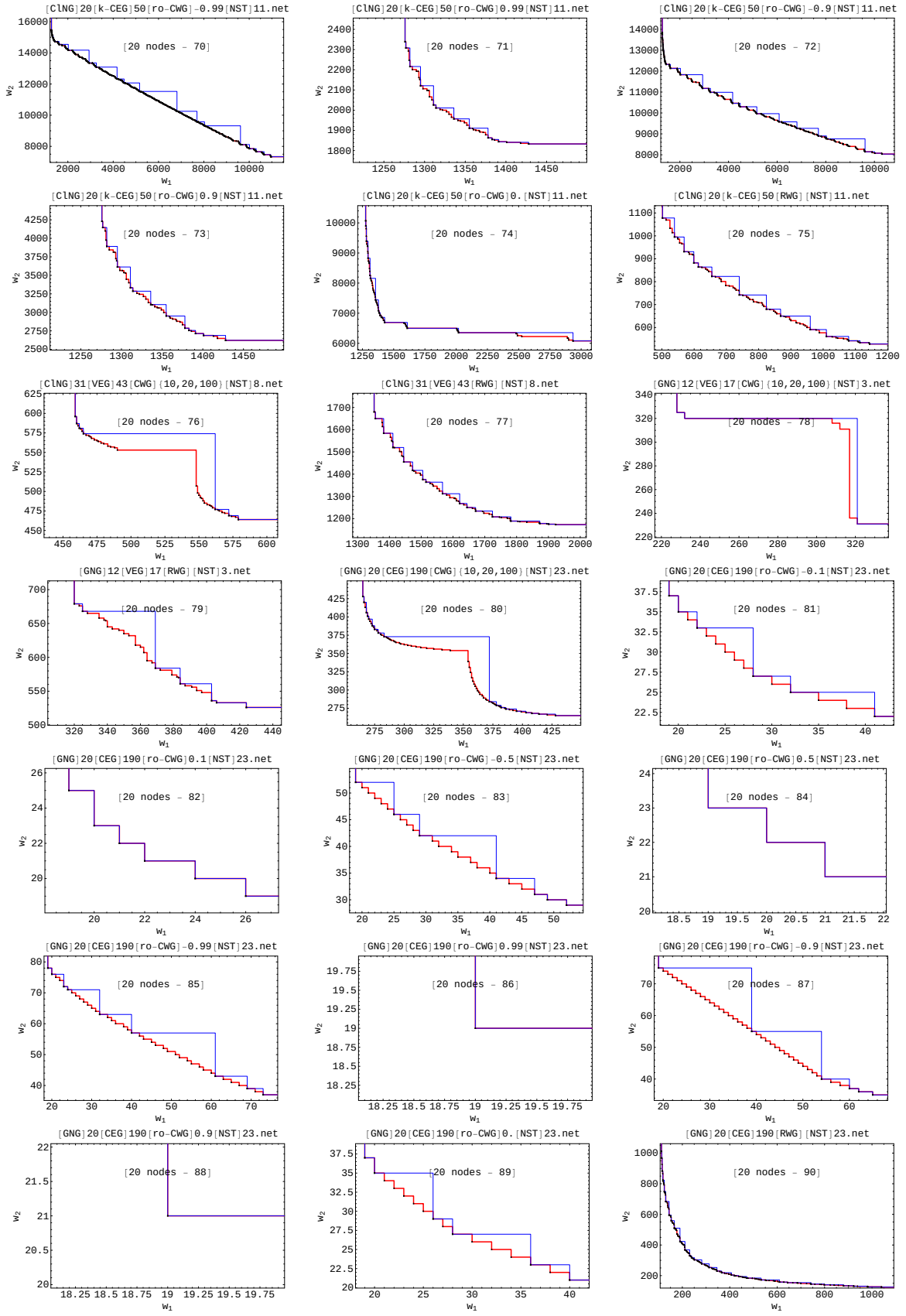

[ClNG]20[k-CEG]50[CWG]{10,20,100}[NST]11.net
[20 nodes - 65]
w2
w1

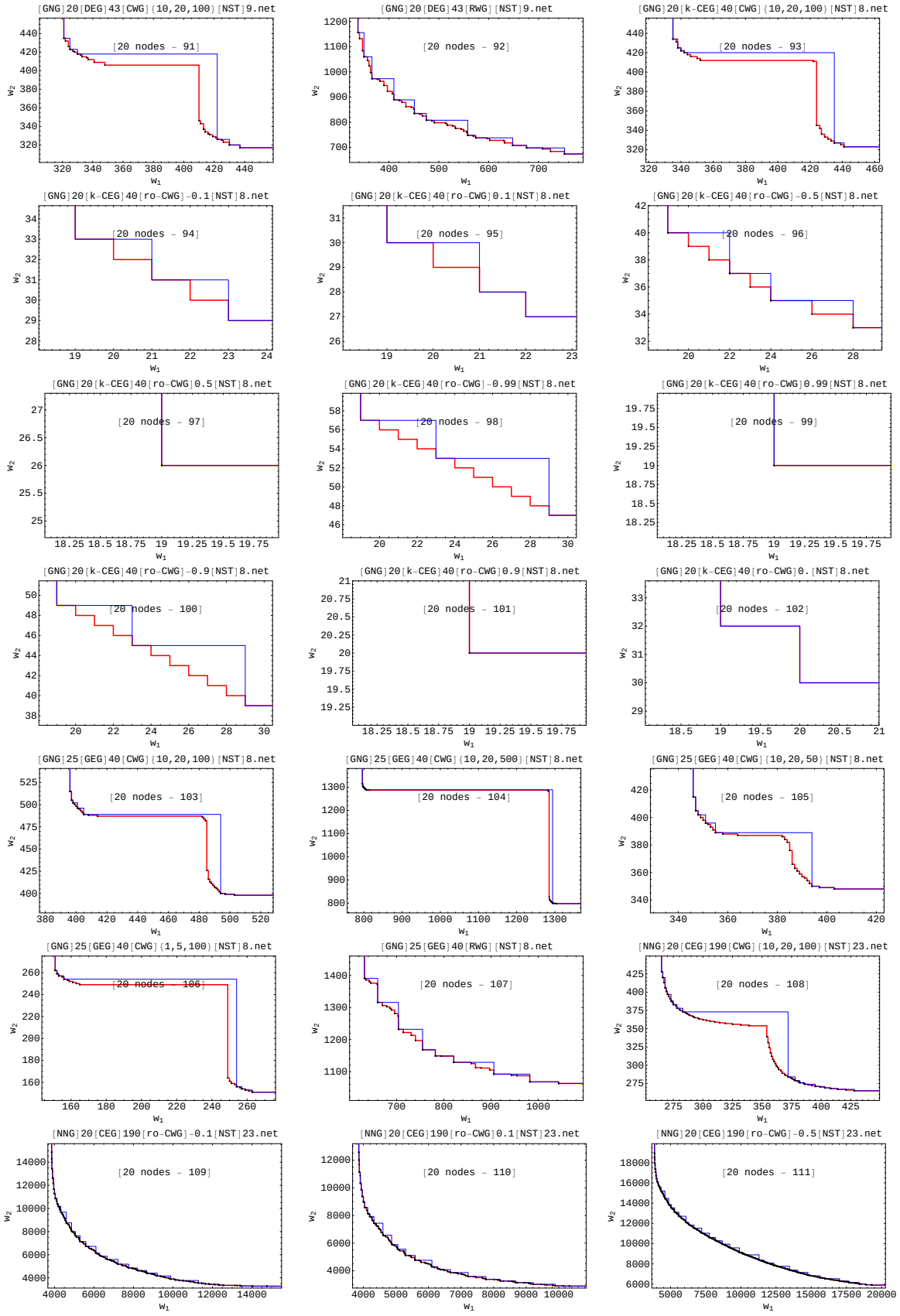

[ClNG]20[k-CEG]50[ro-CWG]-0.1[NST]11.net
[20 nodes - 66]
w2
w1

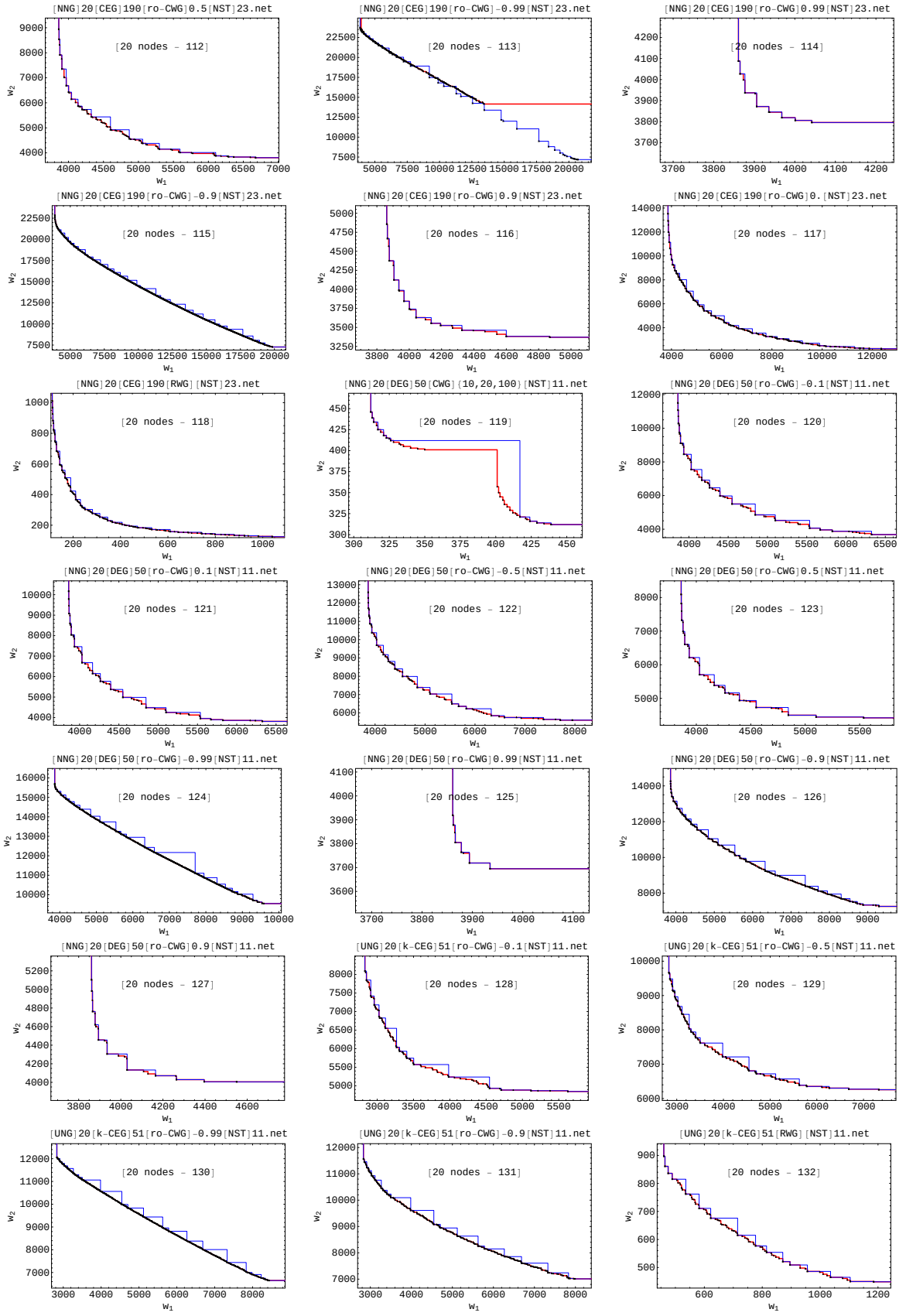

[ClNG]20[k-CEG]50[ro-CWG]0.1[NST]11.net
[20 nodes - 67]
w2
w1

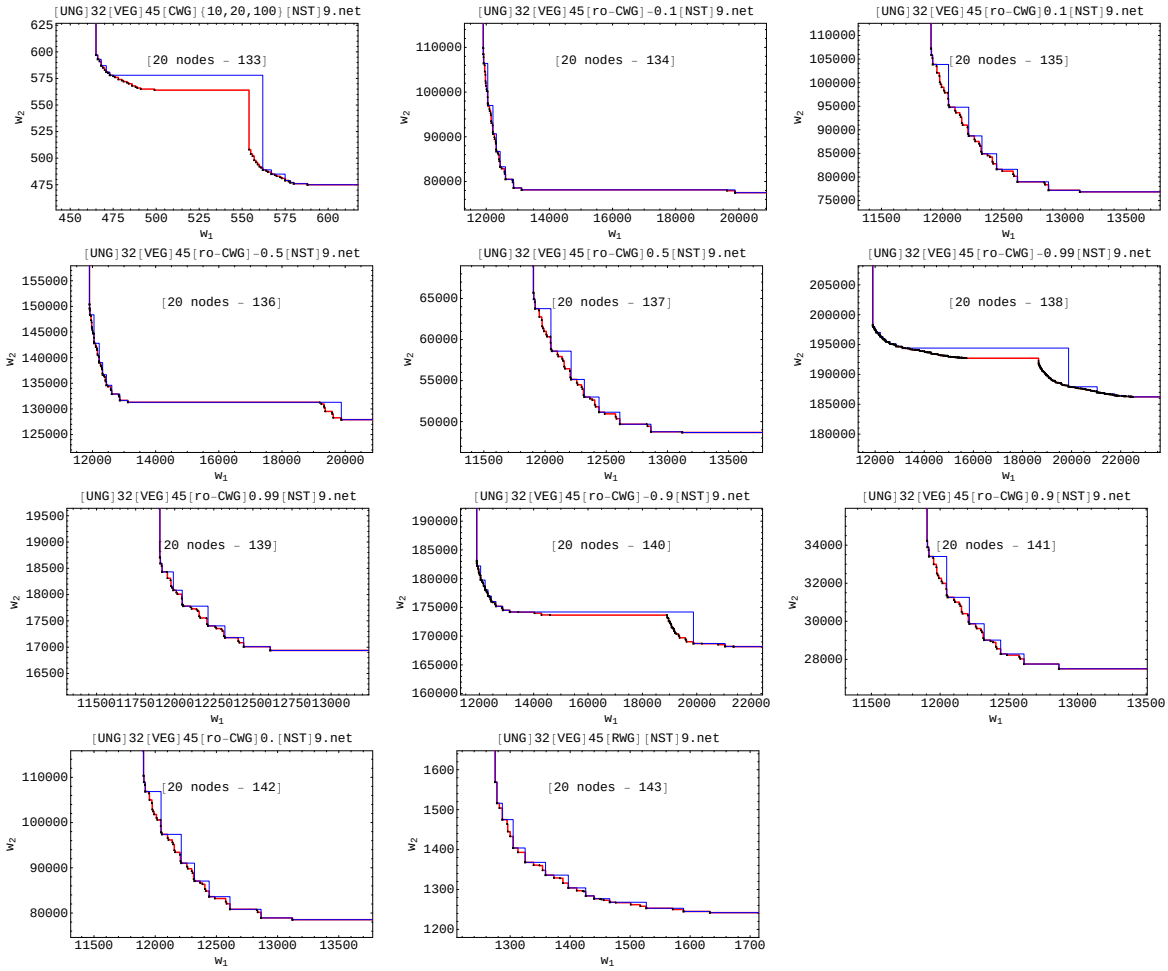

[ClNG]20[k-CEG]50[ro-CWG]-0.5[NST]11.net
[20 nodes - 68]
w2
w1


[ClNG]20[k-CEG]50[ro-CWG]0.5[NST]11.net
[20 nodes - 69]
w2
w1

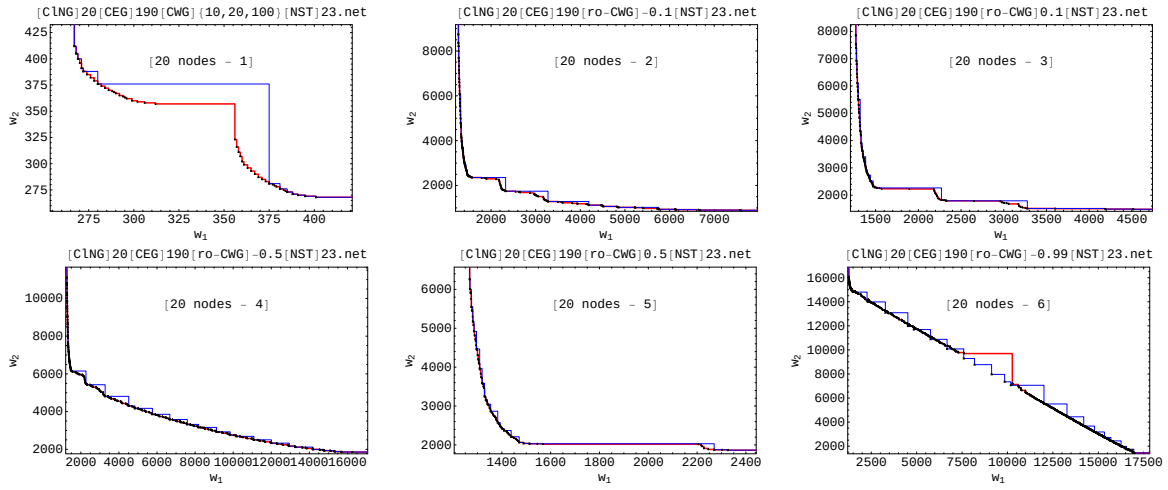

[ClNG]20[k-CEG]50[ro-CWG]-0.99[NST]11.net
[20 nodes - 70]
[ClNG]20[k-CEG]50[ro-CWG]0.99[NST]11.net
[20 nodes - 71]
[ClNG]20[k-CEG]50[ro-CWG]-0.9[NST]11.net
[20 nodes - 72]
[ClNG]20[k-CEG]50[ro-CWG]0.9[NST]11.net
[20 nodes - 73]
[ClNG]20[k-CEG]50[ro-CWG]0.[NST]11.net
[20 nodes - 74]
[ClNG]20[k-CEG]50[RWG][NST]11.net
[20 nodes - 75]
[ClNG]31[VEG]43[CWG]{10,20,100}[NST]8.net
[20 nodes - 76]
[ClNG]31[VEG]43[RWG][NST]8.net
[20 nodes - 77]
[GNG]12[VEG]17[CWG]{10,20,100}[NST]3.net
[20 nodes - 78]
[GNG]12[VEG]17[RWG][NST]3.net
[20 nodes - 79]
[GNG]20[CEG]190[CWG]{10,20,100}[NST]23.net
[20 nodes - 80]
[GNG]20[CEG]190[ro-CWG]-0.1[NST]23.net
[20 nodes - 81]
[GNG]20[CEG]190[ro-CWG]0.1[NST]23.net
[20 nodes - 82]
[GNG]20[CEG]190[ro-CWG]-0.5[NST]23.net
[20 nodes - 83]
[GNG]20[CEG]190[ro-CWG]0.5[NST]23.net
[20 nodes - 84]
[GNG]20[CEG]190[ro-CWG]-0.99[NST]23.net
[20 nodes - 85]
[GNG]20[CEG]190[ro-CWG]0.99[NST]23.net
[20 nodes - 86]
[GNG]20[CEG]190[ro-CWG]-0.9[NST]23.net
[20 nodes - 87]
[GNG]20[CEG]190[ro-CWG]0.9[NST]23.net
[20 nodes - 88]
[GNG]20[CEG]190[ro-CWG]0.[NST]23.net
[20 nodes - 89]
[GNG]20[CEG]190[RWG][NST]23.net
[20 nodes - 90]
w1
w2

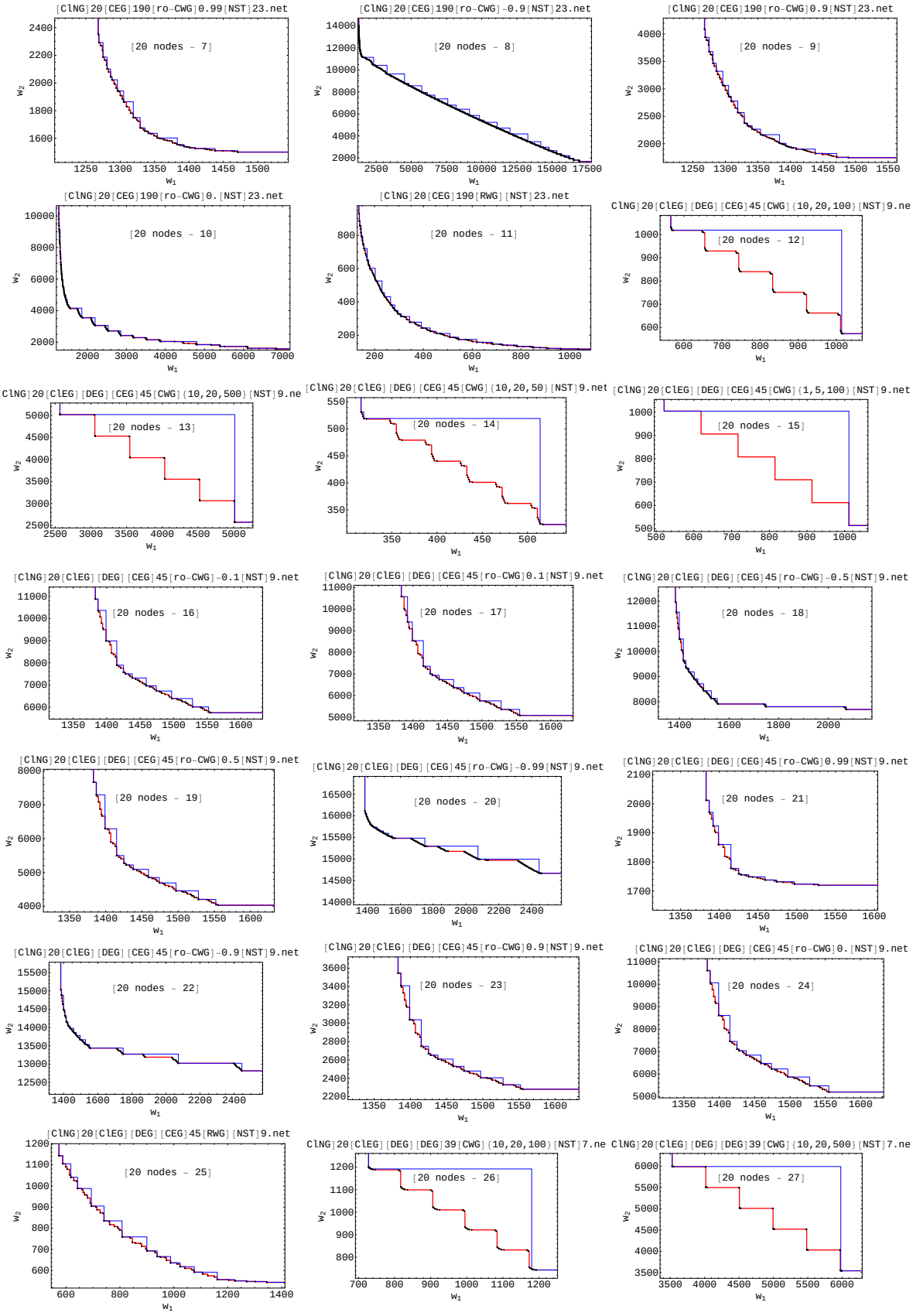

[GNG]20[DEG]43[CWG]{10,20,100}[NST]9.net
[20 nodes - 91]
[GNG]20[DEG]43[RWG][NST]9.net
[20 nodes - 92]
[GNG]20[k-CEG]40[CWG]{10,20,100}[NST]8.net
[20 nodes - 93]
[GNG]20[k-CEG]40[ro-CWG]-0.1[NST]8.net
[20 nodes - 94]
[GNG]20[k-CEG]40[ro-CWG]0.1[NST]8.net
[20 nodes - 95]
[GNG]20[k-CEG]40[ro-CWG]-0.5[NST]8.net
[20 nodes - 96]
[GNG]20[k-CEG]40[ro-CWG]0.5[NST]8.net
[20 nodes - 97]
[GNG]20[k-CEG]40[ro-CWG]-0.99[NST]8.net
[20 nodes - 98]
[GNG]20[k-CEG]40[ro-CWG]0.99[NST]8.net
[20 nodes - 99]
[GNG]20[k-CEG]40[ro-CWG]-0.9[NST]8.net
[20 nodes - 100]
[GNG]20[k-CEG]40[ro-CWG]0.9[NST]8.net
[20 nodes - 101]
[GNG]20[k-CEG]40[ro-CWG]0.[NST]8.net
[20 nodes - 102]
[GNG]25[GEG]40[CWG]{10,20,100}[NST]8.net
[20 nodes - 103]
[GNG]25[GEG]40[CWG]{10,20,500}[NST]8.net
[20 nodes - 104]
[GNG]25[GEG]40[CWG]{10,20,50}[NST]8.net
[20 nodes - 105]
[GNG]25[GEG]40[CWG]{1,5,100}[NST]8.net
[20 nodes - 106]
[GNG]25[GEG]40[RWG][NST]8.net
[20 nodes - 107]
[NNG]20[CEG]190[CWG]{10,20,100}[NST]23.net
[20 nodes - 108]
[NNG]20[CEG]190[ro-CWG]-0.1[NST]23.net
[20 nodes - 109]
[NNG]20[CEG]190[ro-CWG]0.1[NST]23.net
[20 nodes - 110]
[NNG]20[CEG]190[ro-CWG]-0.5[NST]23.net
[20 nodes - 111]
w1
w2

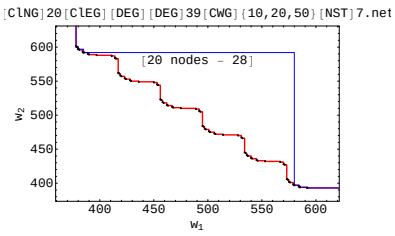

[NNG]20[CEG]190[ro-CWG]0.5[NST]23.net
[20 nodes - 112]
[NNG]20[CEG]190[ro-CWG]-0.99[NST]23.net
[20 nodes - 113]
[NNG]20[CEG]190[ro-CWG]0.99[NST]23.net
[20 nodes - 114]
[NNG]20[CEG]190[ro-CWG]-0.9[NST]23.net
[20 nodes - 115]
[NNG]20[CEG]190[ro-CWG]0.9[NST]23.net
[20 nodes - 116]
[NNG]20[CEG]190[ro-CWG]0.[NST]23.net
[20 nodes - 117]
[NNG]20[CEG]190[RWG][NST]23.net
[20 nodes - 118]
[NNG]20[DEG]50[CWG]{10,20,100}[NST]11.net
[20 nodes - 119]
[NNG]20[DEG]50[ro-CWG]-0.1[NST]11.net
[20 nodes - 120]
[NNG]20[DEG]50[ro-CWG]0.1[NST]11.net
[20 nodes - 121]
[NNG]20[DEG]50[ro-CWG]-0.5[NST]11.net
[20 nodes - 122]
[NNG]20[DEG]50[ro-CWG]0.5[NST]11.net
[20 nodes - 123]
[NNG]20[DEG]50[ro-CWG]-0.99[NST]11.net
[20 nodes - 124]
[NNG]20[DEG]50[ro-CWG]0.99[NST]11.net
[20 nodes - 125]
[NNG]20[DEG]50[ro-CWG]-0.9[NST]11.net
[20 nodes - 126]
[NNG]20[DEG]50[ro-CWG]0.9[NST]11.net
[20 nodes - 127]
[UNG]20[k-CEG]51[ro-CWG]-0.1[NST]11.net
[20 nodes - 128]
[UNG]20[k-CEG]51[ro-CWG]-0.5[NST]11.net
[20 nodes - 129]
[UNG]20[k-CEG]51[ro-CWG]-0.99[NST]11.net
[20 nodes - 130]
[UNG]20[k-CEG]51[ro-CWG]-0.9[NST]11.net
[20 nodes - 131]
[UNG]20[k-CEG]51[RWG][NST]11.net
[20 nodes - 132]
w1
w2

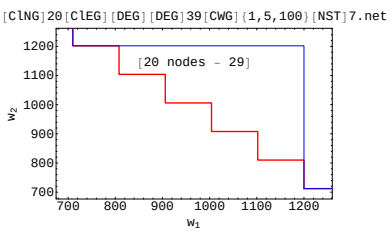


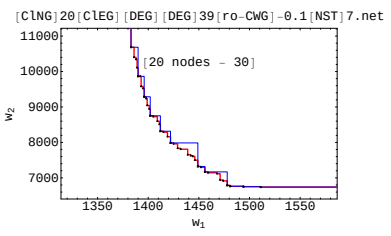
B.2 50 Nodes Networks

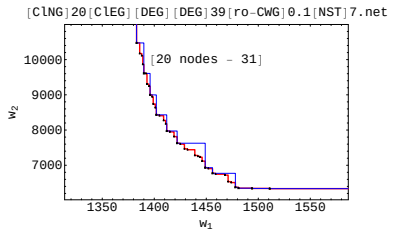


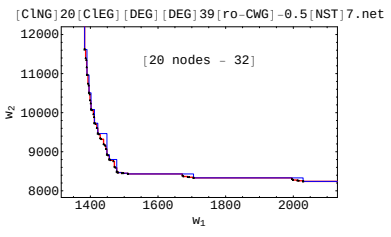

[ClNG]20[CEG]190[ro-CWG]0.99[NST]23.net
[20 nodes - 7]
[ClNG]20[CEG]190[ro-CWG]-0.9[NST]23.net
[20 nodes - 8]
[ClNG]20[CEG]190[ro-CWG]0.9[NST]23.net
[20 nodes - 9]
[ClNG]20[CEG]190[ro-CWG]0.[NST]23.net
[20 nodes - 10]
[ClNG]20[CEG]190[RWG][NST]23.net
[20 nodes - 11]
ClNG]20[ClEG][DEG][CEG]45[CWG]{10,20,100}[NST]9.ne
[20 nodes - 12]
ClNG]20[ClEG][DEG][CEG]45[CWG]{10,20,500}[NST]9.ne
[20 nodes - 13]
[ClNG]20[ClEG][DEG][CEG]45[CWG]{10,20,50}[NST]9.net
[20 nodes - 14]
[ClNG]20[ClEG][DEG][CEG]45[CWG]{1,5,100}[NST]9.net
[20 nodes - 15]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]-0.1[NST]9.net
[20 nodes - 16]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]0.1[NST]9.net
[20 nodes - 17]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]-0.5[NST]9.net
[20 nodes - 18]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]0.5[NST]9.net
[20 nodes - 19]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]-0.99[NST]9.net
[20 nodes - 20]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]0.99[NST]9.net
[20 nodes - 21]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]-0.9[NST]9.net
[20 nodes - 22]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]0.9[NST]9.net
[20 nodes - 23]
[ClNG]20[ClEG][DEG][CEG]45[ro-CWG]0.[NST]9.net
[20 nodes - 24]
[ClNG]20[ClEG][DEG][CEG]45[RWG][NST]9.net
[20 nodes - 25]
ClNG]20[ClEG][DEG][DEG]39[CWG]{10,20,100}[NST]7.ne
[20 nodes - 26]
ClNG]20[ClEG][DEG][DEG]39[CWG]{10,20,500}[NST]7.ne
[20 nodes - 27]
w1
w2

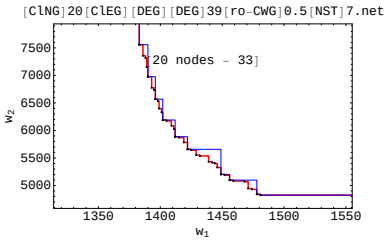

[ClNG]20[ClEG][DEG][DEG]39[CWG]{10,20,50}[NST]7.net
[20 nodes - 28]
w2
w1

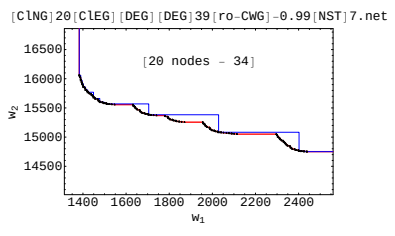

[ClNG]20[ClEG][DEG][DEG]39[CWG]{1,5,100}[NST]7.net
[20 nodes - 29]
w2
w1

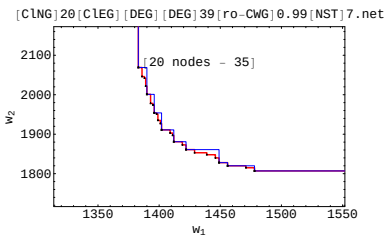

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]-0.1[NST]7.net
[20 nodes - 30]
w2
w1

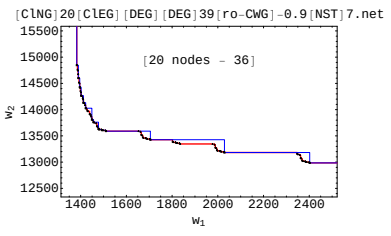

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]0.1[NST]7.net
[20 nodes - 31]
w2
w1

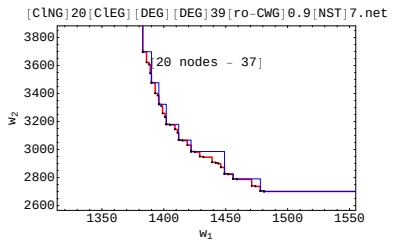

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]-0.5[NST]7.net
[20 nodes - 32]
w2
w1

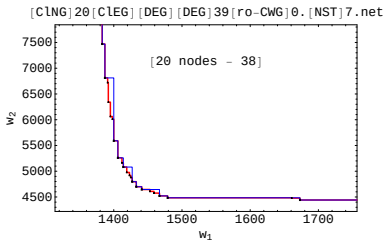

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]0.5[NST]7.net
[20 nodes - 33]
w2
w1

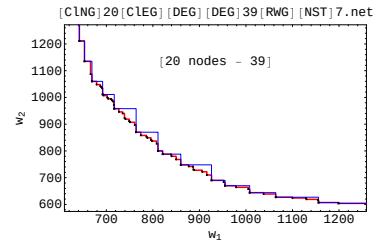

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]-0.99[NST]7.net
[20 nodes - 34]
w2
w1

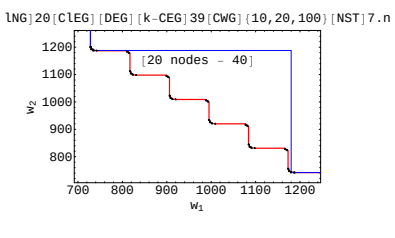

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]0.99[NST]7.net
[20 nodes - 35]
w2
w1

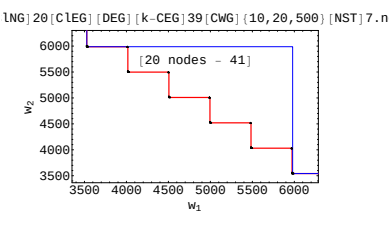

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]-0.9[NST]7.net
[20 nodes - 36]
w2
w1

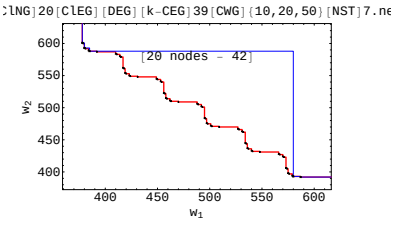

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]0.9[NST]7.net
[20 nodes - 37]
w2
w1

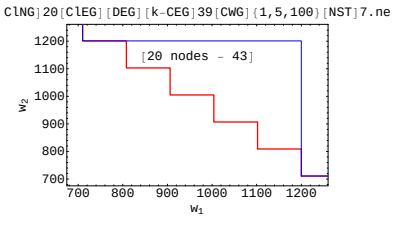

[ClNG]20[ClEG][DEG][DEG]39[ro-CWG]0.[NST]7.net
[20 nodes - 38]
w2
w1

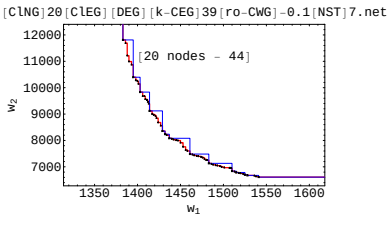

[ClNG]20[ClEG][DEG][DEG]39[RWG][NST]7.net
[20 nodes - 39]
w2
w1

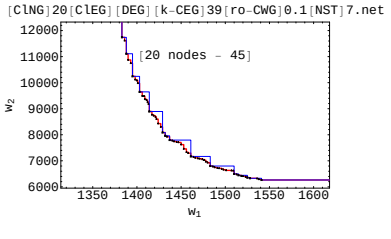

lNG]20[ClEG][DEG][k-CEG]39[CWG]{10,20,100}[NST]7.n
[20 nodes - 40]
w2
w1

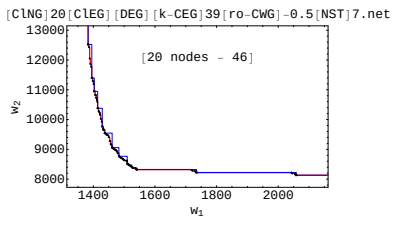

lNG]20[ClEG][DEG][k-CEG]39[CWG]{10,20,500}[NST]7.n
[20 nodes - 41]
w2
w1

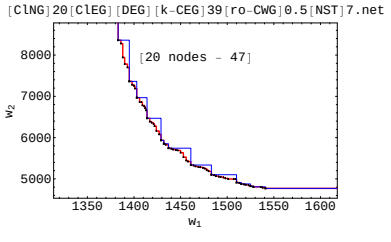

lNG]20[ClEG][DEG][k-CEG]39[CWG]{10,20,50}[NST]7.ne
[20 nodes - 42]
w2
w1

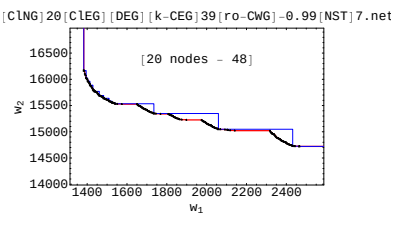

ClNG]20[ClEG][DEG][k-CEG]39[CWG]{1,5,100}[NST]7.ne
[20 nodes - 43]
w2
w1

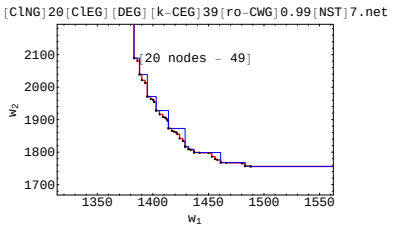

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]-0.1[NST]7.net
[20 nodes - 44]
w2
w1

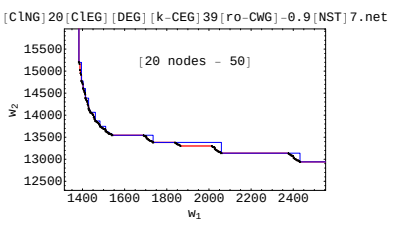

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]0.1[NST]7.net
[20 nodes - 45]
w2
w1

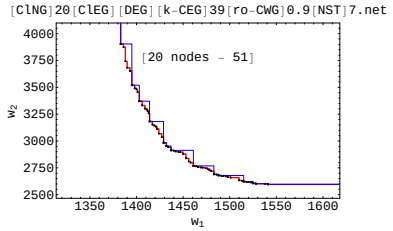

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]-0.5[NST]7.net
[20 nodes - 46]
w2
w1

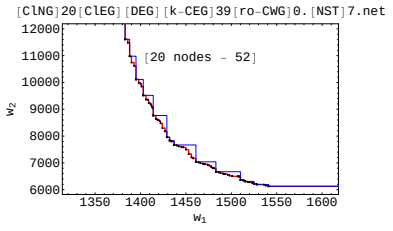

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]0.5[NST]7.net
[20 nodes - 47]
w2
w1

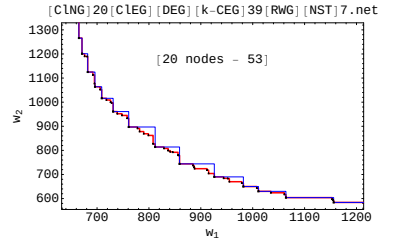

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]-0.99[NST]7.net
[20 nodes - 48]
w2
w1

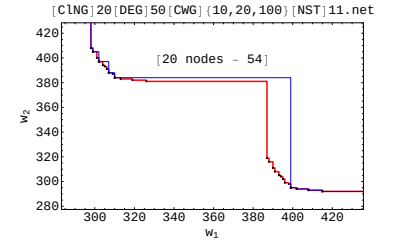

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]0.99[NST]7.net
[20 nodes - 49]
w2
w1

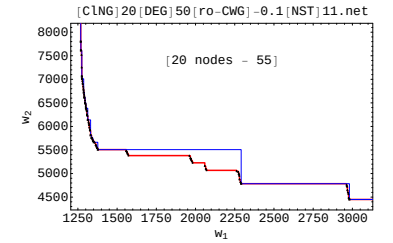

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]-0.9[NST]7.net
[20 nodes - 50]
w2
w1

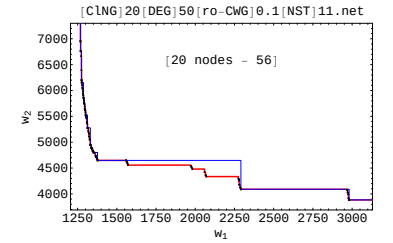

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]0.9[NST]7.net
[20 nodes - 51]
w2
w1

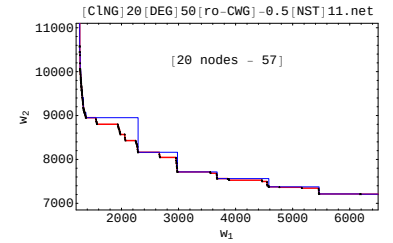

[ClNG]20[ClEG][DEG][k-CEG]39[ro-CWG]0.[NST]7.net
[20 nodes - 52]
w2
w1

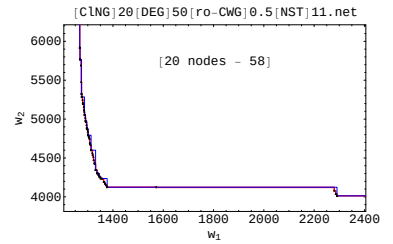

[ClNG]20[ClEG][DEG][k-CEG]39[RWG][NST]7.net
[20 nodes - 53]
w2
w1

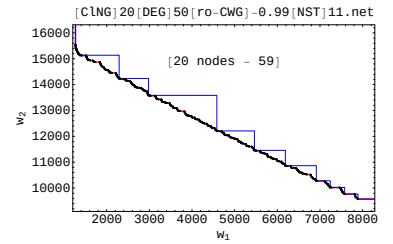

[ClNG]20[DEG]50[CWG]{10,20,100}[NST]11.net
[20 nodes - 54]
w2
w1

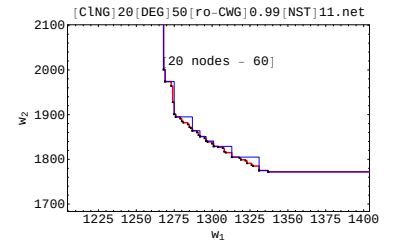

[ClNG]20[DEG]50[ro-CWG]-0.1[NST]11.net
[20 nodes - 55]
w2
w1

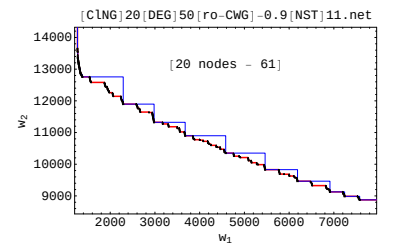

[ClNG]20[DEG]50[ro-CWG]0.1[NST]11.net
[20 nodes - 56]
w2
w1

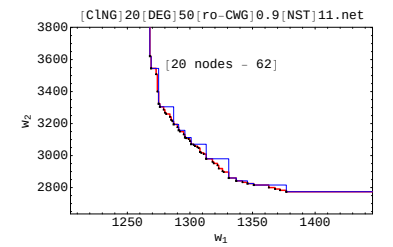

[ClNG]20[DEG]50[ro-CWG]-0.5[NST]11.net
[20 nodes - 57]
w2
w1

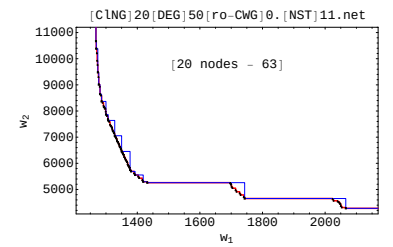

[ClNG]20[DEG]50[ro-CWG]0.5[NST]11.net
[20 nodes - 58]
w2
w1

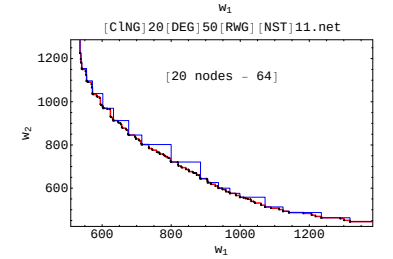

[ClNG]20[DEG]50[ro-CWG]-0.99[NST]11.net
[20 nodes - 59]
w2
w1

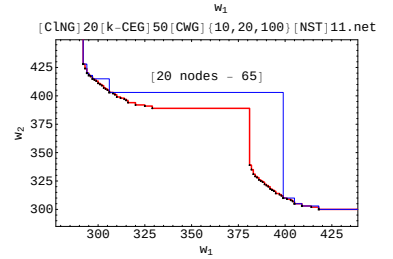

[ClNG]20[DEG]50[ro-CWG]0.99[NST]11.net
20 nodes - 60]
w2
w1

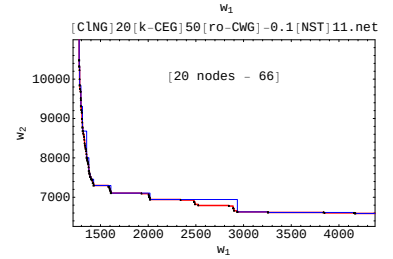

[ClNG]20[DEG]50[ro-CWG]-0.9[NST]11.net
[20 nodes - 61]
w2
w1

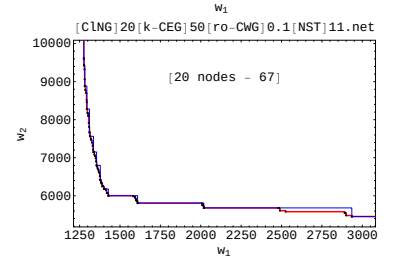

[ClNG]20[DEG]50[ro-CWG]0.9[NST]11.net
[20 nodes - 62]
w2
w1

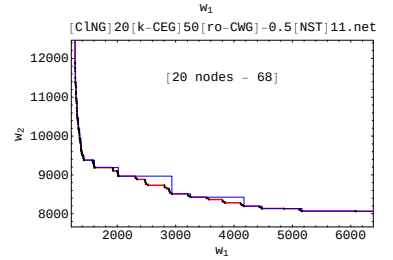

[ClNG]20[DEG]50[ro-CWG]0.[NST]11.net
[20 nodes - 63]
w2
w1

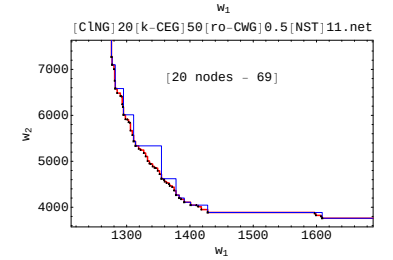

[ClNG]20[DEG]50[RWG][NST]11.net
[20 nodes - 64]
w2
w1

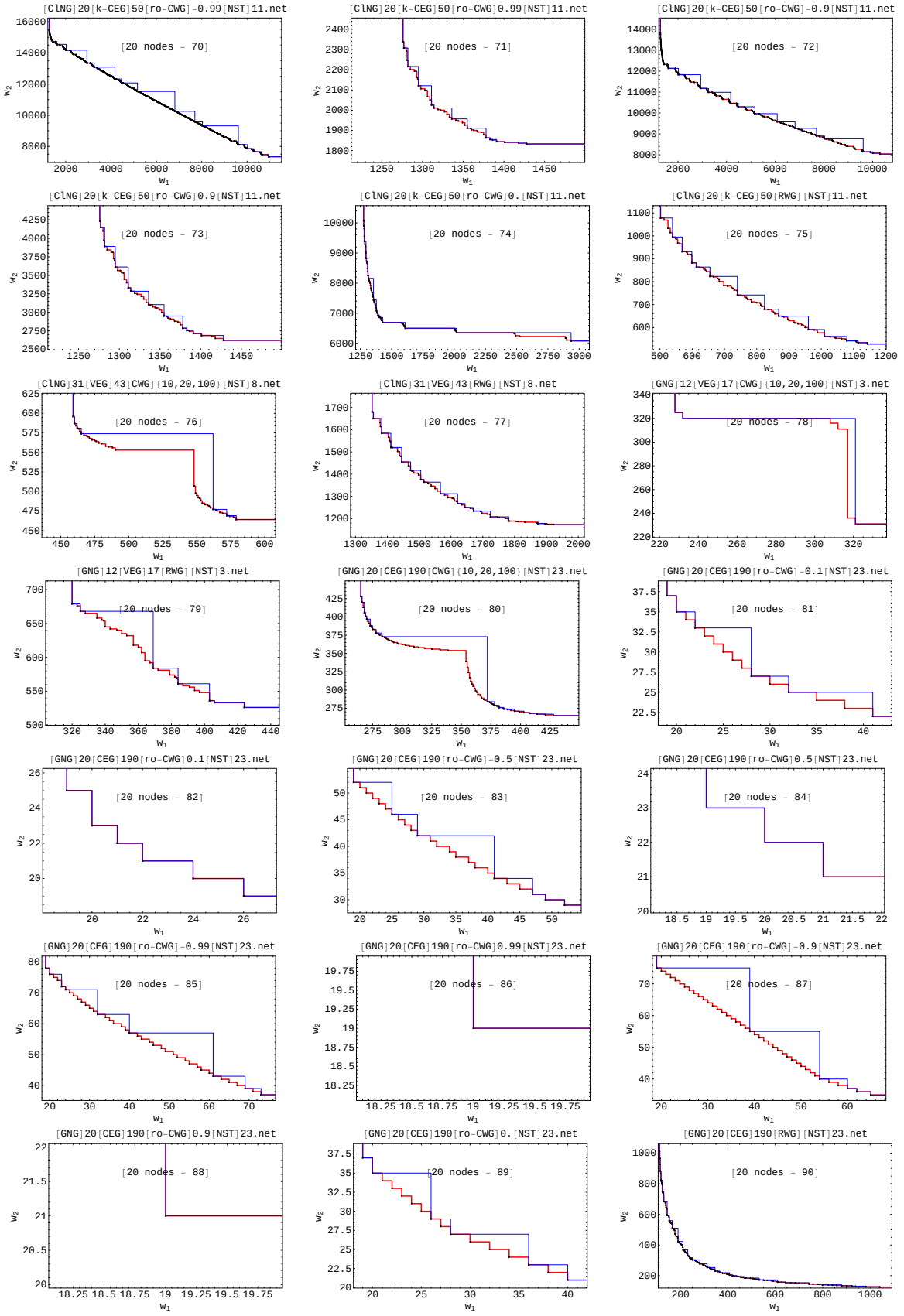

[ClNG]20[k-CEG]50[CWG]{10,20,100}[NST]11.net
[20 nodes - 65]
w2
w1

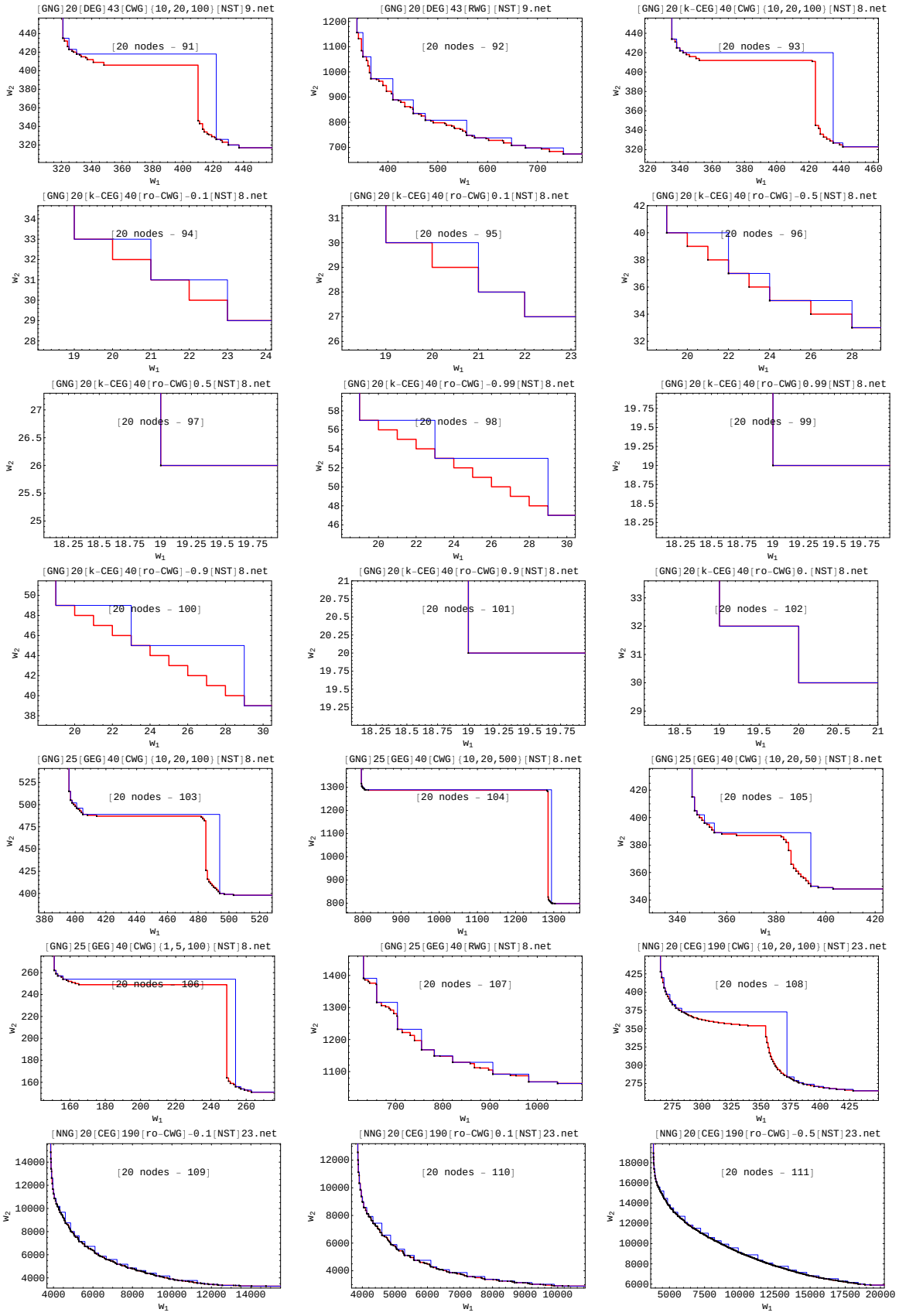

[ClNG]20[k-CEG]50[ro-CWG]-0.1[NST]11.net
[20 nodes - 66]
w2
w1

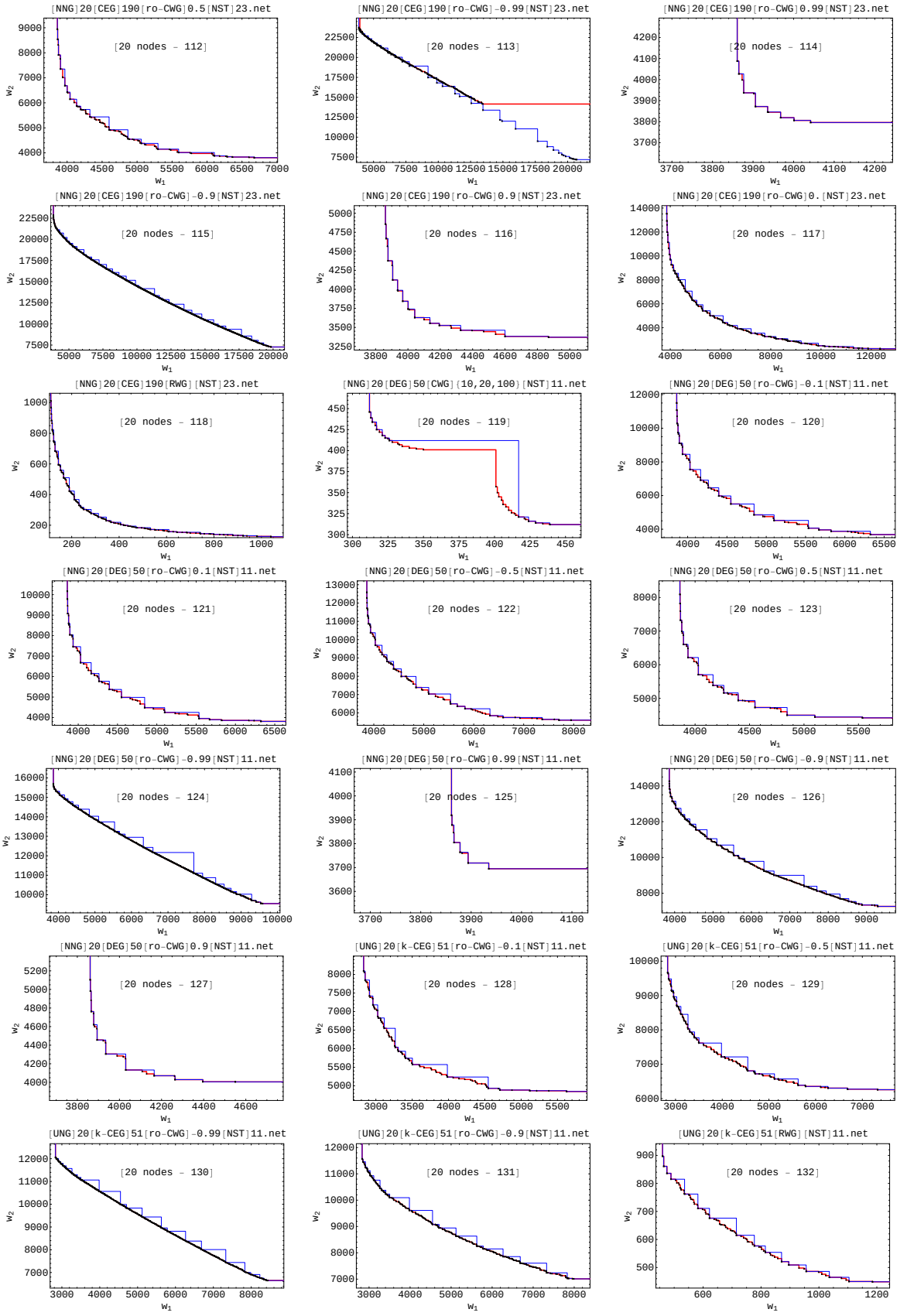

[ClNG]20[k-CEG]50[ro-CWG]0.1[NST]11.net
[20 nodes - 67]
w2
w1

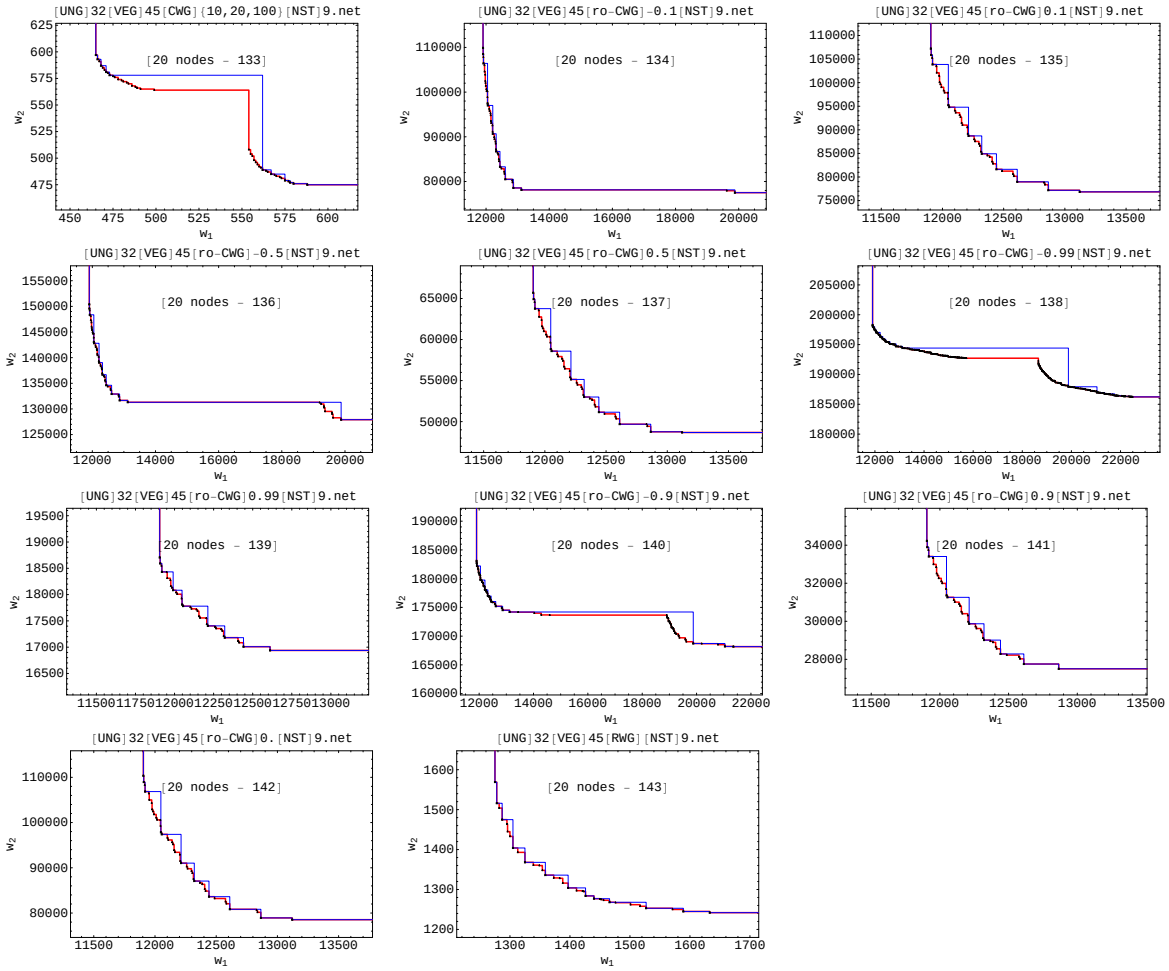

[ClNG]20[k-CEG]50[ro-CWG]-0.5[NST]11.net
[20 nodes - 68]
w2
w1


[ClNG]20[k-CEG]50[ro-CWG]0.5[NST]11.net
[20 nodes - 69]
w2
w1

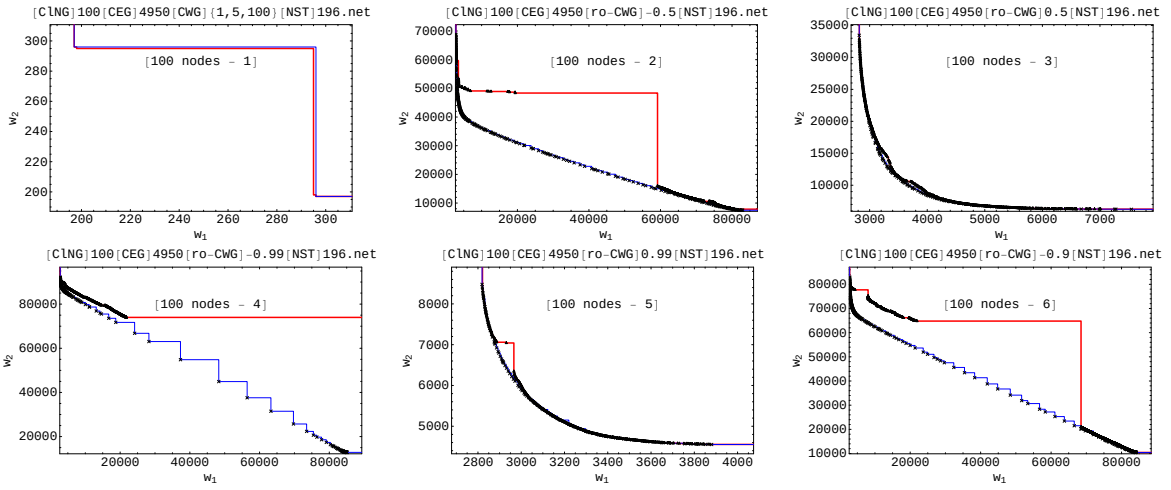

[ClNG]20[k-CEG]50[ro-CWG]-0.99[NST]11.net
[20 nodes - 70]
[ClNG]20[k-CEG]50[ro-CWG]0.99[NST]11.net
[20 nodes - 71]
[ClNG]20[k-CEG]50[ro-CWG]-0.9[NST]11.net
[20 nodes - 72]
[ClNG]20[k-CEG]50[ro-CWG]0.9[NST]11.net
[20 nodes - 73]
[ClNG]20[k-CEG]50[ro-CWG]0.[NST]11.net
[20 nodes - 74]
[ClNG]20[k-CEG]50[RWG][NST]11.net
[20 nodes - 75]
[ClNG]31[VEG]43[CWG]{10,20,100}[NST]8.net
[20 nodes - 76]
[ClNG]31[VEG]43[RWG][NST]8.net
[20 nodes - 77]
[GNG]12[VEG]17[CWG]{10,20,100}[NST]3.net
[20 nodes - 78]
[GNG]12[VEG]17[RWG][NST]3.net
[20 nodes - 79]
[GNG]20[CEG]190[CWG]{10,20,100}[NST]23.net
[20 nodes - 80]
[GNG]20[CEG]190[ro-CWG]-0.1[NST]23.net
[20 nodes - 81]
[GNG]20[CEG]190[ro-CWG]0.1[NST]23.net
[20 nodes - 82]
[GNG]20[CEG]190[ro-CWG]-0.5[NST]23.net
[20 nodes - 83]
[GNG]20[CEG]190[ro-CWG]0.5[NST]23.net
[20 nodes - 84]
[GNG]20[CEG]190[ro-CWG]-0.99[NST]23.net
[20 nodes - 85]
[GNG]20[CEG]190[ro-CWG]0.99[NST]23.net
[20 nodes - 86]
[GNG]20[CEG]190[ro-CWG]-0.9[NST]23.net
[20 nodes - 87]
[GNG]20[CEG]190[ro-CWG]0.9[NST]23.net
[20 nodes - 88]
[GNG]20[CEG]190[ro-CWG]0.[NST]23.net
[20 nodes - 89]
[GNG]20[CEG]190[RWG][NST]23.net
[20 nodes - 90]
w1
w2

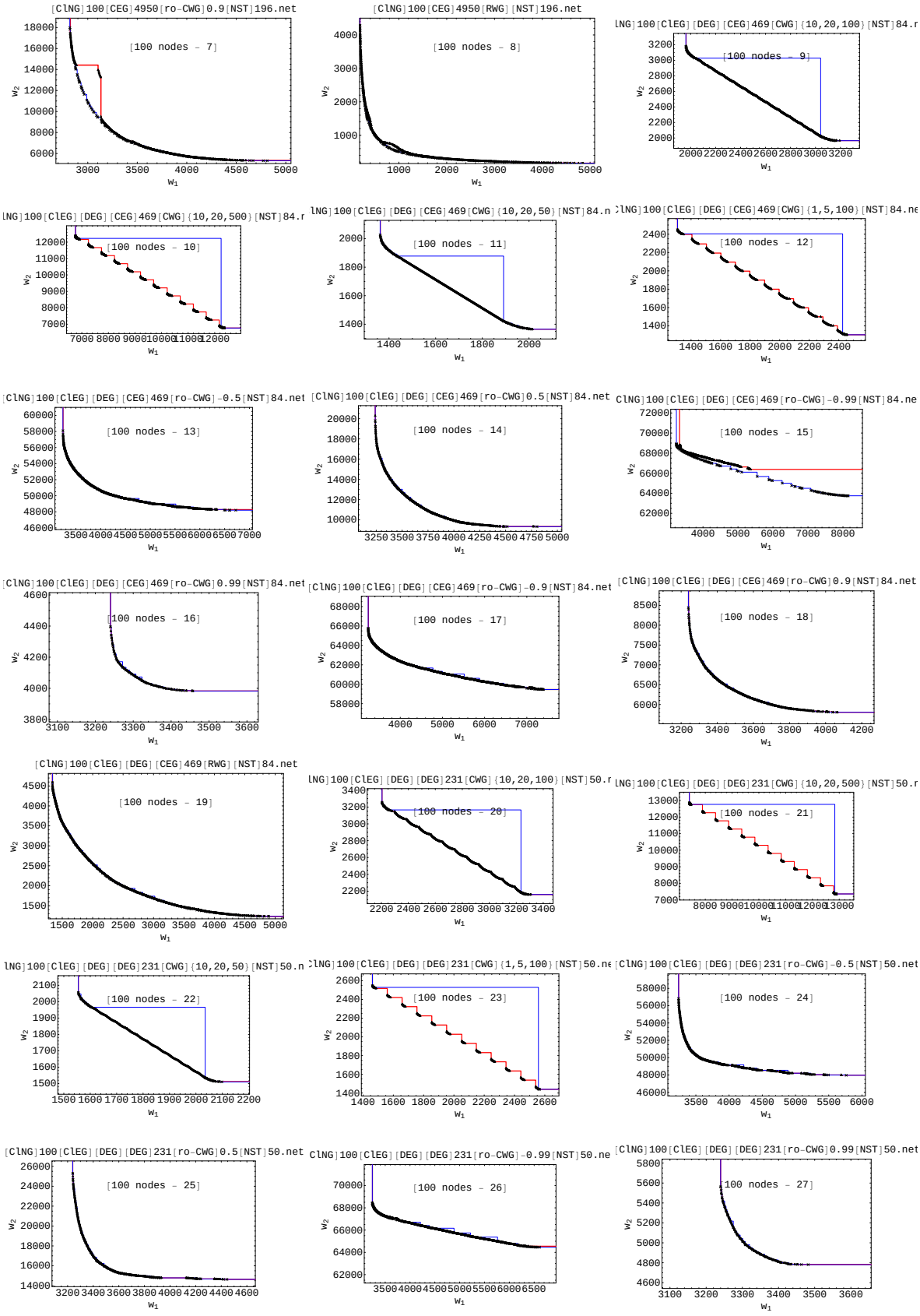

[GNG]20[DEG]43[CWG]{10,20,100}[NST]9.net
[20 nodes - 91]
[GNG]20[DEG]43[RWG][NST]9.net
[20 nodes - 92]
[GNG]20[k-CEG]40[CWG]{10,20,100}[NST]8.net
[20 nodes - 93]
[GNG]20[k-CEG]40[ro-CWG]-0.1[NST]8.net
[20 nodes - 94]
[GNG]20[k-CEG]40[ro-CWG]0.1[NST]8.net
[20 nodes - 95]
[GNG]20[k-CEG]40[ro-CWG]-0.5[NST]8.net
[20 nodes - 96]
[GNG]20[k-CEG]40[ro-CWG]0.5[NST]8.net
[20 nodes - 97]
[GNG]20[k-CEG]40[ro-CWG]-0.99[NST]8.net
[20 nodes - 98]
[GNG]20[k-CEG]40[ro-CWG]0.99[NST]8.net
[20 nodes - 99]
[GNG]20[k-CEG]40[ro-CWG]-0.9[NST]8.net
[20 nodes - 100]
[GNG]20[k-CEG]40[ro-CWG]0.9[NST]8.net
[20 nodes - 101]
[GNG]20[k-CEG]40[ro-CWG]0.[NST]8.net
[20 nodes - 102]
[GNG]25[GEG]40[CWG]{10,20,100}[NST]8.net
[20 nodes - 103]
[GNG]25[GEG]40[CWG]{10,20,500}[NST]8.net
[20 nodes - 104]
[GNG]25[GEG]40[CWG]{10,20,50}[NST]8.net
[20 nodes - 105]
[GNG]25[GEG]40[CWG]{1,5,100}[NST]8.net
[20 nodes - 106]
[GNG]25[GEG]40[RWG][NST]8.net
[20 nodes - 107]
[NNG]20[CEG]190[CWG]{10,20,100}[NST]23.net
[20 nodes - 108]
[NNG]20[CEG]190[ro-CWG]-0.1[NST]23.net
[20 nodes - 109]
[NNG]20[CEG]190[ro-CWG]0.1[NST]23.net
[20 nodes - 110]
[NNG]20[CEG]190[ro-CWG]-0.5[NST]23.net
[20 nodes - 111]
w1
w2

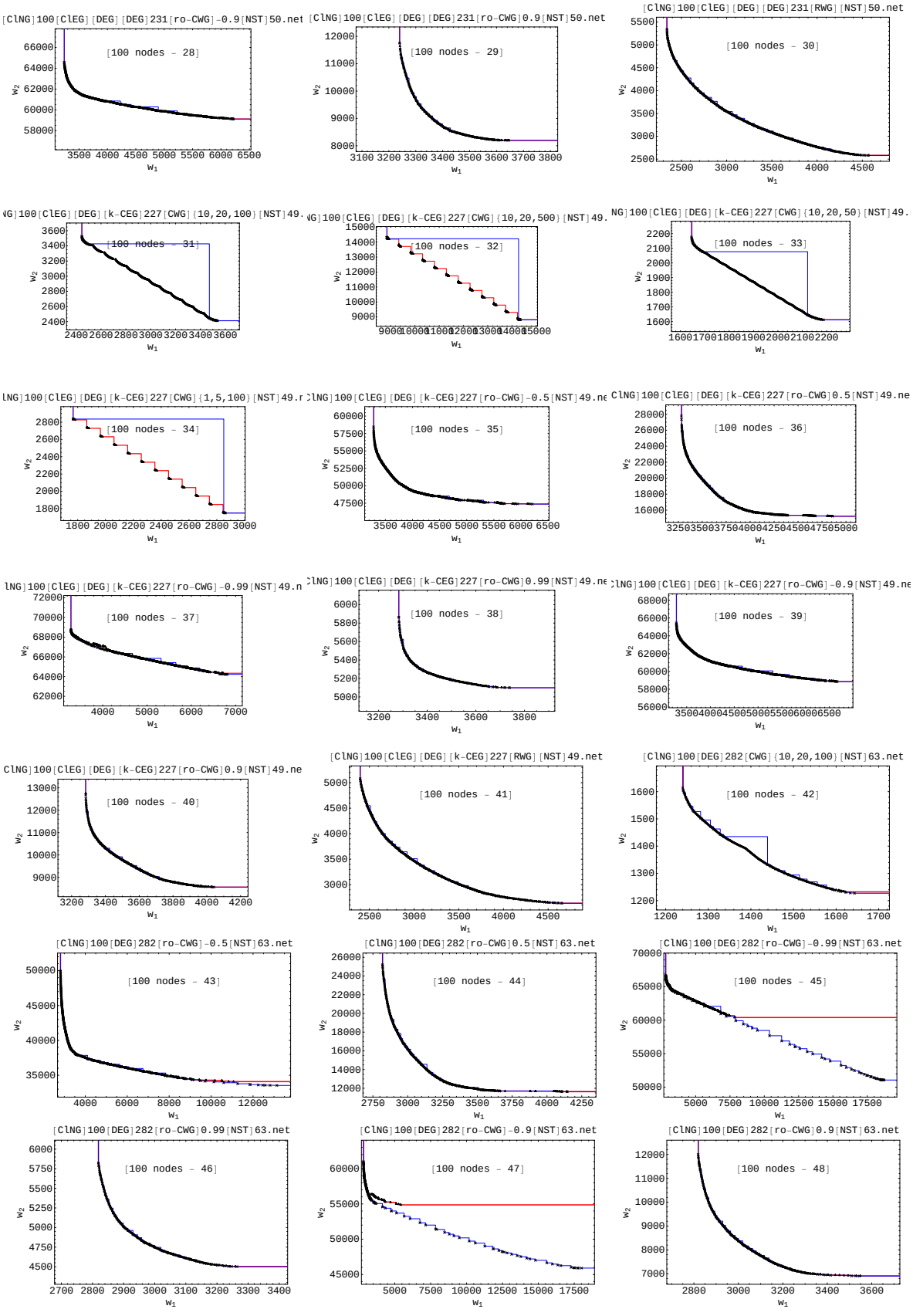

[NNG]20[CEG]190[ro-CWG]0.5[NST]23.net
[20 nodes - 112]
[NNG]20[CEG]190[ro-CWG]-0.99[NST]23.net
[20 nodes - 113]
[NNG]20[CEG]190[ro-CWG]0.99[NST]23.net
[20 nodes - 114]
[NNG]20[CEG]190[ro-CWG]-0.9[NST]23.net
[20 nodes - 115]
[NNG]20[CEG]190[ro-CWG]0.9[NST]23.net
[20 nodes - 116]
[NNG]20[CEG]190[ro-CWG]0.[NST]23.net
[20 nodes - 117]
[NNG]20[CEG]190[RWG][NST]23.net
[20 nodes - 118]
[NNG]20[DEG]50[CWG]{10,20,100}[NST]11.net
[20 nodes - 119]
[NNG]20[DEG]50[ro-CWG]-0.1[NST]11.net
[20 nodes - 120]
[NNG]20[DEG]50[ro-CWG]0.1[NST]11.net
[20 nodes - 121]
[NNG]20[DEG]50[ro-CWG]-0.5[NST]11.net
[20 nodes - 122]
[NNG]20[DEG]50[ro-CWG]0.5[NST]11.net
[20 nodes - 123]
[NNG]20[DEG]50[ro-CWG]-0.99[NST]11.net
[20 nodes - 124]
[NNG]20[DEG]50[ro-CWG]0.99[NST]11.net
[20 nodes - 125]
[NNG]20[DEG]50[ro-CWG]-0.9[NST]11.net
[20 nodes - 126]
[NNG]20[DEG]50[ro-CWG]0.9[NST]11.net
[20 nodes - 127]
[UNG]20[k-CEG]51[ro-CWG]-0.1[NST]11.net
[20 nodes - 128]
[UNG]20[k-CEG]51[ro-CWG]-0.5[NST]11.net
[20 nodes - 129]
[UNG]20[k-CEG]51[ro-CWG]-0.99[NST]11.net
[20 nodes - 130]
[UNG]20[k-CEG]51[ro-CWG]-0.9[NST]11.net
[20 nodes - 131]
[UNG]20[k-CEG]51[RWG][NST]11.net
[20 nodes - 132]
w1
w2

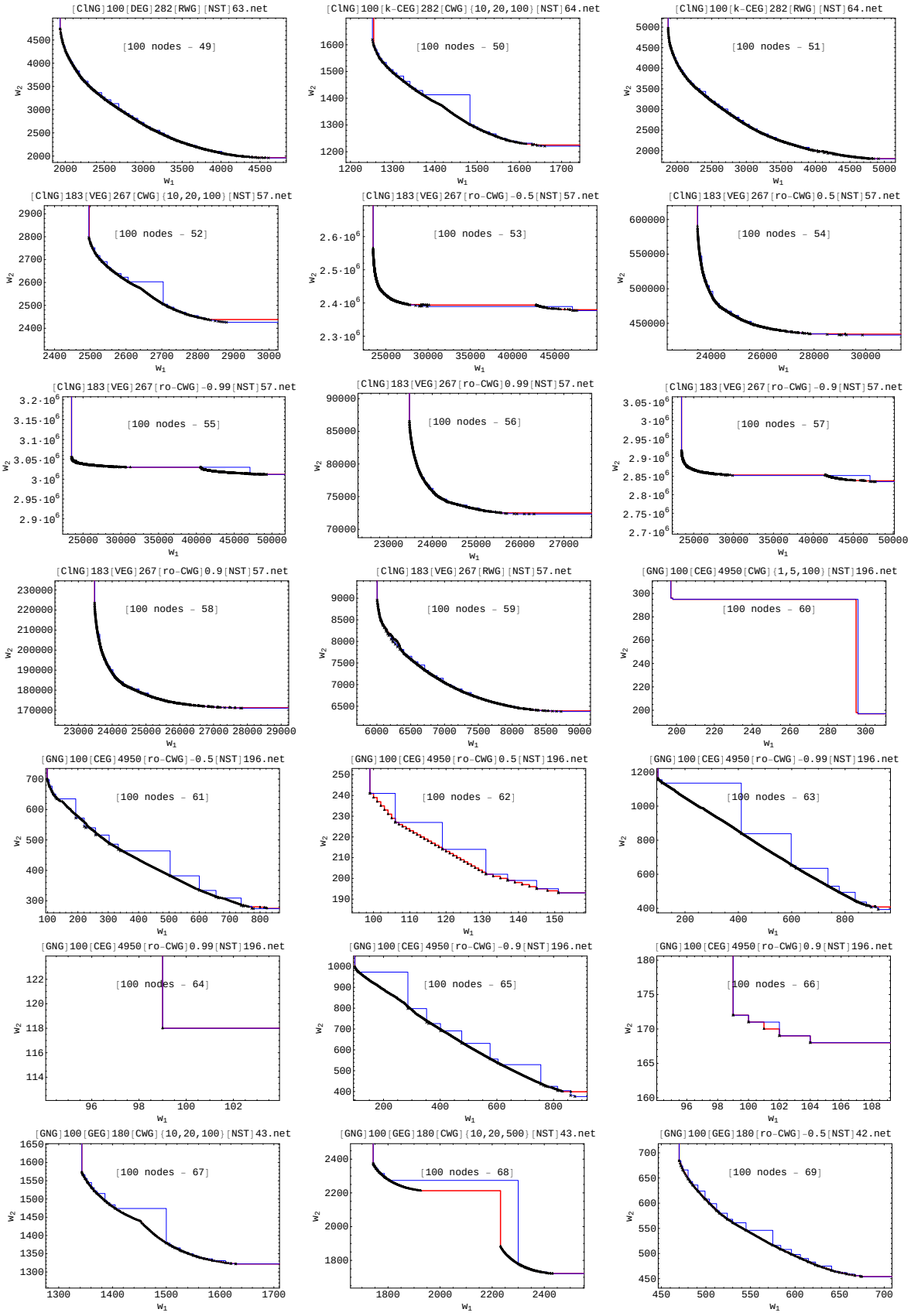


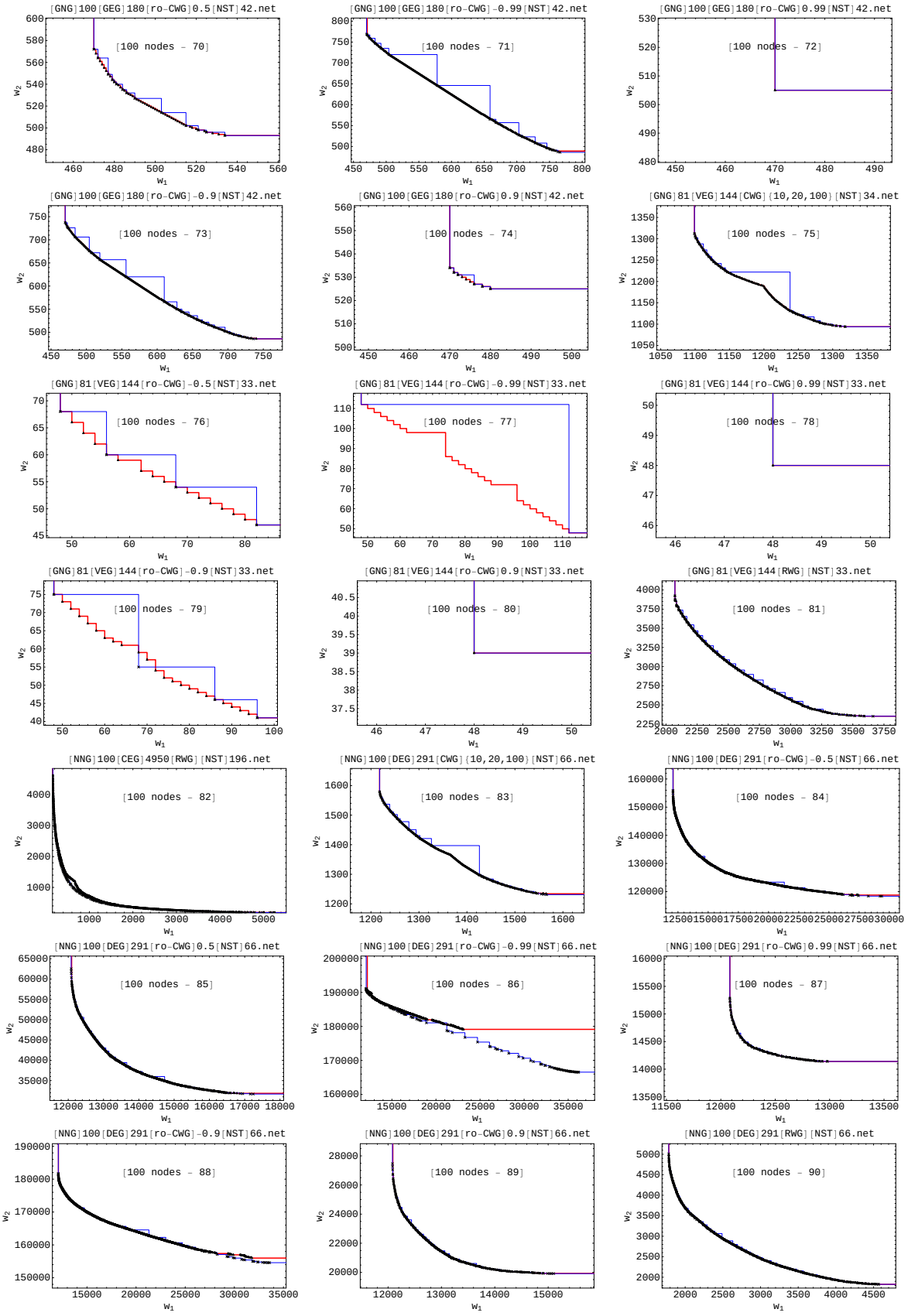
B.3 100 Nodes Networks

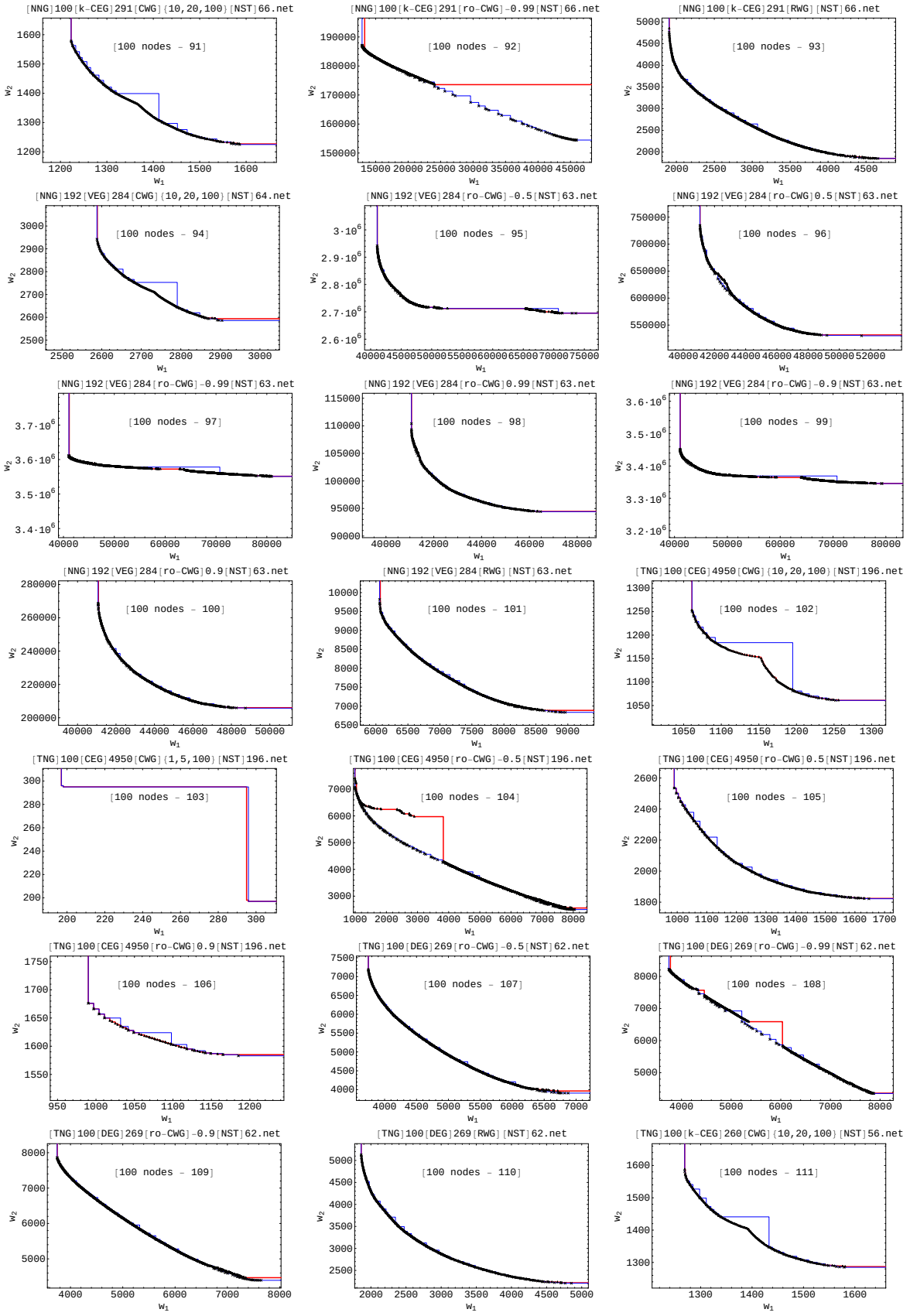


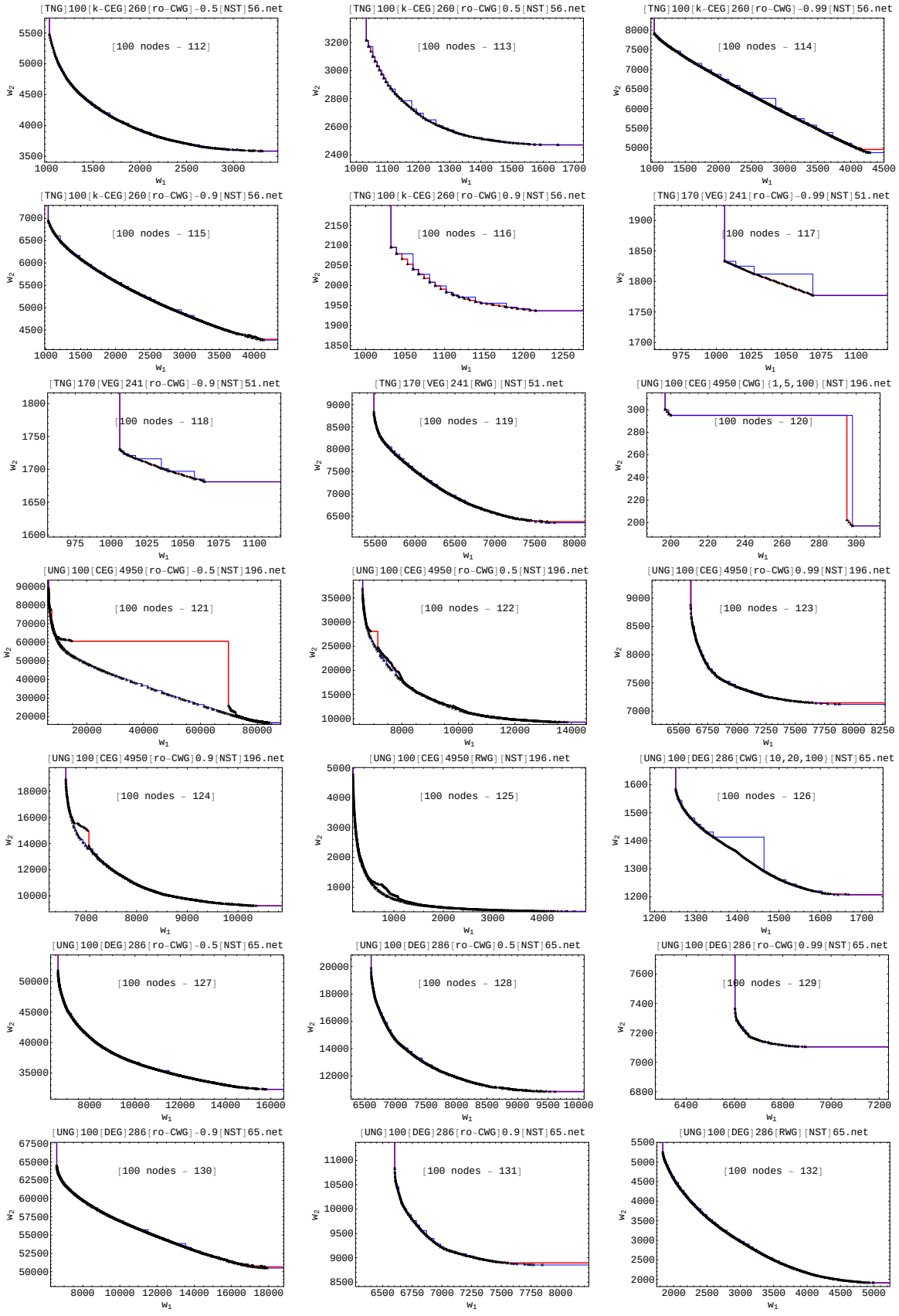

[ClNG]100[CEG]4950[ro-CWG]0.9[NST]196.net
[100 nodes - 7]
[ClNG]100[CEG]4950[RWG][NST]196.net
[100 nodes - 8]
lNG]100[ClEG][DEG][CEG]469[CWG]{10,20,100}[NST]84.r
[100 nodes - 9]
lNG]100[ClEG][DEG][CEG]469[CWG]{10,20,500}[NST]84.r
[100 nodes - 10]
lNG]100[ClEG][DEG][CEG]469[CWG]{10,20,50}[NST]84.n
[100 nodes - 11]
lNG]100[ClEG][DEG][CEG]469[CWG]{1,5,100}[NST]84.ne
[100 nodes - 12]
[ClNG]100[ClEG][DEG][CEG]469[ro-CWG]-0.5[NST]84.net
[100 nodes - 13]
[ClNG]100[ClEG][DEG][CEG]469[ro-CWG]0.5[NST]84.net
[100 nodes - 14]
[ClNG]100[ClEG][DEG][CEG]469[ro-CWG]0.99[NST]84.ne
[100 nodes - 15]
[ClNG]100[ClEG][DEG][CEG]469[ro-CWG]0.99[NST]84.net
[100 nodes - 16]
[ClNG]100[ClEG][DEG][CEG]469[ro-CWG]-0.9[NST]84.net
[100 nodes - 17]
[ClNG]100[ClEG][DEG][CEG]469[ro-CWG]0.9[NST]84.net
[100 nodes - 18]
[ClNG]100[ClEG][DEG][CEG]469[RWG][NST]84.net
[100 nodes - 19]
lNG]100[ClEG][DEG][DEG]231[CWG]{10,20,100}[NST]50.r
[100 nodes - 20]
lNG]100[ClEG][DEG][DEG]231[CWG]{10,20,500}[NST]50.r
[100 nodes - 21]
lNG]100[ClEG][DEG][DEG]231[CWG]{10,20,50}[NST]50.n
[100 nodes - 22]
lNG]100[ClEG][DEG][DEG]231[CWG]{1,5,100}[NST]50.ne
[100 nodes - 23]
[ClNG]100[ClEG][DEG][DEG]231[ro-CWG]-0.5[NST]50.net
[100 nodes - 24]
[ClNG]100[ClEG][DEG][DEG]231[ro-CWG]0.5[NST]50.net
[100 nodes - 25]
lNG]100[ClEG][DEG][DEG]231[ro-CWG]-0.99[NST]50.ne
[100 nodes - 26]
[ClNG]100[ClEG][DEG][DEG]231[ro-CWG]0.99[NST]50.net
[100 nodes - 27]

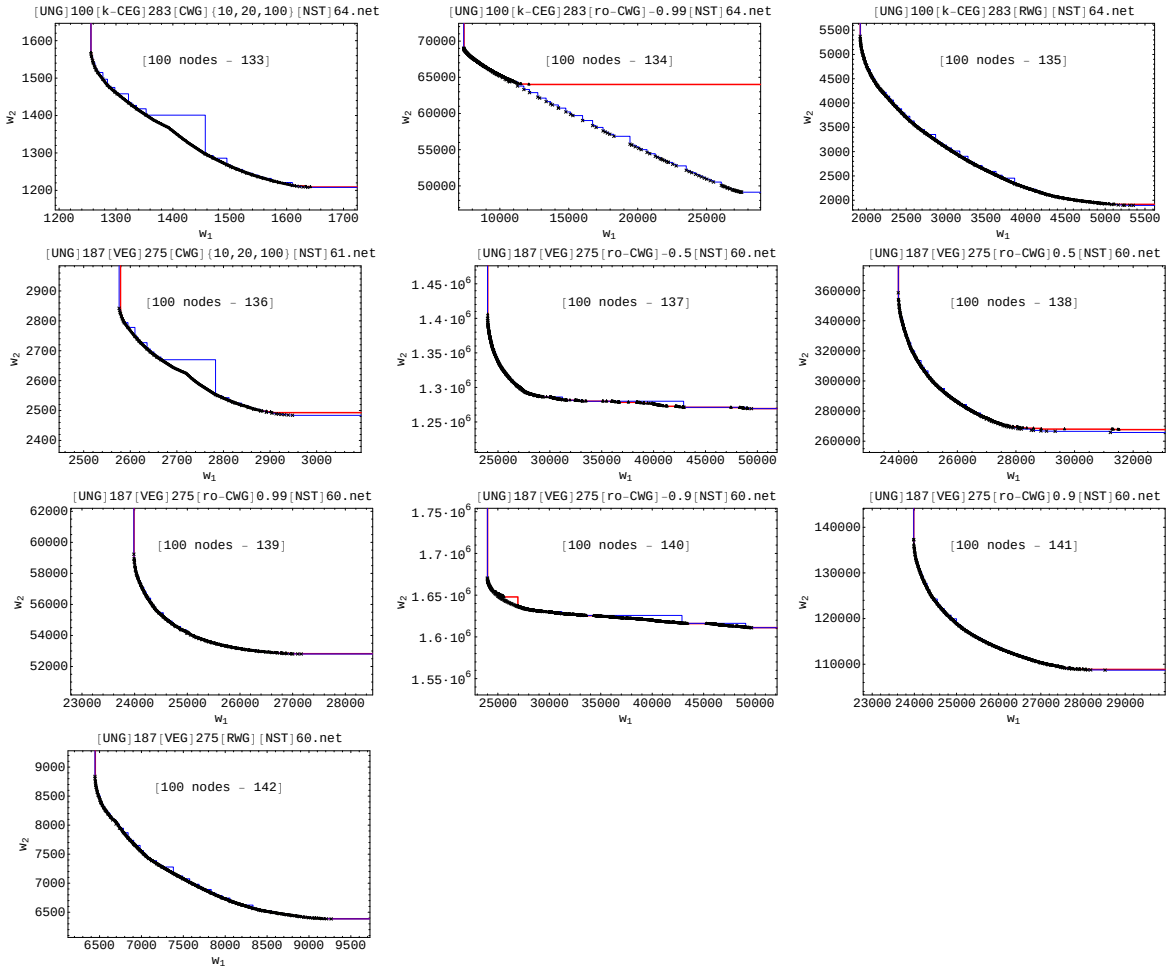

[ClNG]100[ClEG][DEG][DEG]231[ro-CWG]-0.9[NST]50.net
[100 nodes - 28]
[ClNG]100[ClEG][DEG][DEG]231[ro-CWG]0.9[NST]50.net
[100 nodes - 29]
[ClNG]100[ClEG][DEG][DEG]231[RWG][NST]50.net
[100 nodes - 30]
[100 nodes - 31]
[100 nodes - 32]
[100 nodes - 33]
[100 nodes - 34]
[100 nodes - 35]
[100 nodes - 36]
[100 nodes - 37]
[100 nodes - 38]
[100 nodes - 39]
[ClNG]100[ClEG][DEG][k-CEG]227[RWG][NST]49.net
[ClNG]100[DEG]282[CWG]{10,20,100}[NST]63.net
[100 nodes - 40]
[100 nodes - 41]
[100 nodes - 42]
[ClNG]100[DEG]282[ro-CWG]-0.5[NST]63.net
[ClNG]100[DEG]282[ro-CWG]0.5[NST]63.net
[ClNG]100[DEG]282[ro-CWG]-0.99[NST]63.net
[100 nodes - 43]
[100 nodes - 44]
[100 nodes - 45]
[ClNG]100[DEG]282[ro-CWG]0.99[NST]63.net
[ClNG]100[DEG]282[ro-CWG]-0.9[NST]63.net
[ClNG]100[DEG]282[ro-CWG]0.9[NST]63.net
[100 nodes - 46]
[100 nodes - 47]
[100 nodes - 48]


[ClNG]100[DEG]282[RWG][NST]63.net
[100 nodes - 49]
[ClNG]100[k-CEG]282[CWG]{10,20,100}[NST]64.net
[100 nodes - 50]
[ClNG]100[k-CEG]282[RWG][NST]64.net
[100 nodes - 51]
[ClNG]183[VEG]267[CWG]{10,20,100}[NST]57.net
[100 nodes - 52]
[ClNG]183[VEG]267[ro-CWG]-0.5[NST]57.net
[100 nodes - 53]
[ClNG]183[VEG]267[ro-CWG]0.5[NST]57.net
[100 nodes - 54]
[ClNG]183[VEG]267[ro-CWG]-0.99[NST]57.net
[100 nodes - 55]
[ClNG]183[VEG]267[ro-CWG]0.99[NST]57.net
[100 nodes - 56]
[ClNG]183[VEG]267[ro-CWG]-0.9[NST]57.net
[100 nodes - 57]
[ClNG]183[VEG]267[ro-CWG]0.9[NST]57.net
[100 nodes - 58]
[ClNG]183[VEG]267[RWG][NST]57.net
[100 nodes - 59]
[GNG]100[CEG]4950[CWG]{1,5,100}[NST]196.net
[100 nodes - 60]
[GNG]100[CEG]4950[ro-CWG]-0.5[NST]196.net
[100 nodes - 61]
[GNG]100[CEG]4950[ro-CWG]0.5[NST]196.net
[100 nodes - 62]
[GNG]100[CEG]4950[ro-CWG]-0.99[NST]196.net
[100 nodes - 63]
[GNG]100[CEG]4950[ro-CWG]0.99[NST]196.net
[100 nodes - 64]
[GNG]100[CEG]4950[ro-CWG]-0.9[NST]196.net
[100 nodes - 65]
[GNG]100[CEG]4950[ro-CWG]0.9[NST]196.net
[100 nodes - 66]
[GNG]100[GEG]180[CWG]{10,20,100}[NST]43.net
[100 nodes - 67]
[GNG]100[GEG]180[CWG]{10,20,500}[NST]43.net
[100 nodes - 68]
[GNG]100[GEG]180[ro-CWG]-0.5[NST]42.net
[100 nodes - 69]
w1
w2


[GNG]100[GEG]180[ro-CWG]0.5[NST]42.net
[100 nodes - 70]
[GNG]100[GEG]180[ro-CWG]-0.99[NST]42.net
[100 nodes - 71]
[GNG]100[GEG]180[ro-CWG]0.99[NST]42.net
[100 nodes - 72]
[GNG]100[GEG]180[ro-CWG]-0.9[NST]42.net
[100 nodes - 73]
[GNG]100[GEG]180[ro-CWG]0.9[NST]42.net
[100 nodes - 74]
[GNG]81[VEG]144[CWG]{10,20,100}[NST]34.net
[100 nodes - 75]
[GNG]81[VEG]144[ro-CWG]-0.5[NST]33.net
[100 nodes - 76]
[GNG]81[VEG]144[ro-CWG]-0.99[NST]33.net
[100 nodes - 77]
[GNG]81[VEG]144[ro-CWG]0.99[NST]33.net
[100 nodes - 78]
[GNG]81[VEG]144[ro-CWG]-0.9[NST]33.net
[100 nodes - 79]
[GNG]81[VEG]144[ro-CWG]0.9[NST]33.net
[100 nodes - 80]
[GNG]81[VEG]144[RWG][NST]33.net
[100 nodes - 81]
[NNG]100[CEG]4950[RWG][NST]196.net
[100 nodes - 82]
[NNG]100[DEG]291[CWG]{10,20,100}[NST]66.net
[100 nodes - 83]
[NNG]100[DEG]291[ro-CWG]-0.5[NST]66.net
[100 nodes - 84]
[NNG]100[DEG]291[ro-CWG]0.5[NST]66.net
[100 nodes - 85]
[NNG]100[DEG]291[ro-CWG]-0.99[NST]66.net
[100 nodes - 86]
[NNG]100[DEG]291[ro-CWG]0.99[NST]66.net
[100 nodes - 87]
[NNG]100[DEG]291[ro-CWG]-0.9[NST]66.net
[100 nodes - 88]
[NNG]100[DEG]291[ro-CWG]0.9[NST]66.net
[100 nodes - 89]
[NNG]100[DEG]291[RWG][NST]66.net
[100 nodes - 90]
w1
w2


[NNG]100[k-CEG]291[CWG]{10,20,100}[NST]66.net
[100 nodes - 91]
[NNG]100[k-CEG]291[ro-CWG]-0.99[NST]66.net
[100 nodes - 92]
[NNG]100[k-CEG]291[RWG][NST]66.net
[100 nodes - 93]
[NNG]192[VEG]284[CWG]{10,20,100}[NST]64.net
[100 nodes - 94]
[NNG]192[VEG]284[ro-CWG]-0.5[NST]63.net
[100 nodes - 95]
[NNG]192[VEG]284[ro-CWG]0.5[NST]63.net
[100 nodes - 96]
[NNG]192[VEG]284[ro-CWG]-0.99[NST]63.net
[100 nodes - 97]
[NNG]192[VEG]284[ro-CWG]0.99[NST]63.net
[100 nodes - 98]
[NNG]192[VEG]284[ro-CWG]-0.9[NST]63.net
[100 nodes - 99]
[NNG]192[VEG]284[ro-CWG]0.9[NST]63.net
[100 nodes - 100]
[NNG]192[VEG]284[RWG][NST]63.net
[100 nodes - 101]
[TNG]100[CEG]4950[CWG]{10,20,100}[NST]196.net
[100 nodes - 102]
[TNG]100[CEG]4950[CWG]{1,5,100}[NST]196.net
[100 nodes - 103]
[TNG]100[CEG]4950[ro-CWG]-0.5[NST]196.net
[100 nodes - 104]
[TNG]100[CEG]4950[ro-CWG]0.5[NST]196.net
[100 nodes - 105]
[TNG]100[CEG]4950[ro-CWG]0.9[NST]196.net
[100 nodes - 106]
[TNG]100[DEG]269[ro-CWG]-0.5[NST]62.net
[100 nodes - 107]
[TNG]100[DEG]269[ro-CWG]-0.99[NST]62.net
[100 nodes - 108]
[TNG]100[DEG]269[ro-CWG]-0.9[NST]62.net
[100 nodes - 109]
[TNG]100[DEG]269[RWG][NST]62.net
[100 nodes - 110]
[TNG]100[k-CEG]260[CWG]{10,20,100}[NST]56.net
[100 nodes - 111]
w1
w2


[TNG]100[k-CEG]260[ro-CWG]-0.5[NST]56.net
[100 nodes - 112]
[TNG]100[k-CEG]260[ro-CWG]0.5[NST]56.net
[100 nodes - 113]
[TNG]100[k-CEG]260[ro-CWG]-0.99[NST]56.net
[100 nodes - 114]
[TNG]100[k-CEG]260[ro-CWG]-0.9[NST]56.net
[100 nodes - 115]
[TNG]100[k-CEG]260[ro-CWG]0.9[NST]56.net
[100 nodes - 116]
[TNG]170[VEG]241[ro-CWG]-0.99[NST]51.net
[100 nodes - 117]
[TNG]170[VEG]241[ro-CWG]-0.9[NST]51.net
[100 nodes - 118]
[TNG]170[VEG]241[RWG][NST]51.net
[100 nodes - 119]
[UNG]100[CEG]4950[CWG]{1,5,100}[NST]196.net
[100 nodes - 120]
[UNG]100[CEG]4950[ro-CWG]-0.5[NST]196.net
[100 nodes - 121]
[UNG]100[CEG]4950[ro-CWG]0.5[NST]196.net
[100 nodes - 122]
[UNG]100[CEG]4950[ro-CWG]0.99[NST]196.net
[100 nodes - 123]
[UNG]100[CEG]4950[ro-CWG]0.9[NST]196.net
[100 nodes - 124]
[UNG]100[CEG]4950[RWG][NST]196.net
[100 nodes - 125]
[UNG]100[DEG]286[CWG]{10,20,100}[NST]65.net
[100 nodes - 126]
[UNG]100[DEG]286[ro-CWG]-0.5[NST]65.net
[100 nodes - 127]
[UNG]100[DEG]286[ro-CWG]0.5[NST]65.net
[100 nodes - 128]
[UNG]100[DEG]286[ro-CWG]0.99[NST]65.net
[100 nodes - 129]
[UNG]100[DEG]286[ro-CWG]-0.9[NST]65.net
[100 nodes - 130]
[UNG]100[DEG]286[ro-CWG]0.9[NST]65.net
[100 nodes - 131]
[UNG]100[DEG]286[RWG][NST]65.net
[100 nodes - 132]
w1
w2


[UNG]100[k-CEG]283[CWG]{10,20,100}[NST]64.net
[100 nodes - 133]
[UNG]100[k-CEG]283[ro-CWG]-0.99[NST]64.net
[100 nodes - 134]
[UNG]100[k-CEG]283[RWG][NST]64.net
[100 nodes - 135]
[UNG]187[VEG]275[CWG]{10,20,100}[NST]61.net
[100 nodes - 136]
[UNG]187[VEG]275[ro-CWG]-0.5[NST]60.net
[100 nodes - 137]
[UNG]187[VEG]275[ro-CWG]0.5[NST]60.net
[100 nodes - 138]
[UNG]187[VEG]275[ro-CWG]0.99[NST]60.net
[100 nodes - 139]
[UNG]187[VEG]275[ro-CWG]-0.9[NST]60.net
[100 nodes - 140]
[UNG]187[VEG]275[ro-CWG]0.9[NST]60.net
[100 nodes - 141]
[UNG]187[VEG]275[RWG][NST]60.net
[100 nodes - 142]
w1
w2

C

Code

This chapter contains parts of the developed codes in $C++$ for the Multiple Objective Minimum Spanning Trees problem.

C.1 Global classes

C.1.1 Constants

```
/***************************************************************************
                          consts.cpp  -  description

    date                 : Dec 2006
    copyright            : (C) 2006 by Pedro Cardoso
    email                : pcardoso@ualg.pt
 ***************************************************************************
Resume:
  constants definitions
 ***************************************************************************/

#pragma once
#if !defined(CONSTANTES)
#define CONSTANTES

#include <limits.h>
#include <stdio.h>
#include <math.h>

typedef double TREAL;
typedef int TCUSTOS;
typedef int INT;

const int MAXN = 500;
const int NUMERO_MAXIMO_CUSTOS = 2;
const int doistom = 4;
const TREAL INFINITO = (TREAL)1e+37;
const TREAL INFIMO = (TREAL)1e-37;
const TREAL cd = 0.0000005;

//alpha's and beta's variation interval
const TREAL minpar = 0.0;
const TREAL maxpar = 3.0;
```

```
34 static TREAL passopar = 0.01;
35
36 //method choice
37 const int dante_0 = 0;
38 const int dante_ALL = 3;
39 const int edante = 1;
40 const int acoqt = 2;
41
42 //0-no output 1:show cycles 2:(1+) write pheromone matrices and
43 // +/- in the poutput according to entraces/exits in the quad-tree
44 static int _debug_ = 0;
45
46 // Types definitions
47 typedef char STRING[100];
48 typedef TREAL ARRAY_PARAMS[NUMERO_MAXIMO_CUSTOS];
49 typedef TCUSTOS ARRAY_CUSTOS[NUMERO_MAXIMO_CUSTOS];
50 typedef TREAL ARRAY_FEROMONAS[NUMERO_MAXIMO_CUSTOS];
51 typedef TREAL ARRAY_PRODUTOS[NUMERO_MAXIMO_CUSTOS];
52 typedef INT ARRAY_INTEIROS[MAXN];
53 typedef INT MATRIZ_INTEIROS[MAXN][MAXN];
54
55 //Structures difinitions
56 typedef struct LISTA_INTEIROS{
57         int no;
58         struct LISTA_INTEIROS *prox;
59 } LISTA_INT;
60
61 struct LISTA{
62   int no;
63   struct LISTA *prox;
64   };
65
66 struct LISTA_ADJACENCIA{
67     int no;
68         struct LISTA_ADJACENCIA *prox;
69 };
70
71 typedef LISTA_ADJACENCIA *ADJACENCY_MATRIX[MAXN];
72
73 #endif
```

C.1.2 Network Class

```
1  /***************************************************************************
2                           rede.h  -  description
3
4      date                 : Dec 2006
5      copyright            : (C) 2006 by Pedro Cardoso
6      email                : pcardoso@ualg.pt
7   ***************************************************************************
8  Resume:
9    Class Network
10  ***************************************************************************/
11
12 #ifndef REDE_H
13 #define REDE_H
14
15 #include <stdlib.h>
16 #include <math.h>
17 #include <stdio.h>
18 #include "consts.cpp"
19
20 class REDE {
21 public:
22   typedef struct ENTRADA{
23       ARRAY_CUSTOS  VctCustos;
24       ARRAY_FEROMONAS feromon;
25       ARRAY_FEROMONAS delta;
26       TREAL     ProdutoDosProdutos;
27           int       prox;
28   };
29   typedef struct ENTRADA  *ADJACENCY_MATRIX[MAXN][MAXN];
30
31   int NN; //numero de nos
32   int NA; //nº de arestas
33   int NC;//nº de custos
34   int Nos[MAXN][4];  //coordenadas dos nos
35   ADJACENCY_MATRIX  MzAdj;      //matriz de adjacencia
36   int prim_MzAdj[MAXN];    //serve para saber onde começa a lista de adjacencia
37   ARRAY_PARAMS *params_cst;
38   ARRAY_PARAMS *params_fer;
```

```
39    ARRAY_CUSTOS maximos;
40    TREAL rho_;             //factor de evaporacao
41    TREAL SomaDoProdutoDosProdutos[MAXN];
42
43  /*****************************************************/
44  //DESCONSTUCTOR
45  ~REDE(){
46    printf("Deleting Network...");
47      for(int i=0; i<NN; i++){
48      printf(".");
49      for(int j=0; j<NN; j++)
50        if(MzAdj[i][j]!=NULL)
51          delete MzAdj[i][j];
52    }
53    printf("deleted.\n");
54  }
55
56  /*****************************************************/
57  //CONSTUCTOR
58  REDE(STRING nomerede, ARRAY_PARAMS *params_cst_in,
59     ARRAY_PARAMS *params_fer_in,TREAL rho_in){
60
61    int i,t,s,cc;
62    FILE *fnet;
63    ARRAY_CUSTOS vc_aux;
64    char tiporede[256];
65
66      params_cst=params_cst_in;
67      params_fer=params_fer_in;
68      rho_=rho_in;
69
70      //open file
71      if((fnet=fopen(nomerede,"r"))==NULL){
72        printf("erro na abertura do ficheiro com a rede!\n");
73        exit(0);
74      }
75
76      // read network
77      printf("Vai ler a rede: %s...\n",nomerede);
78      //read number of nodes
79      fscanf(fnet,"%d",&NN);
80      //read number of edges
81      fscanf(fnet,"%d",&NA);
82      //read number of weights
83      fscanf(fnet,"%d",&NC);
84
85      //read the nodes
86      int aux;
87      for(i=0;i<NN;i++){
88        fscanf(fnet,"%d",&aux);
89        Nos[i][0]=aux;
90        fscanf(fnet,"%d",&aux);
91        Nos[i][1]=aux;
92
93        for(int j=0;j<NN;j++)
94          MzAdj[i][j]=NULL;
95        prim_MzAdj[i]=-1;
96      }
97
98      //le the edges+weights
99      for(cc=0;cc<NC;cc++){
100       maximos[cc]=0;
101     }
102     for(i=0;i<NA;i++){
103       fscanf(fnet,"%d",&s);
104       fscanf(fnet,"%d",&t);
105       for(cc=0; cc<NC ; cc++){
106         fscanf(fnet,"%d",&aux);
107         vc_aux[cc]=aux;
108         if(aux>maximos[cc])
109           maximos[cc]=aux;
110       }
111       addAresta(s,t,&vc_aux);
112       addAresta(t,s,&vc_aux);
113     }
114     printf("nos= %d \t arestas=%d \t custos = %d \t rede ok.\n",NN,NA,NC);
115     fclose(fnet);
```

```
116 }
117
118 /*************************************************/
119 //adds an edge
120 int addAresta(int x,int y, ARRAY_CUSTOS *vc){
121 int procura;
122 int cc;
123   if(x==y){
124     printf("Pretendia acrescentar um laco!!!");
125     return 0;
126   }
127   if(x>=0 && x<NN && y>=0 && y<NN){
128       //new node
129       MzAdj[x][y] = new ENTRADA;
130       MzAdj[x][y]->ProdutoDosProdutos=1.0;
131       for(cc=0;cc<NC;cc++){
132         MzAdj[x][y]->VctCustos[cc]  = (*vc)[cc];
133         MzAdj[x][y]->feromon[cc]  = 1.0;
134         MzAdj[x][y]->delta[cc]    = 0.0;
135         MzAdj[x][y]->ProdutoDosProdutos *=1.0/pow(INFIMO+(TREAL)((*vc)[cc]),(*
              params_cst)[cc]); // ao ler a rede as feromnas estão a zero
136       }
137       if(prim_MzAdj[x]==-1){//empty list
138         MzAdj[x][y]->prox=-1;
139         prim_MzAdj[x]=y;
140       }else{
141         if(prim_MzAdj[x]>y){//index > y
142                   MzAdj[x][y]->prox=prim_MzAdj[x];
143           prim_MzAdj[x]=y;
144         }else{//index <y
145                  procura=prim_MzAdj[x];
146           while((MzAdj[x][procura]->prox<y)&&(MzAdj[x][procura]->prox!=-1))
147             procura=MzAdj[x][procura]->prox;
148
149           MzAdj[x][y]->prox=MzAdj[x][procura]->prox;
150           MzAdj[x][procura]->prox=y;
151         }
152       }
153   }else{
154     printf("\nErro ao adicionar aresta entre nos nao definidos: %d - %d\n",x,y);
155     getchar();
156     exit(0);
157   }
158   return 1;
159 }
160
161 /*************************************************/
162 //for debuging
163 void Print(){
164 int i;
165     for(i=0;i<NN;i++){
166       printf("\n********%d*********",i);
167       for(int j=0;j<NN;j++){
168         if(MzAdj[i][j]!=NULL)
169           printf("%d ",j);
170       }
171       printf("\n");
172     }
173 }
174
175 /*************************************************/
176 //exists edge?
177 //input:  2 nodes - s,t
178 //output(return):
179 //      1 if edge defined between s and t exists
180 //      0 o.w.
181 int Aresta(int s,int t){
182   if(MzAdj[s][t])
183     return 1;
184   return 0;
185 }
186
187 /*************************************************/
188 //input: the pointers for the new Arrays of parameters
189 //output(return): parameters array updated
190 void AlteraParametros(ARRAY_PARAMS *params_cst_in,
```

```
191             ARRAY_PARAMS *params_fer_in){
192
193   params_cst=params_cst_in;
194   params_fer=params_fer_in;
195   ActualizaProdutosDosProdutos();
196 };
197
198
199
200 /*****************************************************/
201 //input:
202 //output(return): matrix of products updated
203 void ActualizaProdutosDosProdutos(){
204 int aux;
205
206   for(int i=0;i<NN;i++){
207     SomaDoProdutoDosProdutos[i]=0.0;
208     aux=prim_MzAdj[i];
209     while(aux!=-1){
210       MzAdj[i][aux]->ProdutoDosProdutos=1.0;
211       for(int cc=0;cc<NC; cc++){
212         MzAdj[i][aux]->ProdutoDosProdutos*=
213           pow(1.0/(INFIMO+(TREAL)(MzAdj[i][aux]->VctCustos[cc])),(TREAL)(*params_cst)[
                cc])
214           *
215           pow((TREAL)(MzAdj[i][aux]->feromon[cc]),(TREAL)((*params_fer)[cc]));
216       }
217       SomaDoProdutoDosProdutos[i]+=MzAdj[i][aux]->ProdutoDosProdutos;
218       aux=MzAdj[i][aux]->prox;
219     }
220   }
221 }
222
223 /*****************************************************/
224 //input:
225 //output(return): update the pheromone matrix
226 //does fer=rho *fer+ delta
227 void ActualizaFeromonas(){
228   ActualizaFeromonas(params_fer,params_cst);
229 }
230
231 /*****************************************************/
232 //input:
233 //output(return): matrix of products updated
234 void ActualizaFeromonas(ARRAY_PARAMS *params_fer_in,
235             ARRAY_PARAMS *params_cst_in){
236   int aux;
237
238   params_cst=params_cst_in;
239   params_fer=params_fer_in;
240   for(int no=0; no<NN ; no++){
241     aux=prim_MzAdj[no];
242     while(aux!=-1){
243       for(int cc=0;cc<NC;cc++){
244         MzAdj[no][aux]->feromon[cc]=
245           rho_ * MzAdj[no][aux]->feromon[cc] + MzAdj[no][aux]->delta[cc]+0.0001;
246         MzAdj[no][aux]->delta[cc] = 0.0;
247       }
248       aux=MzAdj[no][aux]->prox;
249     }
250   }
251   ActualizaProdutosDosProdutos();
252 }
253
254 /*****************************************************/
255 //does fer=Sqrt(fer,n)+s
256 void NormalizaFeromonasSQRTeSOMA(TREAL n,TREAL s){
257 int aux;
258 TREAL expo = 1.0/n;
259   for(int no=0; no<NN ; no++){
260     aux=prim_MzAdj[no];
261     while(aux!=-1){
262       for(int cc=0;cc<NC;cc++){
263         MzAdj[no][aux]->feromon[cc]= pow(MzAdj[no][aux]->feromon[cc],expo)+s;
264       }
265       aux=MzAdj[no][aux]->prox;
266     }
```

```
267   }
268   ActualizaProdutosDosProdutos();
269
270 }
271
272 /*****************************************************/
273 //does fer=log(fer)+s
274 void NormalizaFeromonasLOGeSOMA(TREAL s){
275 int aux;
276   for(int no=0; no<NN ; no++){
277     aux=prim_MzAdj[no];
278     while(aux!=-1){
279       for(int cc=0;cc<NC;cc++){
280         MzAdj[no][aux]->feromon[cc]= log(MzAdj[no][aux]->feromon[cc]+1)+s;
281       }
282       aux=MzAdj[no][aux]->prox;
283     }
284   }
285   ActualizaProdutosDosProdutos();
286
287 }
288
289 /*****************************************************/
290 //does a normalization between a and b of the pheromones
291 void NormalizaFeromonas(TREAL a, TREAL b){
292 int aux;
293 TREAL maximo,minimo;
294 int i;
295   for(int cc=0; cc< NC; cc++){
296     //determines max/min
297     minimo=MzAdj[0][prim_MzAdj[0]]->feromon[cc];
298     maximo=minimo;
299     for(i=0;i<NN;i++){
300       aux=prim_MzAdj[i];
301       while(aux!=-1){
302         if(minimo>MzAdj[i][aux]->feromon[cc])
303             minimo=MzAdj[i][aux]->feromon[cc];
304         if(maximo<MzAdj[i][aux]->feromon[cc])
305             maximo=MzAdj[i][aux]->feromon[cc];
306         aux=MzAdj[i][aux]->prox;
307       }
308
309     }
310     //normalize
311     if(minimo!=maximo)
312       for(i=0;i<NN;i++){
313         aux=prim_MzAdj[i];
314         while(aux!=-1){
315           MzAdj[i][aux]->feromon[cc]=
316             a + ((a - b)*(minimo - MzAdj[i][aux]->feromon[cc]))/(maximo - minimo);
317           aux=MzAdj[i][aux]->prox;
318         }
319       }
320   }
321   ActualizaProdutosDosProdutos();
322 }
323
324 /*****************************************************/
325 //does fer=tanh(fer)
326 void NormalizaFeromonasTANH(){
327 int aux;
328   for(int no=0; no<NN ; no++){
329     aux=prim_MzAdj[no];
330     while(aux!=-1){
331       for(int cc=0;cc<NC;cc++){
332         MzAdj[no][aux]->feromon[cc]=
333           1+tanh( MzAdj[no][aux]->feromon[cc]/10);
334       }
335       aux=MzAdj[no][aux]->prox;
336     }
337   }
338   ActualizaProdutosDosProdutos();
339 }
340
341 /*****************************************************/
342 //does fer=arctan(fer...)...
343 void NormalizaFeromonasARCTAN(){
```

```
344 int aux;
345   for(int no=0; no<NN ; no++){
346     aux=prim_MzAdj[no];
347     while(aux!=-1){
348       for(int cc=0;cc<NC;cc++){
349         MzAdj[no][aux]->feromon[cc]= 1 + 5 * atan( MzAdj[no][aux]->feromon[cc]/10);
350       }
351       aux=MzAdj[no][aux]->prox;
352     }
353   }
354   ActualizaProdutosDosProdutos();
355 }
356
357 /*****************************************************/
358 //auxiliary function
359 void SomaAsFeromonas(TREAL s){
360 int aux;
361   for(int no=0; no<NN ; no++){
362     aux=prim_MzAdj[no];
363     while(aux!=-1){
364       for(int cc=0;cc<NC;cc++){
365         MzAdj[no][aux]->feromon[cc]+= s;
366       }
367       aux=MzAdj[no][aux]->prox;
368     }
369   }
370   ActualizaProdutosDosProdutos();
371 }
372
373 /*****************************************************/
374 //reset the pheromones
375 void ReinicializaFeromonas(ARRAY_PARAMS *params_fer_in,
376                ARRAY_PARAMS *params_cst_in){
377 int aux;
378
379   params_cst=params_cst_in;
380   params_fer=params_fer_in;
381
382   for(int no=0; no<NN ; no++){
383     aux=prim_MzAdj[no];
384     while(aux!=-1){
385       MzAdj[no][aux]->ProdutoDosProdutos=1.0;
386       for(int cc=0;cc<NC;cc++){
387         MzAdj[no][aux]->feromon[cc]=1.0;
388         MzAdj[no][aux]->delta[cc]=0.0;
389         MzAdj[no][aux]->ProdutoDosProdutos*=pow(((TREAL)maximos[cc])/(INFIMO+(TREAL)(
               MzAdj[no][aux]->VctCustos[cc])),(*params_cst)[cc]);
390       }
391       aux=MzAdj[no][aux]->prox;
392     }
393   }
394   ActualizaProdutosDosProdutos();
395 }
396
397 /*****************************************************/
398 //reset the pheromones
399 void ReinicializaFeromonas(){
400   ReinicializaFeromonas(params_fer,params_cst);
401 }
402
403 /*****************************************************/
404 //auxiliary functions
405 int AdominaB(ARRAY_CUSTOS a, ARRAY_CUSTOS b){
406     int amelhor=0;
407     int apior=0;
408     int igual=0;
409     int m=NC;
410       //compara os valores dos custos
411       for(int cc=0;cc<m;cc++){
412         if(a[cc]<b[cc])
413           amelhor++;
414         else
415           if(a[cc]>b[cc])
416             apior++;
417           else
418             igual++;
419       }
420       if(igual==m){ //tem iguais custos
```

```
421           printf("=");
422           return -2;
423         }
424
425         if(amelhor+igual==m) {//a domina b (igual!=m)
426           printf("D");
427           return -1;
428         }
429
430         if(apior+igual==m) {//b domina a (igual!=0)
431           printf("-");
432           return 1;
433         }
434
435         //nenhum domina
436         return 0;
437
438     }
439
440   /*************************************************/
441   //auxiliary functions
442   void teste(){
443     ARRAY_CUSTOS menores;
444     int aux,cont,c,no;
445     for(c=0 ; c<NC ; c++){
446       menores[c]=MzAdj[0][prim_MzAdj[0]]->VctCustos[c];
447     }
448     for(no=0; no<NN ; no++){
449       aux=prim_MzAdj[no];
450       while(aux!=-1){
451         for(c=0;c<NC;c++){
452           if(MzAdj[no][aux]->VctCustos[c]<menores[c]){
453             menores[c]=MzAdj[no][aux]->VctCustos[c];
454           }
455         }
456         aux=MzAdj[no][aux]->prox;
457       }
458           aux=prim_MzAdj[no];
459       while(aux!=-1){
460         cont=0;
461         for(c=0;c<NC;c++){
462           if(MzAdj[no][aux]->VctCustos[c]==menores[c]){
463             cont++;
464           }
465         }
466         if(cont==NC)
467           printf("domina todos %d %d\n",no,aux);
468         aux=MzAdj[no][aux]->prox;
469       }
470     }
471   }
472
473   /*************************************************/
474   //auxiliary functions
475   void PRINT_FEROMONAS(char *t){
476
477     FILE *f;
478     f=fopen(t,"w");
479     fprintf(f,"NN=%d NC=%d", NN,NC);
480
481     for(int c=0;c<NC;c++){
482       fprintf(f,"\ncusto %d\n",c);
483       for(int ic=0;ic<NN;ic++){
484         for(int jc=0;jc<NN;jc++){
485           if(MzAdj[ic][jc]!=NULL) {
486             fprintf(f,"%6.5f\t",MzAdj[ic][jc]->feromon[c]);
487           }else{
488             fprintf(f,"%6.5f\t",0.0);
489           }
490         }
491         fprintf(f,"\n");
492       }
493       fprintf(f,"\n");
494     }
495     fclose(f);
496   }
497
498
```

```
499 /*************************************************/
500 //auxiliary functions
501 void pts_cst_prob(char *t){
502   FILE *f;
503   int aux;
504   f=fopen(t,"w");
505   fprintf(f,"ptscst = {");
506   fprintf(f,"{%lf,%lf,%lf}",MzAdj[0][prim_MzAdj[0]]->VctCustos[0],
507     MzAdj[0][prim_MzAdj[0]]->VctCustos[1], MzAdj[0][prim_MzAdj[0]]->ProdutoDosProdutos
          );
508   for(int no=0; no<NN ; no++){
509     aux=prim_MzAdj[no];
510     while(aux!=-1){
511       if(no>aux)
512         fprintf(f,",{%lf,%lf,%lf}",MzAdj[no][aux]->VctCustos[0],
513           MzAdj[no][aux]->VctCustos[1], MzAdj[no][aux]->ProdutoDosProdutos);
514       aux=MzAdj[no][aux]->prox;
515     }
516   }
517   fprintf(f,"};\n\n\n\n");
518   fprintf(f,"ptsfer = {");
519   fprintf(f,"{%lf,%lf,%lf}",MzAdj[0][prim_MzAdj[0]]->feromon[0],
520       MzAdj[0][prim_MzAdj[0]]->feromon[1], MzAdj[0][prim_MzAdj[0]]->ProdutoDosProdutos
            );
521   for(int no=0; no<NN ; no++){
522     aux=prim_MzAdj[no];
523     while(aux!=-1){
524       if(no>aux)
525           fprintf(f,",{%lf,%lf,%lf}",MzAdj[no][aux]->feromon[0],
526             MzAdj[no][aux]->feromon[1], MzAdj[no][aux]->ProdutoDosProdutos);
527       aux=MzAdj[no][aux]->prox;
528     }
529   }
530   fprintf(f,"};\n\n\n\n");
531
532
533   fprintf(f,"tudo = {");
534   fprintf(f,"{%lf,%lf,%lf,%lf,%lf}",MzAdj[0][prim_MzAdj[0]]->VctCustos[0],
535       MzAdj[0][prim_MzAdj[0]]->VctCustos[1],MzAdj[0][prim_MzAdj[0]]->feromon[0],
536       MzAdj[0][prim_MzAdj[0]]->feromon[1], MzAdj[0][prim_MzAdj[0]]->ProdutoDosProdutos
            );
537   for(int no=0; no<NN ; no++){
538     aux=prim_MzAdj[no];
539     while(aux!=-1){
540       if(no>aux)
541         fprintf(f,",{%lf,%lf,%lf,%lf,%lf}",MzAdj[no][aux]->VctCustos[0],
542         MzAdj[no][aux]->VctCustos[1],MzAdj[no][aux]->feromon[0],MzAdj[no][aux]->
              feromon[1],
543         MzAdj[no][aux]->ProdutoDosProdutos);
544       aux=MzAdj[no][aux]->prox;
545     }
546   }
547   fprintf(f,"};\n\n\n\n");
548   fclose(f);
549 }
550 };
551 #endif
```

C.1.3 Quad-tree Class

```
1  /***************************************************************************
2                            consts.cpp  -  description
3                               -------------------
4      date                 : Dec 2006
5      copyright            : (C) 2006 by Pedro Cardoso
6      email                : pcardoso@ualg.pt
7   ***************************************************************************
8  Resume:
9    Quadtree class
10  ***************************************************************************/
11 #pragma once
12 #include "consts.cpp"
13 #include "rede.h"
14 #include <math.h>
15
16
17 class QT
```

```
18  {
19  public:
20    struct QUAD_TREE{
21        int         arvore[MAXN][2];
22        int         narestas;
23        ARRAY_CUSTOS    pto;
24        struct  QUAD_TREE *prox[2],*pai;
25
26    };
27    ADJACENCY_MATRIX  ST;
28    QUAD_TREE     *ps;
29    int         nc;
30    int         e,d;
31    int         contdel;
32    int         verbose;
33    int         alterado;
34    clock_t       alterouas;
35    REDE        *rede;
36    TREAL       Qminy[NUMERO_MAXIMO_CUSTOS],Qminx[NUMERO_MAXIMO_CUSTOS];
37    int         contsol;
38    TREAL       Qmedio;
39    TREAL       delta;
40
41
42  /*************************************************/
43  //constructor
44  QT(REDE *rin)
45    {
46      int cc;
47      rede=rin;
48      nc=rede->NC;
49      ps=NULL;
50      e=0;
51      d=1;
52      alterado=1;
53      contsol=0;
54      for(cc=0;cc<rede->NC;cc++){
55        Qminx[cc]=INFINITO;
56        Qminy[cc]=INFINITO;
57      }
58      Qmedio=0;
59
60
61      if(_debug_ >= 2)
62        verbose=1;
63      else
64        verbose=0;
65
66      delta=0;
67
68    }
69
70
71  /*************************************************/
72  //destructor
73    virtual ~QT(void) {
74      ApagaArvore();
75    }
76
77
78  /*************************************************/
79  //delete tree
80  void ApagaArvore(){
81      if(ps!=NULL){
82        ApagaArvore(ps->prox[e]);
83        ApagaArvore(ps->prox[d]);
84        delete ps;
85      }
86    }
87
88  void ApagaArvore(QUAD_TREE *r){
89      if(r!=NULL){
90        ApagaArvore(r->prox[e]);
91        ApagaArvore(r->prox[d]);
92        delete r;
93      }
94    }
95
96
97
98  /*************************************************/
99  //read points
100 void LePontos(char fich[]){
```

```
101       int ii;
102       int n;
103       TCUSTOS aux;
104       ARRAY_CUSTOS v;
105       FILE *f;
106       TREAL *dummy;
107       ADJACENCY_MATRIX  dummyst;
108
109       if((f=fopen(fich,"r"))==NULL){
110         printf("ERRO: erro na abertura do ficheiro com com cj pareto inicial");
111       }
112       fscanf(f,"%d",&n);
113       for(ii=0;ii<n;ii++){
114         fscanf(f,"%lf",&aux);
115         v[0]=aux;
116         fscanf(f,"%lf",&aux);
117         v[1]=aux;
118         Update(&v,dummy,dummyst);
119       }
120       fclose(f);
121     }
122
123
124   /*****************************************************/
125   //
126   void PrintQT(QUAD_TREE *r){
127       if(r!=NULL){
128         printf("{%d,%d}, ",r->pto[0],r->pto[1]);
129         PrintQT(r->prox[e]);
130         PrintQT(r->prox[d]);
131       }
132     }
133
134
135     void PrintQT(){
136       if(ps!=NULL){
137         printf("{%d,%d}, ",ps->pto[0],ps->pto[1]);
138         PrintQT(ps->prox[e]);
139         PrintQT(ps->prox[d]);
140       }
141       printf("\n");
142     }
143
144   /*****************************************************/
145   //print to file
146   void PrintQT(FILE *f,QUAD_TREE *r){
147       if(r!=NULL){
148         fprintf(f,", {%d,%d}",r->pto[0],r->pto[1]);
149         PrintQT(f,r->prox[e]);
150         PrintQT(f,r->prox[d]);
151       }
152     }
153
154   void PrintQT(char *nf,char *pref, clock_t starttime){
155       FILE *f;
156       clock_t finish;
157       double duration;
158       char aux[256];
159
160         time(&finish);
161         if(alterado){
162           alterado=0;
163           f=fopen(nf,"w");
164           sprintf(aux,"%s={",pref);
165           fprintf(f,"%s",aux);
166           if(ps!=NULL){
167             fprintf(f,"{%d,%d}",ps->pto[0],ps->pto[1]);
168             PrintQT(f,ps->prox[e]);
169             PrintQT(f,ps->prox[d]);
170           }
171           fprintf(f,"}");
172           duration=difftime(alterouas, starttime);
173           fprintf(f,"\n\n\n%sTime=%8.4f",pref,duration);
174           duration=difftime(finish, starttime);
175           fprintf(f,"\n\n\n%sTotalTime=%8.4f",pref,duration);
176           fclose(f);
177         }
178     }
```

```
179
180
181
182 /*************************************************/
183 //print to file
184 void PrintQTFULL(FILE *f,QUAD_TREE *r){
185     int i;
186     if(r!=NULL){
187       fprintf(f,"%d \t %d\n",r->pto[0],r->pto[1]);
188       for(i=0;i<r->narestas;i++){
189         fprintf(f,"\t %d \t %d \n",r->arvore[i][0],r->arvore[i][1]);
190       }
191       PrintQTFULL(f,r->prox[e]);
192       PrintQTFULL(f,r->prox[d]);
193     }
194   }
195
196
197
198
199 void PrintQTFULL(char *nf,char *pref,clock_t starttime){
200     FILE *f;
201     clock_t finish;
202     double duration;
203     char aux[256];
204     int i;
205
206       time(&finish);
207       f=fopen(nf,"w");
208       sprintf(aux,"%s={",pref);
209       fprintf(f,"%d\n",contsol);
210       if(ps!=NULL){
211         fprintf(f,"%d \t %d\n",ps->pto[0],ps->pto[1]);
212         for(i=0;i<ps->narestas;i++){
213           fprintf(f,"\t %d \t %d \n",ps->arvore[i][0],ps->arvore[i][1]);
214         }
215         fprintf(f,"\n");
216         PrintQTFULL(f,ps->prox[e]);
217         PrintQTFULL(f,ps->prox[d]);
218       }
219       duration=difftime(alterouas, starttime);
220       fprintf(f,"\n\n\n%sTime=%8.4f",pref,duration);
221       duration=difftime(finish, starttime);
222       fprintf(f,"\n\n\n%sTotalTime=%8.4f",pref,duration);
223       fclose(f);
224   }
225
226 /*************************************************/
227 //merge qt's
228 int Merge(QT *qt,int *in, int *out){
229     return Merge(qt->ps,in,out);
230   }
231
232 int Merge(QUAD_TREE *r,int *in, int *out){
233     *in=0;
234     *out=0;
235     return Merge2(r,in,out);
236   }
237
238
239 int Merge2(QUAD_TREE *r,int *in, int *out){
240     int   cont=0,s,t;
241     TREAL erro;
242     ADJACENCY_MATRIX  mzadj;
243     LISTA_ADJACENCIA  *novo,*aux;
244     int inout=0;
245
246
247     for(int i=0;i<rede->NN;i++)
248     {
249       mzadj[i]=NULL;
250     }
251
252     if(r!=NULL){
253       //copia arvore para ADJACENCY_MATRIX[MAXN]
254       for(int i=0;i<r->narestas;i++){
255         novo=new LISTA_ADJACENCIA;
256         s=r->arvore[i][0];
257         t=r->arvore[i][1];
258         novo->no=t;
```

```
259           novo->prox=mzadj[s];
260           mzadj[s]=novo;
261         }
262         inout=Update(&(r->pto),&erro,mzadj);
263         if(inout){
264           *in+=1;
265           *out+=inout-1;
266         }
267
268         for(int i=0;i<r->narestas;i++){
269           novo=mzadj[i];
270           while(novo){
271             aux=novo->prox;
272             delete novo;
273             novo=aux;
274           }
275
276         }
277
278         cont+=Merge2(r->prox[e],in,out);
279         cont+=Merge2(r->prox[d],in,out);
280       }
281       return cont;
282     }
283
284   /*****************************************************/
285   //dominance relation
286   int AdominaB(ARRAY_CUSTOS *A, ARRAY_CUSTOS *B,TREAL *epsilon){
287       //-1  - A<B - A domina B
288       //0   - A<>B - não se dominam
289       //1   - B<A B domina A
290       //2   - A=B
291       TCUSTOS xa=(*A)[0],
292           ya=(*A)[1],
293           xb=(*B)[0],
294           yb=(*B)[1];
295       if((abs(xa-xb)<cd) && (abs(ya-yb)<cd)){
296         *epsilon=0.0;
297         return 2;
298       }
299
300       if(xa<=xb) {
301         if(ya<=yb){
302           if((xa!=xb) || (ya!=yb)){//A domina B
303             *epsilon=fabs((TREAL)(ya-yb)) + fabs((TREAL)(xa-xb));
304             return -1;
305           }else{//são iguais
306             *epsilon=0.0;
307             return 2;
308           }
309         }else{
310           if(xa!=xb){//não se dominam
311             return 0;
312           }else{//B domina A
313             return 1;
314           }
315         }
316       }else{
317         if(ya>=yb){//B domina A
318           return 1;
319         }else{
320           return 0;
321         }
322       }
323     }
324
325
326   /*****************************************************/
327   //
328   int DetectaDominancia(QUAD_TREE *r,ARRAY_CUSTOS *pto,TREAL *epsilon){
329       int i;
330       int aux=AdominaB(&(r->pto),pto,epsilon);
331       if((aux==-1) || (aux==2))
332         return 1;
333       for(i=0;i<=1;i++){
334         if((r->pto[i] >= (*pto)[i]) && (r->prox[i]!=NULL)){
335           if(DetectaDominancia(r->prox[i],pto,epsilon)==1)
336             return 1;
337         }
```

```
338       }
339       return 0;
340     }
341
342   /*************************************************/
343   //
344   int DeleteDominated(QUAD_TREE *r, ARRAY_CUSTOS *pto){
345       int i;
346       TREAL epsilon;
347       for(i=0;i<=1;i++){
348         if((*pto)[i] <= r->pto[i]){
349           if(r->prox[i]!=NULL){
350             DeleteDominated(r->prox[i],pto);
351           }
352         }
353   /*        if((*pto)[i] > r->pto[i])
354           break;*/
355
356       }
357       if(AdominaB(pto,&(r->pto),&epsilon)==-1){//*pto domina r->pto
358         contdel++;
359         if(verbose) printf("-");
360         QUAD_TREE *rdir,*resq,*pai;
361         if((r->prox[e]==NULL) && (r->prox[d]==NULL)){//se for folha é só apagar a memos
                que seja a raiz e nesse caso e so substituir
362           if(r->pai==NULL){//é a raiz
363                       delete r;
364             ps=NULL;
365             return 1;
366           }
367           pai=r->pai;
368           if(pai->prox[e]==r)
369             pai->prox[e]=NULL;
370           else
371             pai->prox[d]=NULL;
372           delete r;
373           return 1;
374         }
375
376         //indice do menor ramo de r
377         if(r->prox[e]!=NULL){
378           pai=r->pai;
379           resq=r->prox[e];
380           rdir=r->prox[d];
381           r->pto[0]=resq->pto[0];
382           r->pto[1]=resq->pto[1];
383           r->prox[e]=resq->prox[e];
384           if(resq->prox[e]!=NULL)
385             resq->prox[e]->pai=r;
386
387           r->prox[d]=resq->prox[d];
388           if(resq->prox[d]!=NULL)
389               resq->prox[d]->pai=r;
390
391           delete resq;
392           if(rdir!=NULL)//se existe ramo para a direita insere-o
393             TreeInsert(ps,rdir);
394         }else{//só existe o ramo para a direita - apaga r
395           rdir=r->prox[d];
396           r->pto[0]=rdir->pto[0];
397           r->pto[1]=rdir->pto[1];
398           r->prox[e]=rdir->prox[e];
399           if(rdir->prox[e]!=NULL)
400             rdir->prox[e]->pai=r;
401
402           r->prox[d]=rdir->prox[d];
403           if(rdir->prox[d]!=NULL)
404             rdir->prox[d]->pai=r;
405
406           delete rdir;
407         }
408       }
409       return -10000;
410     }
411
412
413   /*************************************************/
414   //
415   void TreeInsert(QUAD_TREE *r,QUAD_TREE *s){
```

```
416       if(s->prox[e]!=NULL)
417         TreeInsert(r,s->prox[e]);
418       if(s->prox[d]!=NULL)
419         TreeInsert(r,s->prox[d]);
420       Insert(r,&(s->pto));
421       delete s;
422     }
423
424
425   /*****************************************************/
426   //
427   void Insert(QUAD_TREE *r, ARRAY_CUSTOS *pto){
428       //ve qual o máximo dos valores obtidos
429       Qmedio=0.0;
430       if(Qminx[0]>(*pto)[0]){
431           Qminx[0]=(*pto)[0];
432           Qminx[1]=(*pto)[1];
433       }
434       if(Qminy[1]>(*pto)[1]){
435           Qminy[0]=(*pto)[0];
436           Qminy[1]=(*pto)[1];
437       }
438       Qmedio=(Qminy[0]+Qminx[0])/rede->NC;
439
440       if(ps==NULL){
441         if(verbose) printf("+");
442         ps=new QUAD_TREE;
443         ps->prox[e]=NULL;
444         ps->prox[d]=NULL;
445         ps->pai=NULL;
446         for(int k=0;k<nc; k++)
447           ps->pto[k]=(*pto)[k];
448
449         Qminx[0]=Qminy[0]=(*pto)[0];
450         Qminx[1]=Qminy[1]=(*pto)[1];
451
452         //copia a arvore
453         LISTA_ADJACENCIA *auxp;
454         int i,linha=0;
455         ps->narestas=0;
456         for(i=0;i<rede->NN;i++){
457           auxp=ST[i];
458           while(auxp!=NULL){
459             if(i<auxp->no){
460               ps->arvore[linha][0]=i;
461               ps->arvore[linha][1]=auxp->no;
462               ps->narestas+=1;
463               linha++;
464             }
465             auxp=auxp->prox;
466           }
467         }
468
469
470       }else{
471         int i;
472         QUAD_TREE *novo;
473
474         if((*pto)[0] < r->pto[0])
475           i=0;
476         else
477           i=1;
478
479         if(r->prox[i]!=NULL)
480             Insert(r->prox[i],pto);
481         else{
482           novo      = new QUAD_TREE;
483           novo->pto[0]  = (*pto)[0];
484           novo->pto[1]  = (*pto)[1];
485           novo->prox[e] = NULL;
486           novo->prox[d] = NULL;
487           novo->pai   = r;
488           r->prox[i]    = novo;
489           //copia a arvore
490           LISTA_ADJACENCIA *auxp;
491           int i,linha=0;
492           novo->narestas=0;
493           for(i=0;i<rede->NN;i++){
494             auxp=ST[i];
495             while(auxp!=NULL){
```

```
496             if(i<auxp->no){
497               novo->arvore[linha][0]=i;
498               novo->arvore[linha][1]=auxp->no;
499               novo->narestas+=1;
500               linha++;
501             }
502             auxp=auxp->prox;
503             }
504           }
505
506         }
507       }
508     }
509
510
511   /*************************************************/
512   //
513   int Update(ARRAY_CUSTOS *pto, TREAL *epsilon, ADJACENCY_MATRIX  STin){
514       //epsilon - distancia ao ponto mais proximo
515       int aux;
516       int i,linha=0;
517
518       //para copiar a arvore tem de copiar os ponteiros
519       for(i=0;i<rede->NN;i++)
520         ST[i] = STin[i];
521
522       if(ps==NULL){
523         if(verbose) printf("+");
524         ps=new QUAD_TREE;
525         ps->prox[e]=NULL;
526         ps->prox[d]=NULL;
527         ps->pai=NULL;
528         for(int k=0;k<nc; k++)
529           ps->pto[k]=(*pto)[k];
530
531
532         //copia a arvore
533         LISTA_ADJACENCIA *auxp;
534         ps->narestas=0;
535         for(i=0;i<rede->NN;i++){
536           auxp=ST[i];
537           while(auxp!=NULL){
538             if(i<auxp->no){
539               ps->arvore[linha][0]=i;
540               ps->arvore[linha][1]=auxp->no;
541               ps->narestas+=1;
542               linha++;
543             }
544             auxp=auxp->prox;
545           }
546         }
547
548         contsol++;
549         return 1;
550       }
551       *epsilon=INFINITO;
552       aux=DetectaDominancia(ps,pto,epsilon);
553       if(DetectaDominancia(ps,pto,epsilon)==1){
554           return 0;
555       }
556       *epsilon=-1;//melhorou logo devolve negativo
557       contdel=0;
558       if(verbose) printf("+");
559       DeleteDominated(ps,pto);
560       Insert(ps,pto);
561       alterado=1;
562       time(&alterouas);
563       contsol-=contdel;//os que apagou
564       contsol++;//1 que adicionou
565       return contdel+1; //os que apagou mais +1 de inserir
566     }
567
568   /*************************************************/
569   //funtions to update the pheromone matrices using the elements of the QT
570   void UpdateDeltaFeromonas(TREAL Q){
571       int i,s,t,cc;
572       if(ps!=NULL){
573         for(i=0;i<ps->narestas;i++){
574           s=ps->arvore[i][0];
```

```
575           t=ps->arvore[i][1];
576           for(cc=0;cc<rede->NC;cc++){
577             rede->MzAdj[s][t]->delta[cc]+= Q/ps->pto[cc]+delta;
578             rede->MzAdj[t][s]->delta[cc]+= Q/ps->pto[cc]+delta;
579           }
580         }
581         UpdateDeltaFeromonas(Q,ps->prox[e]);
582         UpdateDeltaFeromonas(Q,ps->prox[d]);
583       }
584
585     }
586
587     void UpdateDeltaFeromonas(TREAL Q, QUAD_TREE *r){
588       int i,s,t,cc;
589       if(r!=NULL){
590         for(i=0;i<r->narestas;i++){
591           s=r->arvore[i][0];
592           t=r->arvore[i][1];
593           for(cc=0;cc<rede->NC;cc++){
594             rede->MzAdj[s][t]->delta[cc]+= Q/r->pto[cc]+delta;
595             rede->MzAdj[t][s]->delta[cc]+= Q/r->pto[cc]+delta;
596           }
597         }
598         UpdateDeltaFeromonas(Q,r->prox[e]);
599         UpdateDeltaFeromonas(Q,r->prox[d]);
600       }
601     }
602
603
604   /*****************************************************/
605   //funtions to update the pheromone matrices using the elements of the QT
606   //probabilistic update
607   int UpdateDeltaFeromonas_p(TREAL Q, int p){
608       int i,s,t,cc;
609       int qtusou=0;
610       if(ps!=NULL){
611         if(rand()%100 < p){
612           qtusou++;
613           for(i=0;i<ps->narestas;i++){
614             s=ps->arvore[i][0];
615             t=ps->arvore[i][1];
616             for(cc=0;cc<rede->NC;cc++){
617               rede->MzAdj[s][t]->delta[cc]+= Q/ps->pto[cc]+delta;
618               rede->MzAdj[t][s]->delta[cc]+= Q/ps->pto[cc]+delta;
619             }
620           }
621         }
622         qtusou+=UpdateDeltaFeromonas_p(Q,ps->prox[e],p);
623         qtusou+=UpdateDeltaFeromonas_p(Q,ps->prox[d],p);
624       }
625       return qtusou;
626     }
627
628   int UpdateDeltaFeromonas_p(TREAL Q, QUAD_TREE *r,int p){
629       int i,s,t,cc;
630       int qtusou=0;
631       if(r!=NULL){
632         if(rand()%100 < p){
633           qtusou++;
634           for(i=0;i<r->narestas;i++){
635             s=r->arvore[i][0];
636             t=r->arvore[i][1];
637             for(cc=0;cc<rede->NC;cc++){
638               rede->MzAdj[s][t]->delta[cc]+= Q/r->pto[cc]+delta;
639               rede->MzAdj[t][s]->delta[cc]+= Q/r->pto[cc]+delta;
640             }
641           }
642         }
643         qtusou+=UpdateDeltaFeromonas_p(Q,r->prox[e],p);
644         qtusou+=UpdateDeltaFeromonas_p(Q,r->prox[d],p);
645       }
646       return qtusou;
647     }
648
649   /*****************************************************/
650   //funtions to update the pheromone matrices using the elements of the QT
651   //solutions in interval
```

```
652   int UpdateDeltaFeromonasVizinhanca(TREAL Q,TREAL ferx0){
653     int   i,s,t,cc;
654     int   npassos=(int)((maxpar-minpar)/fabs(passopar));
655     TREAL   h=((TREAL)(Qminy[0]-Qminx[0]))/((TREAL)npassos);
656     TREAL   a=((TREAL)(Qminy[0])+ferx0/fabs(passopar)*h);
657     TREAL   b=a+h;
658     TREAL   Qaux=0.0;
659     int     qtusou=0;
660
661
662     if(ps!=NULL){
663       if((ps->pto[0]>=a) && (ps->pto[0]<=b)){
664         Qaux+=Q;
665       }
666       if(Qaux>0.0){
667         qtusou++;
668         for(i=0;i<ps->narestas;i++){
669           s=ps->arvore[i][0];
670           t=ps->arvore[i][1];
671           for(cc=0;cc<rede->NC;cc++){
672             rede->MzAdj[s][t]->delta[cc]+= Qaux/((TREAL)ps->pto[cc])+delta;
673             rede->MzAdj[t][s]->delta[cc]+= Qaux/((TREAL)ps->pto[cc])+delta;
674           }
675         }
676       }
677       qtusou+=UpdateDeltaFeromonasVizinhanca(Q,ps->prox[e],a,b,h);
678       qtusou+=UpdateDeltaFeromonasVizinhanca(Q,ps->prox[d],a,b,h);
679     }
680     return qtusou;
681   }
682
683
684
685
686 int UpdateDeltaFeromonasVizinhanca(
687                     TREAL Q,
688                     QUAD_TREE *r,
689                     TREAL a,
690                     TREAL b,
691                     TREAL h){
692     int i,s,t,cc;
693     TREAL Qaux=0.0;
694     int qtusou=0;
695
696     if(r!=NULL){
697       if((r->pto[0]>=a) && (r->pto[0]<=b)){
698         Qaux+=Q;
699       }
700       if(Qaux>0.0){
701         qtusou++;
702         for(i=0;i<r->narestas;i++){
703           s=r->arvore[i][0];
704           t=r->arvore[i][1];
705           for(cc=0;cc<rede->NC;cc++){
706             rede->MzAdj[s][t]->delta[cc]+= Q/((TREAL)r->pto[cc])+delta;
707             rede->MzAdj[t][s]->delta[cc]+= Q/((TREAL)r->pto[cc])+delta;
708           }
709         }
710       }
711       qtusou+=UpdateDeltaFeromonasVizinhanca(Q,r->prox[e],a,b,h);
712       qtusou+=UpdateDeltaFeromonasVizinhanca(Q,r->prox[d],a,b,h);
713     }
714     return qtusou;
715   }
716
717
718
719
720 /*************************************************/
721 //funtions to update the pheromone matrices using the elements of the QT
722 //angle pheromone update
723 int UpdateDeltaFeromonasVizinhancaAngulo(
724                     TREAL Q,
725                     TREAL ferx0){
726     int   i,s,t,cc;
727     int   npassos=(int)((maxpar-minpar)/fabs(passopar));
728     TREAL alphamin=atan(Qminy[1]/Qminy[0]);
729     TREAL alphamax=atan(Qminx[1]/Qminx[0]);
730     TREAL alphah=(alphamax-alphamin)/((TREAL) npassos+1);
731     TREAL alphaa=alphamin+(ferx0/fabs(passopar))*alphah;
```

```
732     TREAL alphab;
733         TREAL Qaux=0.0;
734     TREAL beta;
735     int   qtusou=0;
736 
737       alphab=alphaa+alphah;
738 
739     if(ps!=NULL){
740       do{
741         beta=atan(((TREAL)ps->pto[1])/((TREAL)ps->pto[0]));
742         if((beta >= alphaa) && (beta <= alphab))
743             Qaux+=Q;
744         if(Qaux>0.0){
745           qtusou++;
746           for(i=0;i<ps->narestas;i++){
747             s=ps->arvore[i][0];
748             t=ps->arvore[i][1];
749             for(cc=0;cc<rede->NC;cc++){
750               rede->MzAdj[s][t]->delta[cc]+= Qaux/((TREAL)ps->pto[cc])+delta;
751               rede->MzAdj[t][s]->delta[cc]+= Qaux/((TREAL)ps->pto[cc])+delta;
752             }
753           }
754         }
755         qtusou+=UpdateDeltaFeromonasVizinhancaAngulo(Q,ps->prox[e],alphaa,alphab,
                alphah);
756         qtusou+=UpdateDeltaFeromonasVizinhancaAngulo(Q,ps->prox[d],alphaa,alphab,
                alphah);
757         if(qtusou<3){
758           if(_debug_) printf("U");
759           alphaa-=alphah;
760           alphab+=alphah;
761         }
762       }while(qtusou==0);
763       return qtusou;
764     }
765     printf("UpdateDeltaFeromonasVizinhancaAngulo VAZIO????????\n\n");
766     return 0;
767 
768   }
769 
770 int UpdateDeltaFeromonasVizinhancaAngulo(
771                   TREAL Q,
772                   QUAD_TREE *r,
773                   TREAL alphaa,
774                   TREAL alphab,
775                   TREAL alphah){
776     int i,s,t,cc;
777     TREAL Qaux=0.0;
778     int qtusou=0;
779     TREAL beta;
780 
781     if(r!=NULL){
782       beta=atan(((TREAL)r->pto[1])/((TREAL)r->pto[0]));
783       if((beta >= alphaa) && (beta <= alphab))
784         Qaux+=Q;
785       if(Qaux>0.0){
786         qtusou++;
787         for(i=0;i<r->narestas;i++){
788           s=r->arvore[i][0];
789           t=r->arvore[i][1];
790           for(cc=0;cc<rede->NC;cc++){
791             rede->MzAdj[s][t]->delta[cc]+= Q/((TREAL)r->pto[cc])+delta;
792             rede->MzAdj[t][s]->delta[cc]+= Q/((TREAL)r->pto[cc])+delta;
793           }
794         }
795       }
796       qtusou+=UpdateDeltaFeromonasVizinhancaAngulo(Q,r->prox[e],alphaa,alphab,alphah);
797       qtusou+=UpdateDeltaFeromonasVizinhancaAngulo(Q,r->prox[d],alphaa,alphab,alphah);
798     }
799     return qtusou;
800   }
801 };
```

C.1.4 ANT Class

```
1  /***************************************************************************
2                           consts.cpp  -  description
3                              -------------------
4      date                 : Dec 2006
5      copyright            : (C) 2006 by Pedro Cardoso
```

```
6       email                : pcardoso@ualg.pt
7    ***************************************************************************
8   Resume:
9     ANT class
10   ***************************************************************************/
11
12  #pragma once
13  #ifndef ANTPACKET_H
14  #define ANTPACKET_H
15
16  #include  <math.h>
17  #include  "rede.h"
18  #include  "quadtree.h"
19
20
21  class ANT {
22  public:
23      REDE        *rede;
24    ARRAY_CUSTOS    Custos,copiacustos;
25    ADJACENCY_MATRIX  ST, copia;
26    int         tabu[MAXN];
27    int         graus[MAXN];
28
29  /*************************************************/
30  //constructor
31  ANT(REDE *redein){
32      rede    = redein;
33    for(int i=0;  i<rede->NN;   i++){
34      ST[i]=NULL;
35    }
36    for(int cc = 0; cc < rede->NC;  cc++){
37      Custos[cc] = 0;
38    }
39  }
40
41  /*************************************************/
42  //destructor
43  ~ANT(){
44  LISTA_ADJACENCIA *aux,*auxapaga;
45  int i;
46    for(i=0;i<rede->NN;i++){
47      aux=ST[i];
48      while(aux){
49        auxapaga=aux;
50        aux=aux->prox;
51        delete auxapaga;
52      }
53    }
54
55  }
56
57  /*************************************************/
58  //add edge to the ST
59  void ADD_ARESTA(int s, int t){
60    LISTA_ADJACENCIA *aux;
61
62    //(s,t)
63    graus[s]++;
64    graus[t]++;
65
66    aux=new LISTA_ADJACENCIA;
67    aux->no=t;
68    if(ST[s]==NULL)
69      aux->prox=NULL;
70    else
71      aux->prox=ST[s];
72    ST[s]=aux;
73
74    //(t,s)
75    aux=new LISTA_ADJACENCIA;
76    aux->no=s;
77    if(ST[t]==NULL)
78      aux->prox=NULL;
79    else
80      aux->prox=ST[t];
81    ST[t]=aux;
82
83    //add weights
84    for(int cc=0;cc<rede->NC;cc++){
85      Custos[cc]+=rede->MzAdj[s][t]->VctCustos[cc];
86    }//for
```

```
87  }
88
89  /*************************************************/
90  //remove edge from the ST
91  void REMOVE_ARESTA(int s, int t){
92    LISTA_ADJACENCIA *aux,*antaux;
93    //(s,t)
94    graus[s]--;
95    graus[t]--;
96
97    aux=ST[s];
98    antaux=NULL;
99    while(aux->no!=t){
100     antaux=aux;
101     aux=aux->prox;
102   }
103   if(antaux==NULL){
104     ST[s]=aux->prox;
105   }else{
106     antaux->prox=aux->prox;
107   }
108   delete aux;
109
110   //(t,s)
111   aux=ST[t];
112   antaux=NULL;
113   while(aux->no!=s){
114     antaux=aux;
115     aux=aux->prox;
116   }
117   if(antaux==NULL){
118     ST[t]=aux->prox;
119   }else{
120     antaux->prox=aux->prox;
121   }
122   delete aux;
123
124   //remove weights
125   for(int cc=0;cc<rede->NC;cc++){
126     Custos[cc]-=rede->MzAdj[s][t]->VctCustos[cc];
127   }//for
128 }
129
130 /*************************************************/
131 //print tree
132 void PRINT(){
133   LISTA_ADJACENCIA *aux;
134   for(int i=0;i<rede->NN;i++){
135     aux=ST[i];
136     while(aux){
137       if(i<aux->no)
138         printf("%d-%d  ",i,aux->no);
139       aux=aux->prox;
140     }
141   }
142   printf("weight:");
143   for(int i=0;i<rede->NC;i++)
144       printf("%lf ",Custos[i]);
145   printf("\n");
146 }
147
148 /*************************************************/
149 //update pheromones
150 void ActualizaDeltaFeromonas(TREAL Q ){
151
152   LISTA_ADJACENCIA    *aux;
153   for(int no=0;no<rede->NN;no++){
154     aux=ST[no];
155     while(aux!=NULL){
156       for(int cc=0;cc<rede->NC;cc++){
157         rede->MzAdj[no][aux->no]->delta[cc]+= Q/((TREAL)Custos[cc]);
158         rede->MzAdj[aux->no][no]->delta[cc]+= Q/((TREAL)Custos[cc]);
159       }
160       aux=aux->prox;
161     }
162   }
163 }
164
165 /*************************************************/
```

```
166 //copy  tree
167 void copia_arvore(){
168     LISTA_ADJACENCIA *aux,*cp,*last;
169   int i,cc;
170
171   for(i=0;i<rede->NN;i++){
172     copia[i]=NULL;
173     aux=ST[i];
174     while(aux!=NULL){
175       cp=new LISTA_ADJACENCIA;
176       cp->no=aux->no;
177       cp->prox=NULL;
178       if(copia[i]==NULL){
179         copia[i]=cp;
180         last=cp;
181       }else{
182         last->prox=cp;
183         last=cp;
184       }
185       aux=aux->prox;
186     }
187   }
188   for(cc=0;cc<rede->NC;cc++){
189     copiacustos[cc]=Custos[cc];
190   }//for
191 }
192
193 /*********************************************/
194 //restaures  tree
195 void restaura_arvore(){
196     LISTA_ADJACENCIA *aux,*prox;
197   int i,cc;
198   //printf("(%d,%d) ",s,t);
199   //(s,t)
200   //apaga ST
201   for(i=0;i<rede->NN;i++){
202     aux=ST[i];
203     while(aux!=NULL){
204       prox=aux->prox;
205       delete aux;
206       aux=prox;
207     }
208   }
209   //copia da cópia
210   for(i=0;i<rede->NN;i++){
211     ST[i]=copia[i];
212   }
213   for(cc=0;cc<rede->NC;cc++){
214     Custos[cc]=copiacustos[cc];
215   }//for
216 }
217
218 /*********************************************/
219 //delete tree
220 void  ApagaArvore(){
221 LISTA_ADJACENCIA    *aux1,*aux2;
222   for(int i=0; i<rede->NN ;  i++){
223     aux1=ST[i];
224     while(aux1){
225         aux2=aux1->prox;
226         delete aux1;
227         aux1=aux2;
228     }
229   }
230 }
231
232 };
233 #endif
```

C.2 MONACO

C.2.1 MOST_MONACO Class

```
1 /***************************************************************************
2                          antpacket.h  -  description
3                          -------------------
4     begin                : Qua Nov 6 2002
```

```
5       copyright            : (C) 2002 by Pedro Cardoso
6       email                : pcardoso@ualg.pt
7    ***************************************************************************/
8
9    /***************************************************************************
10   *                                                                         *
11   *    This program is free software; you can redistribute it and/or modify  *
12   *    it under the terms of the GNU General Public License as published by  *
13   *    the Free Software Foundation; either version 2 of the License, or     *
14   *    (at your option) any later version.                                   *
15   *                                                                         *
16   ***************************************************************************/
17   #pragma once
18   #ifndef ANTPACKET_H
19   #define ANTPACKET_H
20
21
22   /**
23     *@author Pedro Cardoso
24     */
25   #include  <math.h>
26   #include  "rede.h"
27   #include  "ant.h"
28   #include  "quadtree.h"
29
30
31   class MOMST_MONACO:
32     public ANT{
33   public:
34
35
36       REDE        *rede;
37     ARRAY_CUSTOS    Custos,copiacustos;
38     ADJACENCY_MATRIX  ST, copia;
39     int         tabu[MAXN];
40
41
42   MOMST_MONACO(REDE *redein){
43       rede    = redein;
44     for(int i=0;  i<rede->NN;   i++){
45       ST[i]=NULL;
46     }
47     for(int cc = 0; cc < rede->NC;  cc++){
48       Custos[cc] = 0;
49     }
50   }
51
52
53   /************************************************/
54   ~MOMST_MONACO(){
55   LISTA_ADJACENCIA *aux,*auxapaga;
56   int i;
57     for(i=0;i<rede->NN;i++){
58       aux=ST[i];
59       while(aux){
60         auxapaga=aux;
61         aux=aux->prox;
62         delete auxapaga;
63       }
64     }
65
66   }
67
68   /************************************************/
69   void ADD_ARESTA(int s, int t){
70     LISTA_ADJACENCIA *aux;
71     aux=new LISTA_ADJACENCIA;
72     aux->no=t;
73     if(ST[s]==NULL)
74       aux->prox=NULL;
75     else
76       aux->prox=ST[s];
77     ST[s]=aux;
78
79     //(t,s)
80     aux=new LISTA_ADJACENCIA;
81     aux->no=s;
82     if(ST[t]==NULL)
83       aux->prox=NULL;
84     else
85       aux->prox=ST[t];
86     ST[t]=aux;
```

```
87
88    for(int cc=0;cc<rede->NC;cc++){
89      Custos[cc]+=rede->MzAdj[s][t]->VctCustos[cc];
90    }//for
91  }
92
93  /*********************************************/
94  void REMOVE_ARESTA(int s, int t){
95    LISTA_ADJACENCIA *aux,*antaux;
96    //(s,t)
97    aux=ST[s];
98    antaux=NULL;
99    while(aux->no!=t){
100     antaux=aux;
101     aux=aux->prox;
102   }
103   if(antaux==NULL){
104     ST[s]=aux->prox;
105   }else{
106     antaux->prox=aux->prox;
107   }
108   delete aux;
109
110   //(t,s)
111   aux=ST[t];
112   antaux=NULL;
113   while(aux->no!=s){
114     antaux=aux;
115     aux=aux->prox;
116   }
117   if(antaux==NULL){
118     ST[t]=aux->prox;
119   }else{
120     antaux->prox=aux->prox;
121   }
122   delete aux;
123
124   //retira ao custos
125   for(int cc=0;cc<rede->NC;cc++){
126     Custos[cc]-=rede->MzAdj[s][t]->VctCustos[cc];
127   }//for
128 }
129
130
131 /*********************************************/
132 void PRINT(){
133   LISTA_ADJACENCIA *aux;
134   for(int i=0;i<rede->NN;i++){
135     aux=ST[i];
136     while(aux){
137       if(i<aux->no)
138         printf("%d-%d  ",i,aux->no);
139       aux=aux->prox;
140     }
141   }
142   printf("custo:");
143   for(int i=0;i<rede->NC;i++)
144       printf("%d ",Custos[i]);
145   printf("\n");
146 }
147
148
149
150
151
152 void ActualizaDeltaFeromonas(TREAL Q ){
153   LISTA_ADJACENCIA    *aux;
154   for(int no=0;no<rede->NN;no++){
155     aux=ST[no];
156     while(aux!=NULL){
157       for(int cc=0;cc<rede->NC;cc++){
158         rede->MzAdj[no][aux->no]->delta[cc]+= Q/((TREAL)Custos[cc]);
159         rede->MzAdj[aux->no][no]->delta[cc]+= Q/((TREAL)Custos[cc]);
160       }
161       aux=aux->prox;
162     }
163   }
164 }
165
```

```
166
167 void AntSystemST_PORCAMINHOS(){
168 int           contnosinseridos,idxno;
169 int           nobase,noin,noant;
170 int           nosadj;//indice para percorrer a lista de nós adjacentes
171 int             inictblst=0,tabulist[MAXN+1],tabulistback[MAXN+1]; //-1 nao inserido,
        n>-1 - subarvore onde foi inserido
172 TREAL         somatotal;
173 int         idxsubarvore,i;
174 TREAL         r,soma;
175 LISTA_INTEIROS      *antint,*auxint,*subarvores[MAXN],*fimsubarvores[MAXN]; //vai ter
        as subarvores com os varios indices
176
177
178   //printf("*************************************");
179   //Cria uma lista com todos os nós
180   for(i = 0; i<rede->NN; i++){
181     tabulistback[i]=i-1;
182     tabulist[i]=i+1;
183     subarvores[i]=NULL; //não era necessario colocar tantas a NULL
184   }
185   //comeca a criar a arvore
186   contnosinseridos=0;
187   idxsubarvore=0;
188   nobase=-1;
189   while(contnosinseridos<rede->NN){
190     if(nobase==-1){
191       //vai iniciar uma nova sub-arvore a partir de um no escolhido
192       //entre os ainda nao usados em nenhuma arvore
193       idxsubarvore++;
194       //escolhe aleatoriamente idx ordenado do no de onde vai (re)começar
195       idxno=rand()%(rede->NN-contnosinseridos);
196       noant=-1;
197       for(i=0,nobase=inictblst;i<idxno;i++){
198         noant=nobase;
199         nobase=tabulist[nobase];
200       }
201       if(noant==-1){
202         //vai colocar o primeiro da tabulist
203         inictblst=tabulist[inictblst];
204       }else{
205         //vai colocar um do meio da tabulist
206         tabulist[noant]=tabulist[nobase];
207         if(tabulist[nobase]<MAXN)
208           tabulistback[tabulist[nobase]]=tabulistback[nobase]; //se for o ultimo tenta
                aceder a uma posição não definida. Será que vai dar problemas?
209       }
210       tabulist[nobase]=-idxsubarvore;
211       auxint=new LISTA_INTEIROS;
212       auxint->no=nobase;
213       auxint->prox=NULL;
214       //é sempre o primeiro das subarvores
215       subarvores[idxsubarvore]=auxint;
216       fimsubarvores[idxsubarvore]=auxint;
217       contnosinseridos++;
218     }
219     //escolhe um no a partir do ultimo no pseudo-aleatoriamente
220     somatotal=rede->SomaDoProdutoDosProdutos[nobase];
221     //determina um valor aleatório entre 0 e somatotal
222     r=(TREAL)(rand()*somatotal/(RAND_MAX+1.0));
223     soma=0.0;
224     nosadj=rede->prim_MzAdj[nobase];
225     noin=-1;
226     while(soma<=r){
227       soma +=  rede->MzAdj[nobase][nosadj]->ProdutoDosProdutos;
228       if(soma<=r){
229         nosadj=rede->MzAdj[nobase][nosadj]->prox;
230       }else{
231         noin=nosadj;
232       }
233     }
234     //se o no escolhido ja pertencia à subarvore em construção então a subarvore acaba
235     //e vai começar uma nova
236     if(tabulist[noin]==-idxsubarvore){ //tenta ligar a um no que pertence à subarvore
          actual
237       nobase=-1;//logo não faz nada e começa outra arvore
238     }else{
```

```
239         //se o no escolhido ainda nao estava na arvore acrescenta actual
240         //acrescenta a aresta
241         ADD_ARESTA(nobase,noin);
242         //se intersectou com outra arvore então
243         if(tabulist[noin]<0){
244           //em tabulist[noin] está -indice da árvore a que pertence noin
245           //vai comecar nova arvore
246           nobase=-1;
247             //faz a actulização dos indices de subarvore
248           auxint=subarvores[idxsubarvore];
249           while(auxint){
250             tabulist[auxint->no]=tabulist[noin];
251             auxint=auxint->prox;
252           }
253           fimsubarvores[idxsubarvore]->prox=subarvores[-tabulist[noin]];
254           subarvores[-tabulist[noin]]=subarvores[idxsubarvore];
255           subarvores[idxsubarvore]=NULL;
256           fimsubarvores[idxsubarvore]=NULL;
257           idxsubarvore--;
258         }else{
259           // não intersectou com nada vai continuar
260           // a acrescentar nós à subarvore em construção
261           contnosinseridos++;
262           if(inictblst==noin){//se for o primeiro da lista
263             inictblst=tabulist[inictblst];
264             tabulist[noin]=-idxsubarvore;
265
266           }else{
267             tabulist[tabulistback[noin]]=tabulist[noin];
268             if(tabulist[noin]<MAXN)
269               tabulistback[tabulist[noin]]=tabulistback[noin];
270             tabulist[noin]=-idxsubarvore;
271           }
272           nobase=noin;
273           auxint=new LISTA_INTEIROS;
274           auxint->no=noin;
275           auxint->prox=subarvores[idxsubarvore];
276           subarvores[idxsubarvore]=auxint;
277         }
278       }
279     }//while
280
281   //MERGE DAS SUBARVORES
282     //Agora se houver mais de uma subarvore tem de fazer o merge
283     for(;idxsubarvore>1;idxsubarvore--){
284       //Escolhe de entre os nos com indice de subarvore maior
285       //o que melhor resultado tem a ligar-se com outra subarvore
286       //de formula probabilistica! ou seja agora nao permite escolha
287       //de arestas que liguem nos da mesma sub arvore
288       somatotal=0.0;
289       auxint=subarvores[idxsubarvore];
290       while(auxint){
291         nosadj=rede->prim_MzAdj[auxint->no];
292         while(nosadj!=-1){
293           if(tabulist[nosadj]!=-idxsubarvore){ // so considera nos noutras arvore
294             somatotal +=  rede->MzAdj[auxint->no][nosadj]->ProdutoDosProdutos;
295           }
296           nosadj=rede->MzAdj[auxint->no][nosadj]->prox;
297         }
298         auxint=auxint->prox;
299       }     //determina um valor aleatório entre 0 e somatotal
300
301       int rr=rand()+1;
302       r=(((TREAL)(rr))*somatotal)/((TREAL)(RAND_MAX+1))*0.999999999;
303       soma=0.0;
304       auxint=subarvores[idxsubarvore];
305       int sai=0;
306       nosadj=-1;
307       noin=-1;
308       while(auxint && !sai){
309         nosadj=rede->prim_MzAdj[auxint->no];
310         while(soma<r && nosadj!=-1){
311           if(tabulist[nosadj]!=-idxsubarvore){ // so considera nos noutras arvore
312             soma +=    rede->MzAdj[auxint->no][nosadj]->ProdutoDosProdutos;
313           }
314
315           if(soma<=r){
```

```
316           nosadj=rede->MzAdj[auxint->no][nosadj]->prox;
317         }else{
318           nobase=auxint->no;
319           noin=nosadj;
320           sai=1;
321         }
322       }
323       auxint=auxint->prox;
324     }
325     ADD_ARESTA(nobase,noin);
326     //acrescenta custos
327     for(int cc=0;cc<rede->NC;cc++){
328       Custos[cc]+=rede->MzAdj[nobase][nosadj]->VctCustos[cc];
329     }
330     //passa todos os nos de indice idxsubarvore para o indice da sub arvore que ligou
331     auxint=subarvores[idxsubarvore];
332     while(auxint){
333       tabulist[auxint->no]=tabulist[noin];
334       auxint=auxint->prox;
335     }
336
337     fimsubarvores[idxsubarvore]->prox=subarvores[-tabulist[noin]];
338     subarvores[-tabulist[noin]]=subarvores[idxsubarvore];
339   }
340   //apaga lista
341   antint=subarvores[1];
342   while(antint){
343     auxint=antint->prox;
344     delete antint;
345     antint=auxint;
346   }
347 }
348 };
349 #endif
```

C.3 ϵ-DANTE

C.3.1 DANTE Class

```
1  /***************************************************************************
2                            consts.cpp  -  description
3
4      date                 : Dec 2006
5      copyright            : (C) 2006 by Pedro Cardoso
6      email                : pcardoso@ualg.pt
7   ***************************************************************************
8  Resume:
9    DANTE class
10  ***************************************************************************/
11
12 #ifndef ACOSIMUL_H
13 #define ACOSIMUL_H
14
15 #include <stdio.h>
16 #include <time.h>
17 #include "consts.cpp"
18 #include "rede.h"
19 #include "pareto.h"
20 #include "quadtree.h"
21
22
23 class DANTE {
24   REDE      *rede;
25   int       NumCiclos;    //numero de ciclos a correr
26   pareto      *ps;
27   long      solucoes;
28 public:
29   QT        *qt;
30
31
32 /*************************************************/
33 //CONSTUCTOR
34 DANTE(REDE *redein, int NumCiclos_in,  QT *qtin){
35   rede     = redein;
36   NumCiclos = NumCiclos_in;
37   ps       = new pareto(rede->NC);
38   qt       = new QT(rede);
39   solucoes  = 0;
40 }
41
```

```
42  /*****************************************************/
43  //DESCONSTUCTOR
44  ~DANTE(){
45  }
46
47  /*****************************************************/
48  //reset pheromones
49  void Reinicializa_feromonas(){
50    rede->ReinicializaFeromonas();
51  }
52
53  /*****************************************************/
54  //returns the number of nodes
55  int NumNos(){
56      return rede->NN;
57  }
58
59  /*****************************************************/
60  //RUN DANTE
61  void RUN(QT *qt, int op, TREAL Q, int opdante,TREAL maxcicletime,
62       int volta, int NumAnts,int NumCiclos){
63    long       nant=0;
64    eDANTE_MOMST  *eant;
65    MONACO_MOMST_by_singletree  *antaco;
66    long int     cont,in=0,out=0,lin,lout,inout;
67    long int     totalant=0;
68    TREAL      dummy;
69    TREAL      epsilon=0.001;
70    int        nivelmax=25,nivel2;
71    int        limpatabu=1;
72    int        deltataus=1;
73    int        outoftime=0;
74    clock_t      start,finish;
75    TREAL      tempoporformiga=maxcicletime;
76    TREAL      q0;
77    int        ramos=5;
78    int        ciclos_sem_entrar_nada=0;
79
80
81      time(&start);
82      cont=0;
83      deltataus=0;
84      solucoes=0;
85      while((cont<NumCiclos) && (!outoftime)){
86        if(_debug_ >= 1) printf(".");
87        cont++;
88        lin=0;
89        lout=0;
90        for(nant=NumAnts; (nant>0) && (!outoftime); nant--){
91          if(_debug_>1) printf("a");
92          nivelmax=1;
93                  nivel2=rede->NN/2;//rede->NN/5;//(int)sqrt((TREAL)nivelmax);
94          ramos=2;
95          epsilon=0.0;
96          q0=(TREAL)(rand()%5+5.0)/10.0;
97          deltataus=0;
98          tempoporformiga=log((TREAL)rede->NN);
99          limpatabu=1;
100         if(_debug_>2) printf("%.1f ",q0);
101         eant = new eDANTE_MOMST(rede,Q,q0,epsilon,nivelmax,nivel2,
102           limpatabu,deltataus,start,tempoporformiga,volta,ramos);
103         eant->ANT_RUN(qt,op,epsilon);
104         totalant++;
105         lin+=eant->in;
106         lout+=eant->out;
107         solucoes+=eant->in - eant->out;
108         if(eant->in){
109           ciclos_sem_entrar_nada=0;
110           time(&start);
111           if(_debug_) printf("q%.1f(%d,%d)%d ",
112             q0, eant->in, eant->out, eant->conta_entradas_no_delta);
113
114         }
115         delete eant;
116         time(&finish);
117         outoftime= (difftime(finish, start) < maxcicletime)?0:1;
118         if(_debug_ && outoftime) printf("!");
119       }//for - nant
120
121       in+=lin;
122       out+=lout;
123       if (lout || lin){
```

```
124         time(&start);
125       }else{
126         ciclos_sem_entrar_nada++;
127         if(difftime(finish, start) <maxcicletime){// 30){
128         }
129       }
130     }// while
131     if(_debug_ >= 1) printf("\n\t #P:%ld \t In:%ld\t Out:%ld",qt->contsol,in,out);
132 }
133
134 };
135 #endif
```

C.3.2 MOST_eDANTE Class

```
1  /***************************************************************************
2                            consts.cpp  -  description
3                              -------------------
4      date                 : Dec 2006
5      copyright            : (C) 2006 by Pedro Cardoso
6      email                : pcardoso@ualg.pt
7   ***************************************************************************
8  Resume:
9    eDANTE_MOMST class
10  ***************************************************************************/
11
12 #pragma once
13
14 #include "ant.h"
15
16
17 class eDANTE_MOMST :
18   public ANT
19 {
20 public:
21   int in,out;
22   TREAL Q;
23   QT    *qt;
24   int   op;
25   TREAL erro,epsilon,maior,q,q0;
26   int   nivelmax,donivel;
27   int   aindaha;//global variable for the recursive process
28   int   tabulocal[MAXN][MAXN];
29   int   lstarestas[MAXN*MAXN/2][2];
30   int   verbose;
31   int   apagouaux;
32   int   STetabulocal;
33   int   limpatabu;//clean tabu
34   int   deltataus;//solution to enter the pheromone trails?
35   TREAL r,soma, somatotal;
36   double  duracaomax;
37   clock_t start,finish;
38   int   volta;
39   int   nivel2;
40   int   ramos;
41   int   pslocal;
42   long  conta_entradas_no_delta;
43   QT    *qtaux;
44   int   graumax;
45
46
47 /*************************************************/
48 //constructor
49 eDANTE_MOMST(REDE *redein, TREAL Qin, TREAL q0in,TREAL epsilonin, int nivelmaxin, int
       nivel2in,int  limpatabuin, int deltatausin, clock_t startin, double duracaomaxin,
       int voltain,int ramosin):ANT(redein){
50   Q      = Qin;
51   q0       = q0in;
52   //counter in/out sols.
53   in       = 0;
54   out      = 0;
55   //parameters
56   epsilon    = epsilonin;
57   duracaomax  = duracaomaxin;
58   start    = startin;
59   volta    = voltain;
60   nivelmax  = nivelmaxin;
61   nivel2    = nivel2in;
62   ramos    = ramosin;
63   conta_entradas_no_delta = 0;
64   graumax    = rede->NN;
65   for(int i=0;i<=rede->NN;i++)
66     graus[i]=0;
```

```
67    verbose=0;
68    donivel=rede->NN-nivelmax;
69    limpatabu=limpatabuin;
70    deltataus=deltatausin;
71
72    }
73
74  /*********************************************/
75  //caler function
76  int ANT_RUN(QT *qt_in, int opin, TREAL epsilonin){
77    int i;
78
79    op      = opin;
80    epsilon   = epsilonin;
81    qt      = qt_in;
82    for(i=0;i<rede->NC;i++)
83      Custos[i]=0;
84    for(i=0;i<rede->NN;i++){
85      tabu[i]=0;
86    }
87    for(int i=0;i<rede->NN;i++){
88      for(int j=0; j<rede->NN; j++){
89        tabulocal[i][j]=-1;
90      }
91    }
92    time(&start);
93    pslocal=0;
94
95    ANT_RUN(0);
96
97    if(pslocal){
98      if(_debug_) printf("<<");
99      qt->Merge(qtaux,&in,&out);
100     rede->ReinicializaFeromonas();
101     qtaux->UpdateDeltaFeromonas((qt->Qminy[1]+qt->Qminx[0])/2.0);
102     rede->ActualizaFeromonas();
103     if(_debug_) printf("%d>>",qt->contsol);
104     delete qtaux;
105   }
106   return 1;
107 }
108
109 int ANT_RUN(int numarestas)
110 {
111   int idxno,noin,nosadj,s,t,apagou=0,apagoul=0,xants,apaux,procura=0;
112
113   if(numarestas==rede->NN-1){
114     conta_entradas_no_delta++;
115     if(!pslocal){
116             apagou=qt->Update(&Custos,&erro,ST);
117       if((apagou)){
118         pslocal=1;
119         qtaux=new QT(rede);
120         qtaux->Update(&Custos,&erro,ST);
121       }
122     }else{
123       apagou=qtaux->Update(&Custos,&erro,ST);
124     }
125
126     if(apagou>0){
127       in++;
128             out+=apagou-1;
129       procura=2;
130       time(&start);
131     }else{
132       if(  erro <=(Custos[0]+Custos[1]) * epsilon){
133         if(deltataus) ActualizaDeltaFeromonas(Q);
134         if(apagou==0) procura=1;
135       }
136     }
137     time(&finish);
138
139     if(duracaomax<difftime(finish, start)){
140       procura=-1;
141     }
142     return  (procura);
143
144   }
145
146
```

```
147   if(numarestas==0){//empty tree
148     q=((TREAL)rand())/((TREAL)(RAND_MAX+1.0));
149     if(q>q0){
150       idxno=rand()%(rede->NN);//choose a node
151       somatotal=rede->SomaDoProdutoDosProdutos[idxno];
152       r=(TREAL)(rand()*somatotal/(RAND_MAX+1.0));
153       soma=0.0;
154       nosadj=rede->prim_MzAdj[idxno];
155       noin=-1;
156       while(soma<=r){
157         soma +=  rede->MzAdj[idxno][nosadj]->ProdutoDosProdutos;
158         if(soma<=r){
159           nosadj=rede->MzAdj[idxno][nosadj]->prox;
160         }else{
161           noin=nosadj;
162         }
163       }
164     }else{
165       maior=-1.0;
166       for(s=0; s<rede->NN; s++){
167         t=rede->prim_MzAdj[s];
168         while(t!=-1){
169           if(rede->MzAdj[s][t]->ProdutoDosProdutos > maior){
170             idxno=s;
171             noin=t;
172             maior=rede->MzAdj[s][t]->ProdutoDosProdutos;
173           }
174           t=rede->MzAdj[s][t]->prox;
175         }
176
177       }
178     }
179     ADD_ARESTA(idxno,noin/*s0*/);
180     tabu[idxno]++;
181     tabu[noin]++;
182     tabulocal[idxno][noin]=numarestas;
183     tabulocal[noin][idxno]=numarestas;
184     while(ANT_RUN(numarestas+1));
185     return 0; //search ends
186   }else{
187     for(xants =  1 ; xants > 0 ; xants--){
188       q=((TREAL)rand())/((TREAL)(RAND_MAX+1.0));
189       if(q>q0){
190         somatotal=0.0;
191         aindaha=0;
192         for(s=0; s<rede->NN; s++){
193           if((tabu[s]>0)&&(tabu[s]< graumax)){
194             t=rede->prim_MzAdj[s];
195             while(t!=-1){
196               if(tabu[t]==0){
197                 if(tabulocal[s][t]==-1){
198                   somatotal+=rede->MzAdj[s][t]->ProdutoDosProdutos;
199                   lstarestas[aindaha][0]=s;
200                   lstarestas[aindaha][1]=t;
201                   aindaha++;
202                 }
203               }
204               t=rede->MzAdj[s][t]->prox;
205             }
206           }
207         }//for s
208         if(!aindaha){
209           if(limpatabu){
210             for(int i=0; i<rede->NN; i++)
211               for(int j=0; j<rede->NN; j++){
212                 if(tabulocal[i][j]==numarestas){
213                   tabulocal[i][j]=-1;
214                   tabulocal[j][i]=-1;
215                 }
216               }
217           }
218           return procura;
219         }
220         r=(TREAL)(rand()*somatotal/(RAND_MAX+1.0));
221         soma=0.0;
222         noin=-1;
223         s=0;
```

```
224           while(noin==-1){
225             aindaha--;
226             s=lstarestas[aindaha][0];
227             t=lstarestas[aindaha][1];
228               soma+=rede->MzAdj[s][t]->ProdutoDosProdutos;
229             if(soma>r){
230               idxno=s;
231               noin=t;
232             }
233           }
234         }
235         else{//q<=q0
236           maior=-1.0;
237           for(s=0;s<rede->NN;s++){
238             if((tabu[s]>0)&&(tabu[s]<graumax)){
239               t=rede->prim_MzAdj[s];
240               while(t!=-1){
241                 if(tabu[t]==0){
242                   if(tabulocal[s][t]==-1){
243                     if(rede->MzAdj[s][t]->ProdutoDosProdutos > maior){
244                       idxno=s;
245                       noin=t;
246                       maior=rede->MzAdj[s][t]->ProdutoDosProdutos;
247                     }
248                   }
249                 }
250                 t=rede->MzAdj[s][t]->prox;
251               }
252             }//if
253           }//for
254           if(maior<0){
255             if(limpatabu){
256               for(int i=0; i<rede->NN; i++)
257                 for(int j=0; j<rede->NN; j++){
258                   if(tabulocal[i][j]==numarestas){
259                     tabulocal[i][j]=-1;
260                     tabulocal[j][i]=-1;
261                   }
262                 }
263             }
264             return procura;
265
266           }
267           s=idxno;
268           t=noin;
269         }
270         ADD_ARESTA(s,t);
271         tabulocal[s][t]=numarestas;
272         tabulocal[t][s]=numarestas;
273         tabu[t]++;
274         tabu[s]++;
275         if(verbose) escreve(numarestas);
276         apaux = ANT_RUN(numarestas+1);
277         if(apaux<0){
278           return apaux;
279         }
280
281         if(apaux>0){
282           procura=(procura < apaux) ? apaux : procura;
283           if(procura==2){
284             xants += (numarestas > nivelmax)? ramos: 0;
285           }else{
286             if(procura==1){
287               xants += (numarestas > nivel2)? ramos: 0;
288             }
289           }
290         }
291         REMOVE_ARESTA(s,t);
292         tabu[t]--;
293         tabu[s]--;
294       }//for
295       if(limpatabu){
296         for(int i=0; i<rede->NN; i++)
297           for(int j=0; j<rede->NN; j++){
298             if(tabulocal[i][j]==numarestas){
299               tabulocal[i][j]=-1;
300               tabulocal[j][i]=-1;
```

```
301           }
302         }
303     }
304     return procura;
305   }//else
306 }
307 };
```

C.4 Brute Force

C.4.1 AllTREES Class

```
1  /***************************************************************************
2                           alltrees.h  -  description
3
4      date                 : Dec 2006
5      copyright            : (C) 2006 by Pedro Cardoso
6      email                : pcardoso@ualg.pt
7   ***************************************************************************
8  Resume:
9    Brute force method to compute all trees
10  ***************************************************************************/
11
12 #pragma once
13 #include "rede.h"
14 #include "PARETOSET.h"
15 #include <time.h>
16
17 typedef int VECTOR[N];
18
19 class ALLTREES
20 {
21 public:
22   REDE *r;
23   PARETOSET *ps;
24   int arvore[N];
25   ARRAY_CUSTOS custos;
26   int n;
27   int num_arv,mostraacada;
28   int cont,inttest;
29   FILE *f;
30   double start,start0;
31   int aux[N][N];
32   VECTOR tabu;
33
34
35 /*****************************************************/
36 //constructor
37 ALLTREES(REDE *r_in,PARETOSET *ps_in,FILE *f_in,double s)
38   {
39     r=r_in;
40     ps=ps_in;
41     n=r->nn;
42     num_arv=0;
43     cont=0;
44     f=f_in;
45     start=s;
46     start0=s;
47     inttest=10;
48     mostraacada=1;
49   }
50
51
52 /*****************************************************/
53 //destructor
54 ~ALLTREES(void)
55   {
56   }
57
58 /*****************************************************/
59 //
60 void ARVORES(){
61     int idx=1;
62
63     for(int j=1;j<n;j++)
64       tabu[j]=-1;
65
66     for(j=0; j<r->nc;j++)
67       custos[j]=0;
68
```

```
69      ARVORES(0,0);//Starts
70    }
71
72    void ARVORES(int i,int idx){
73      int j;
74
75      if(tabu[i]<0)
76        tabu[i]=idx;
77
78      REDE::LISTA_ADJACENCIA *p;
79      p=r->MzAdj[i];
80      while(p){
81        j=p->no;
82        //j not in ST
83        if(tabu[j]<0){
84          //adds edge i-j
85          for(int k=0; k<r->nc; k++)
86            custos[k]+=p->VctCustos[k];
87          arvore[i]=j;
88          if(i==n-2){
89            num_arv++;
90            if(num_arv==mostraacada){
91              num_arv=0;
92              PRINT_ARVORE();
93            }
94            ps->ActualizaPS(&custos);
95          }else{
96            tabu[j]=tabu[i];
97            ARVORES(i+1,idx+1);
98            tabu[j]=-1;
99          }
100         //remoce edge i-j
101         for(int k=0; k<r->nc; k++)
102           custos[k]-=p->VctCustos[k];
103       }else{// j is in ST
104         if((tabu[j]!=tabu[i])){//continue
105           //add edge i-j
106           for(int k =0;k<r->nc;k++)
107             custos[k]+=p->VctCustos[k];
108
109           arvore[i]=j;
110           if(i==n-2){
111             num_arv++;
112             ps->ActualizaPS(&custos);
113             if(num_arv==mostraacada){
114               num_arv=0;
115               PRINT_ARVORE();
116             }
117           }else{
118             int tabuj=tabu[j];
119             for(int l=0;l<n;l++){
120               if(tabu[l]==tabuj){
121                 aux[idx][l]=1;
122                 tabu[l]=tabu[i];
123               }else{
124                 aux[idx][l]=-1;
125               }
126             }
127             ARVORES(i+1,idx+1);
128             for(int l=0;l<n;l++){
129               if(aux[idx][l]==1){
130                 tabu[l]=tabuj;
131               }
132             }
133
134           }
135           //remove edge i-j
136           for(int k=0;k<r->nc;k++)
137             custos[k]-=p->VctCustos[k];
138         }
139       }
140       p=p->prox;
141     }//while
142   }
143
144
145
```

```
146 /*****************************************************/
147 //
148 void ARVORES(int i,int idx,VECTOR tabu){
149     int aux[N],j;
150 
151     if(tabu[i]<0)
152       tabu[i]=idx;
153 
154     REDE::LISTA_ADJACENCIA *p;
155     p=r->MzAdj[i];
156     while(p){
157       j=p->no;
158       if(tabu[j]<0){
159         for(int k=0; k<r->nc; k++)
160           custos[k]+=p->VctCustos[k];
161         if(i==n-2){
162           arvore[i]=j;
163           num_arv++;
164           ps->ActualizaPS(&custos);
165         }else{
166           arvore[i]=j;
167           tabu[j]=tabu[i];
168           ARVORES(i+1,idx+1,tabu);
169           tabu[j]=-1;
170         }
171         for(int k =0;k<r->nc;k++)
172           custos[k]-=p->VctCustos[k];
173       }else{
174         if((tabu[j]!=tabu[i])){
175           for(int k =0;k<r->nc;k++)
176             custos[k]+=p->VctCustos[k];
177           if(i==n-2){
178             arvore[i]=j;
179             num_arv++;
180             ps->ActualizaPS(&custos);
181           }else{
182             arvore[i]=j;
183             int tabuj=tabu[j];
184             for(int l=0;l<n;l++){
185               if(tabu[l]==tabuj){
186                 aux[l]=1;
187                 tabu[l]=tabu[i];
188               }else{
189                 aux[l]=-1;
190               }
191             }
192             ARVORES(i+1,idx+1,tabu);
193             for(int l=0;l<n;l++){
194               if(aux[l]==1){
195                 tabu[l]=tabuj;
196               }
197             }
198 
199           }
200           for(int k=0;k<r->nc;k++)
201             custos[k]-=p->VctCustos[k];
202         }
203       }
204       p=p->prox;
205     }//while
206   }
207 
208 
209 
210 /*****************************************************/
211 //print tree
212 void PRINT_ARVORE(){
213         for(int i=0;i<n-1;i++)
214       fprintf(f,"%d ",arvore[i]);
215     fprintf(f," -- ");
216 
217     for(int i=0;i<r->nc;i++)
218       fprintf(f,"%d ",custos[i]);
219     fprintf(f,"\n");
220   }
221 };
```

Index