U.PORTO
FEP FACULDADE DE ECONOMIA UNIVERSIDADE DO PORTO

Ant Colony Optimization Algorithms to solve Nonlinear Network Flow Problems

por

Marta Sofia Rodrigues Monteiro

Tese de Doutoramento em Ciências Empresariais

Orientada por:
Prof. Doutora Dalila B. M. M. Fontes
Prof. Doutor Fernando A. C. C. Fontes

2012

Biography Note

Marta Monteiro was born and raised in Porto. In 1998 she finishes her graduation in Applied Mathematics to Technology in the Faculty of Sciences of the University of Porto.

After a brief incursion in a business job, in 2001 she decides to apply to a research grant in Image Processing at the Faculty of Engineering of the University of Porto.

This experience was so rewarding that in 2002 she decides to improve her scientific knowledge by attending a two years Masters Degree in Data Analysis and Decision Support Systems at the Faculty of Economics of the University of Porto.

Meanwhile she also teaches during two years at the University of Minho, at the Mathematics for Science and Technology Department.

Having finished her Master's Degree she applies and is accepted to a Doctoral Programme in Business and Management Studies at the Faculty of Economics of the University of Porto.

She has been a full time data analyst at "Confidencial Imobiliário" for the last four years and, at the same time, a teacher at the Portuguese School of Bank Management.

Acknowledgements

When we think about all the years that have passed by while we were walking this long road, we come to the conclusion that almost everyone that surrounded us was a helping vehicle for this long journey. Therefore, there is no “first of all” and “second of all”, because all people contributed in the best way they knew and could. Nonetheless, some must be mentioned for their utmost effort.

An enormous “thank you” is due to my supervisors, Prof. Dalila Fontes and Prof. Fernando Fontes, but with a special affection for Prof. Dalila Fontes. Without her guidance and word of advice I could not have made it. So many times I was down and tired but she always managed to be there for me, in both professional and private moments. More than a supervisor I consider her as a dear friend.

On a more personal level, my thanks go to my husband Miguel and to my parents Blandina and António for their sacrifice. To Miguel I want to thank his patience for a wife not always one hundred percent present. His words of advice, from one who has already travelled this road, and for taking care of our daughter providing me with the necessary time to work. My mother and father were exceptional because lately they took care of all my day-life issues so that I could accelerate this process. Rute, my mother-in-law is also to be thanked for also taking care of Alice, which I hopefully believe was not such a big “sacrifice” for her.

Finally, an infinite thank you to my lovely daughter Alice that was born during this journey. She was the one who made the utmost sacrifice because during the months preceding the finish of this work, she had to stay away from home to let her mother work during weekends. I do hope she will forgive me and be proud of her mother one day.

This thesis is dedicated to
Miguel and Alice

Resumo

Este trabalho aborda dois problemas de optimização em redes de fluxos: a minimização de custos em redes com uma só fonte e sem restrições na capacidade dos arcos (SSU MCNFP) e a minimização de custos em redes com restrições no número de arcos em cada caminho definido entre a fonte e os nós procura (HMFST). De salientar que para ambos os problemas são consideradas funções de custo não lineares côncavas.

Os modelos matemáticos obtidos podem ser solucionados por software específico, como por exemplo o software de optimização CPLEX, mas apenas para algumas das funções de custo não lineares consideradas. Para além disso, dado que a dimensão do modelo cresce rapidamente com a dimensão do problema, o tempo computacional torna-se proibitivo. Propõem-se então duas heurísticas, HACO1 e HACO2, baseadas em optimização por colónias de formigas hibridizadas com pesquisa local, como forma de solucionar os problemas apontados realçando-se ainda o estudo efectuado à sensibilidade das heurísticas relativamente à variação dos seus parâmetros.

As heurísticas desenvolvidas são avaliadas num conjunto de problemas de referência que estão disponíveis na internet. Os resultados computacionais obtidos permitem mostrar a eficiência das heurísticas propostas tanto para problemas de tamanho pequeno como de tamanho grande.

Em particular, a heurística HACO1 conseguiu melhorar resultados obtidos previamente com uma heurística evolutiva tanto no que diz respeito à qualidade da solução como aos tempos computacionais, estes últimos à custa da análise de um número reduzido de soluções comparativamente com outros métodos. A heurística HACO2 conseguiu obter a solução óptima em mais de 75% das vezes em que cada problema foi resolvido, tendo também conseguido melhorar os resultados reportados na literatura e obtidos com uma heurística baseada em populações de soluções. Os tempos computacionais também foram reduzidos tanto em comparação com os tempos obtidos pelo CPLEX como com resultados reportados na literatura.

Abstract

In this work, we address two nonlinear network flow problems: the Single Source Uncapacitated Minimum Cost Network Flow Problem (SSU MCNFP) and the Hop-Constrained Minimum cost Flow Spanning Tree (HMFST) problem, both with concave cost functions.

We propose two hybrid heuristics, HACO1 and HACO2, which are based on Ant Colony Optimization (ACO) and on Local Search (LS), to, respectively, solve SSU MCNFPs and HMFST problems. Heuristics are proposed due to their combinatorial nature and also to the fact that the total costs are nonlinear concave with a fixed-charge component.

In order to test the performance of our algorithms we solve a set of benchmark problems available online and compare the results obtained with other population based heuristic methods recently published (HGA and MPGA) and also with a commercial available software CPLEX.

HACO1 and HACO2 have proven to be very efficient while solving both small and large size problem instances. HACO1 was able to improve further upon some results of HGA, both in terms of computational time and solution quality, proving that ACO based algorithms are a very good alternative approach to solve these problems, with the great advantage of reducing the computational effort by analysing a smaller number of solutions.

The results obtained with HACO2 were compared with the ones obtained and reported in literature for MPGA. Our algorithm proved to be able to find an optimum in more than 75% of the runs, for each problem instance solved, and was also able to improve on many of the results reported in the literature. Furthermore, for every single problem instance used we were always able to find a feasible solution, which was not the case for the other heuristic. Our algorithm improved upon the computational time needed by CPLEX and was always lower than that of MPGA.

Contents

List of Figures

List of Tables

List of Algorithms

List of Models

Summary of Notation

Herein we introduce a list of the most commonly used notation and abbreviations and their respective interpretation.

$\mathbb{Z}$	:	The set of integer numbers, $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
S	:	Any solution
$N(S)$	:	The set of solutions in the neighbourhood of S
S'	:	A solution in the neighbourhood of solution S
S^i	:	The best solution found at iteration i
S^g	:	The best solution found from the beginning of the algorithm, also known as the incumbent solution
$F(\ldots)$ or $G(\ldots)$	:	Cost function
MCNFP	:	Minimum Cost Network Flow Problem
SSU	:	Single Source Uncapacitated
LS	:	Local Search
AS	:	Ant System
ACO	:	Ant Colony Optimization
HACO	:	Hybrid Ant Colony Optimization
HMST	:	Hop-constrained Minimum Spanning Tree
HMFST	:	Hop-constrained Minimum cost Flow Spanning Tree
GA	:	Genetic Algorithm
MPGA	:	Multi-Population Genetic Algorithm
DP	:	Dynamic Programming
TS	:	Tabu Search
UB	:	Upper Bound
LB	:	Lower Bound

1

Introduction

Network is a word often used in our daily life within different fields, whether we are speaking of a social network such as *Facebook* or *LinkedIn*, at work to captivate a customer by referring to the network of customers already working with our company, or when we are visiting another country, and we want to identify the transportation network in order to choose the most adequate means of transport to be used during our stay. Their vital importance is unquestionable, specially when associated with flows, becoming then very interesting from the economical point of view as they can be applied to model many practical situations. Therefore, it does not come as a surprise that network flow problems are one of the largest classes within the Combinatorial Optimization field.

Let us assume that we own a manufacturing company dedicated to the production of a certain kind of dolls. The dolls are produced in a small factory located somewhere in the country. Now, let us suppose it is Christmas time and this specific doll is one of the most popular items demanded by little girls and that several toy stores have placed an order. If we want to establish a distribution network to deliver the dolls, it is obvious that to service the different stores we will incur in different costs. Also, we do not expect them to be only dependent on the distance between our factory and the toy stores because we may have to travel trough local roads, highways, by boat or even by airplane, all of them representing different costs. For example, the necessity to use a highway implies a toll

charge to be paid both ways. This cost will also have to be accounted for in the overall cost of the delivery, but it will represent a decreasing per unit cost with the increase of the number of transported dolls, a concave cost function. Other situations where the costs are nonlinear arise due to economies of scale. Economies of scale are referred to in microeconomics in association to the benefits over costs due to the expansion of an enterprise. Several factors associated with the expansion may lead to savings in terms of cost per unit with the increase on the production. Returning to the example of the dolls factory, the acquisition cost of the production machines must be accounted for in the price of every doll being produced in it. Therefore, if there is an increase on the number of dolls being produced the unit production cost to be added to the dolls' selling price will decrease, again a concave cost function.

Network problems considering costs with these characteristics can provide us with very interesting optimization problems mainly due to the practical applications that can benefit from the results.

1.1. Motivation

It is not always easy to decide on a subject to investigate for a thesis. Sometimes, all we know is that we want to develop a scientific research and to see our work and effort recognized by our peers. The area of optimization is a vast one, with many different and interesting branches making it difficult sometimes to decide. Following a previous work on a network flow problem (Monteiro, 2005), the decision to continue to explore this class of problems was taken.

The main reason for us to study the Single Source Uncapacitated Minimum Cost Network Flow Problem (SSU MCNFP) and the Hop-constrained Minimum cost Flow Spanning Tree problem (HMFST) is their versatility to model practical network design problems. The SSU MCNFP is often used to model applications in logistics, supply chains, and transportation and the HMFST problem is specially used to model telecommunication networks, for example multicast networks, where reliability is an issue, since often some links of the network fail. The latter one is an area in constant technological development but always dependent on what goes on with the physical part of the network, such as computers or cables, this way justifying the need to impose constraints to lessen the effects of network disruptions.

Furthermore, we consider an extra characteristic which makes them more appealing to practical applications: nonlinear and concave costs. These costs are usually related to economies of scale, for example when a long time supplying contract implies a reduction

on the average unit price of a product, and appear very often in practice specially, but not limited to, in the areas already mentioned.

The choice to use an approximate method has been motivated by an increasing interest in recent metaheuristics, such as Ant Colony Optimization (ACO), swarm optimization, or Genetic Algorithms (GAs). ACO, in particular, seems very interesting because it is fairly recent, very good results have been reported in many areas of optimization, and in particular because of its nature inspired structure. It is somehow very appealing to have a set of artificial ants solving combinatorial optimization problems.

Local search is also a very useful approximate method that, together with a good initial solution, can provide good local solutions. Therefore, local search is usually hybridized with these types of metaheuristics in order to improve their performance, by introducing an intensification on the search nearby the good solutions found by the aforementioned methods. Many practical and academical optimization problems have been solved by hybrid algorithms and the results found are amongst the best ones to the moment (Talbi, 2002).

ACO algorithms were initially developed to solve hard combinatorial optimization problems (Dorigo and Stützle, 2004; Dorigo and Blum, 2005), and were firstly applied to solve the well-known and NP-hard Travelling Salesman Problem (TSP), where the shortest circular route between a given set of nodes is to be defined, without visiting twice the same node. Furthermore, ACO algorithms have also been successfully applied to solve flow problems with concave cost functions, such as the transportation problem (Altiparmak and Karaoglan, 2007). The problems we want to solve in this work join together these characteristics, among others: concave costs, shortest paths, and flow between pairs of nodes.

All things considered, we have elected the ant colony optimization class of algorithms as the main solution method, and the reasons for it are twofold.

Firstly, after reading several research papers describing these problems, we came to the conclusion that an heuristic method was the most adequate one, due to the combinatorial nature of the problems to be solved.

Secondly, the application of an ACO heuristic has never before been applied to these specific problems, as far as we are aware of. Therefore, we can add another class of combinatorial optimization problems to the ones that have already been tackled with ACO algorithms. However, network flow problems have already been solved with genetic algorithms with very good results (Fontes and Gonçalves, 2007; Hajiaghaei-Keshteli et al, 2010), and GAs are known to have very good performance in solving combinatorial opti-

mization problems regarding both solution values and computational effort. Nevertheless, several optimization problems have been solved by ACO algorithms with improved results when compared with other heuristics, such as GAs, among others, see e.g. (Bui and Zrncic, 2006; Yin and Wang, 2006; Bin et al, 2009; Faria et al, 2006; Putha et al, 2012). Therefore, even before implementing our algorithms, we expected that ACO would have a competitive performance, in comparison with other heuristic methods already used to solve these problems. As we will see, our expectations were confirmed.

1.2. Aims and Scope

We have set two main objectives to achieve with this thesis.

On the one hand, we want to develop a study around two network flow problems that consider nonlinear costs. In these problems, we have to decide on the minimum cost layout of a network such that all demand is satisfied. One such problem is the single source uncapacitated minimum cost network flow problem and the other is the hop-constrained minimum cost flow spanning tree problem. Although these problems have been fairly studied (in the latter case, the HMSTP, at least has), the ones considering concave and nonlinear costs however have not. These problems are considered difficult to solve mainly because they deal with the minimization of a concave function over a convex feasible region. Therefore, there is nothing to guarantee that a local optimum is a global optimum. Given the combinatorial nature of the aforementioned problems, exact methods can be very expensive in terms of computational effort, even for small sized problems, as they tend to enumerate vertices of the convex polyhedron defined by the network constraints. Heuristic methods can easily overcome this problem and they have become very popular in recent years, although they may provide only a local optimum.

On the other hand, we propose for the first time, as far as we are aware of, ant colony optimization algorithms to solve both single source uncapacitated minimum cost network flow problems and hop-constrained minimum cost flow spanning tree problems. Furthermore, only a few works consider concave cost functions made of nonlinear concave routing costs and fixed costs simultaneously, which are those of (Burkard et al, 2001; Fontes et al, 2003, 2006b,a; Fontes and Gonçalves, 2007; Fontes, 2010), and more recently (Dang et al, 2011; Fontes and Gonçalves, 2012). We also want to perform a study on the sensibility of the algorithm to changes on the ACO parameter values and their influence over the performance of the algorithm on the solution quality.

1.3. Contribution

In this work we develop algorithms based on the ant colony optimization metaheuristic. Such an heuristic has never been used to address the singles source uncapacitated minimum concave cost network flow problem or the hop-constrained minimum cost spanning tree problem with flows.

We also provide a study on the sensitivity of the algorithms to the variation of the ACO parameters values, in order to infer its influence on the solution quality. The ACO algorithms that were developed are coupled with a local search heuristic in order to further improve the results obtained.

Our algorithms proved to be able to find an optimum solution in more than 75% of the runs, for each problem instance solved. Furthermore, we improve upon the results obtained with two genetic algorithms: one of the algorithms is a multi population GA, that evaluates up to 80% more solutions when compared with the number of solutions evaluated with our ACO based algorithm; the other GA is based on a recently proposed framework to solve combinatorial optimization problems with genetic algorithms using biased random-keys representation, with several successful applications of this method reported on literature.

The improvement is obtained by reducing the time spent to run the algorithm, which is achieved due to the reduction in the number of solutions that are evaluated, and also by further improving the gap between our results and some of the best solutions found so far.

A working paper discussing and reviewing network flow problems, based on Chapters 2 and 4, has been published. Marta S.R. Monteiro, Dalila B.M.M. Fontes, Fernando A.C.C. Fontes (2012) Solving Concave Network Flow Problems, Working Paper no. 475, Faculdade de Economia, Universidade do Porto.

A literature survey, based on Chapter 3, has been published as a working paper. Marta S.R. Monteiro, Dalila B.M.M. Fontes, Fernando A.C.C. Fontes (2012) Ant Colony Optimization: a literature survey, Working Paper no. 474, Faculdade de Economia, Universidade do Porto.

A preliminary version of the work presented in Chapter 4 has been published as a full article in the proceedings of an international conference. Marta S.R. Monteiro, Dalila B.M.M. Fontes, Fernando A.C.C. Fontes (2011) An Ant Colony Optimization Algorithm to solve the Minimum Cost Network Flow Problem with Concave Cost Functions, In: Krasnogor N, Lanzi PL (eds) GECCO, ACM, pp 139-146.

Furthermore, a full version of the results presented in Chapter 4 has been published. Marta S.R. Monteiro, Dalila B.M.M. Fontes, Fernando A.C.C. Fontes (2012) Concave minimum cost network flow problems solved with a colony of ants, Journal of Heuristics DOI 10.1007/s10732-012-9214-6.

A version of Chapter 5 has been submitted for publication in an international journal.

1.4. Thesis Outline

This work is organized as follows. Chapter 2 introduces Network Flow Problems (NFPs). Issues such as classification of NFPs, problem complexity and nonlinear programming are discussed. Furthermore, we also provide a brief introduction on the mechanism of some of the most popular metaheuristics used to solve combinatorial problems.

In Chapter 3, we review the ant colony optimization metaheuristic. We start by showing its origin and its principles and then describe the Ant System (AS), which was the first ant colony algorithm to be developed and to be applied to the solution of the travelling salesman problem, a well-known NP-hard problem. The final sections of the chapter are dedicated to the literature review which is performed based on the identification of the building blocks of Ant Colony Optimization algorithms.

While the first three chapters are introductory and provide a summary of the state-of-the-art, Chapters 4 and 5 concentrate the novel contributions.

Chapter 4 follows. We start by introducing the mathematical programming formulation of the concave minimum cost network flow problem and of the modifications necessary to address the single source uncapacitated concave minimum cost network flow problem. A literature review is given, and this is made from two perspectives: exact solution methods and approximate and heuristic solution techniques. The new method based in ant colony optimization, never before used to solve these problems, as far as we know of, is next explained and discussed. This algorithm is hybridized with a local search procedure, in order to perform exploitation. The sensibility of the performance of the ACO algorithm to different parameter values is also discussed. Finally, we provide an account of the results obtained and compare them with the results obtained with the use of other population based heuristic techniques.

In Chapter 5, we approach a fairly recent problem which is the hop-constrained minimum cost flow spanning tree problem. The problem is defined and its importance in solving many practical problems is discussed. A mathematical programming formulation for this problem is also given. A literature review is presented being mainly focused on

the nearest problem which is the hop-constraint minimum spanning tree problem, given the recent nature of the problem.

Finally, Chapter 6 summarizes our contributions and draws conclusions on what has been done and achieved with this thesis. Future work is also discussed.

2

Network Flow Problems

Generally speaking, a network flow problem can be described as the problem of minimizing supply costs while satisfying demands. Network flow problems arise frequently in many everyday life situations such as in transportation, communications, and distribution networks and several such applications are already reported in literature: a large Swedish hydrothermal power system short term operation planning is optimized based on concepts of nonlinear network flows in (Brannlund et al, 1986); in (Barros et al, 1998) a sand recycling problem is addressed by modelling it as a location problem; and Shangyao and Chung-Gee (1997) use network flow models to minimize the schedule-perturbated time after forced closures of airports with an application to a Taiwan airline operations, just to mention but a few.

But, while the former networks may be obvious, other examples of application of networks arise which are not. For example, Yeh et al (2012) use a network flow approach to predict drug targets on prostate cancer. Cancer gene networks are made up clusters of gene sub-networks. In these sub-networks there is almost always one center gene, the so-called hub, with many links to other cancer genes. The identification of these hubs to be drug targets allows to disrupt a large number of paths thus helping the eradication of sub-networks of cancer genes. Borgatti (2005) considers several types of traffic that can be flowing in a social network, such as used goods, money, gossip, e-mail, attitudes,

infection and packages in order to study centrality. The model is afterwards applied on a data set on marriages between Renaissance families in Florence.

Many versions of network flow problems exist which can be differentiated by the different constraints and/or characteristics associated to each one of them. Therefore, the interest in solving network flow problems is high.

In this work, we will focus in minimum cost network flow problems. The class of minimum cost network flow problems includes a wide range of combinatorial optimization problems such as the shortest path problem, the assignment problem, the transportation problem, and the max-flow problem, each of which has its own special cases.

Roughly speaking, in this problem the objective is to distribute some commodity from the source nodes to the demand nodes while minimizing the total cost incurred. It is easy to see that many applications exist for the MCNFP, for instance in supply chains, logistics, production planning, communications, facility location and transportation, just to mention but a few (Geunes and Pardalos, 2005; Ahuja et al, 1995).

Regarding the costs to be accounted for in MCNFPs, they can have a linear or a nonlinear nature, where nonlinear functions can be concave, convex or general nonlinear. MCNFPs with linear and convex cost function are well studied in literature and they are considered easy to solve, whereas concave and general nonlinear cost functions can represent a challenge even for small size problem instances.

Furthermore, several optimization problems can also be modelled as a transportation problem thus allowing the application of the solution methodologies already developed to solve it. travelling salesman problems, assignment problems and shortest path problems are some of those classes.

2.1. Terminology

The language used in a research area is very important to the ones interested in it. If different persons use the same word with different meanings, then knowledge cannot evolve/be shared. In this section, we introduce some pertinent terminology so that there is no ambiguity when someone is reading it.

Graphs are very close to network flows, therefore they are globally used to illustrate the functioning of the latter and it is the graph theory language that we will define here.

Generally speaking, a *graph* is an ordered pair $G = (N, A)$, where N is a set whose elements are called *nodes* (also known as *vertices*), and A is a set of pairs of nodes, i.e.

$A = \{(n_i, n_j) : n_i \in N, n_j \in N\}$, which are called *arcs* (also known as *edges*). Two nodes are *adjacent* if there is an arc in A linking them.

A graph is *undirected* if no direction is associated to an arc, i.e. $(n_i, n_j) = (n_j, n_i)$ otherwise, if $(n_i, n_j) \neq (n_j, n_i)$ the graph is *directed*, and is usually called a *digraph*. Our work will deal with *directed graphs*. In a digraph, arc $(n_i, n_j) \in N$ is said to be an arc from n_i to n_j. Note that, in a digraph, we can not assume that if arc (n_i, n_j) exists arc (n_j, n_i) also exists. If we compare this with a street the analogy is immediate: there are one way streets and two way streets. Given an undirected graph, the number of arcs incident to a node i, i.e. arcs of the form (n_i, n_j) or (n_j, n_i), represents the *degree* of node i. An arc starting and ending at the same node, (n_i, n_i) is a *loop*. In our work these loop arcs will not be considered because they are meaningless in the networks that will be studied.

Another important definition used in graph theory is the definition of *path*. A path is an ordered set of nodes $(n_1 n_2 n_3 ... n_k)$ linking two nodes, in this case node n_1 and node n_k. Each pair of adjacent nodes in the path represents an arc in the graph, for example, $n_2 n_3$ represents arc (n_2, n_3). A path starting and ending at the same node is called a *cycle*. If there is a path connecting any two distinct nodes, then the graph is said to be *connected*.

Sometimes, specific configurations of graphs are needed. A *tree*, also known as *arborescence*, i.e. a connected graph in which, for a given node known as *root* and any other node i there is only one possible directed path, is one such case.

A graph is *weighted* if a weight (number) is associated to each arc. Weights can be seen as costs or arc lengths, for example. This characteristic is very important, because it is a bridge between graph theory and network flow problems. For more information on graph theory, we refer the reader to the books by Wiitala (1987) and by Wilson (1985).

The analogy between graphs and network flow problems is very straightforward but first we need to formalize the definition of a network flow problem. We will use the definition given in (Bertsekas, 1998),

> "Loosely speaking, network flow problems consist of supply and demand points, together with several routes that connect these points and are used to transfer the supply to the demand."

Therefore, a network can be seen as a graph where commodities flow between nodes. Mathematically speaking, a flow is represented by a number indicating the amount of commodity passing through an arc. Therefore, we usually refer to the *flow on an arc* instead of the flow incoming or outgoing from a node. Nodes can be classified into three

categories. *Source* nodes can be compared to warehouses where the commodities are ready to be shipped away, in an outgoing flow. Source nodes have no incoming flow. If the amount of flow entering a node is smaller than the amount of flow leaving that same node, then the node is said to be a *demand* node. Whenever the amount of flow entering the demand node equals the demand quantity, then there is no flow leaving the demand node and the node is also known as a *sink* node. Finally, *transshipment* nodes are intermediate nodes between source and demand nodes. Flow must pass through them in order to be distributed through the network, but the amount of flow entering the node and leaving the node is always the same. To travel between nodes, flow must use arcs.

2.2. Classification of Network Flow Problems

Network flow problems comprise several classes of problems from which we can identify minimum cost flow problems, network flow problems with convex cost, multicommodity flow problems or discrete network optimization problems. It is important to notice though that there is no unique way to classify problems and, accordingly to the objective of the investigation at hand, we can always find the one most suitable to our purpose. In this work, we want to give a general idea of network flow problems, however we are more interested in a single class, the class of minimum cost network flow problems. The MCNFP can be defined as the problem of finding a set of arc flows that minimize a linear cost function, provided that they satisfy the demand of each node in the path, and that flows do not violate some given bounds. The MCNFP can be formally written as follows:

d_i - demand of node i
x_{ij} - flow on arc (i, j)
c_{ij} - cost of arc (i, j)
U_{ij} - upper limit on flow through arc (i, j)
L_{ij} - lower limit on flow through arc (i, j)

$$\text{min:} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.1}$$

$$\text{s.t.:} \quad \sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = d_i, \forall_{i\in N}, \tag{2.2}$$

$$L_{ij} \leq x_{ij} \leq U_{ij}, \quad \forall_{(i,j)\in A}, \tag{2.3}$$

where equation (2.1) represents the flow dependent linear cost to be minimized. Constraints (2.2) are known as the *flow conservation constraints*. The first term of these constraints, $\sum_{\{j|(i,j)\in A\}} x_{ij}$, represents the flow coming out of node i whereas the second term of the equation, $\sum_{\{j|(j,i)\in A\}} x_{ji}$, represents the flow entering node i. Therefore, the flow conservation constraints state that the difference between the flow going into a node and the flow coming out from a node must be equal to the demand of the node. Constraints (2.3), are called the *capacity constraints* and, as the name indicates, state the flow capacity of each arc. Without loss of generality, it is generally assumed that the costs c_{ij} are nonnegative.

There are many problems, extensions, and special cases of problems belonging to the MCNFP class, from which we give a few examples:

Shortest Path Problem - if we consider the cost of an arc as the length of the arc, we can say that this problem is to find a path between a pair of nodes in a graph (network) such that the sum of the costs of the arcs is minimized. For example, this problem may be used to model data communication networks where costs stand for average delay times of travelling through a link (arc). In this case the objective would be to minimize the sum of average delays incurred to travel between a given origin and destination. Another example can be to determine the shortest path between two points in a map, an option of the *in vogue* GPS terminal devices. In Fig. 2.1 we provide a small example where the shortest path between O and D is found, given the set of possible ways.

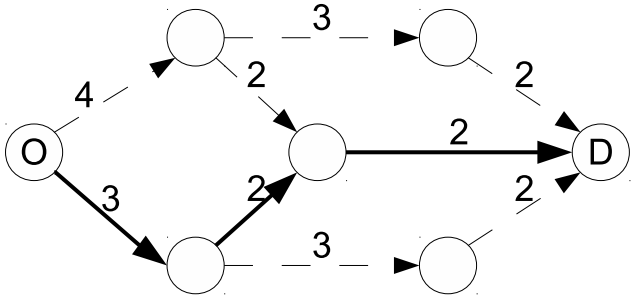

Fig. 2.1: A shortest path problem and its solution (shown in bold), where O identifies the origin node and D the destination node, and the numbers next to the arcs represent the length of the arc.

Transportation Problem - in a transportation problem we have a set of sources and a set of demand nodes. The objective is to satisfy all demand nodes at the lowest cost provided that the capacity of the sources is not exceeded. This problem has many applications in practice, from distribution and scheduling to location, (Adlakha and

Kowalski, 2003). Fig. 2.2 represents an hypothetical solution for a transportation problem with three sources, that could represent warehouses, and four demand nodes, representing retail shops, for example. We have omitted the capacities of the arcs to simplify the representation.

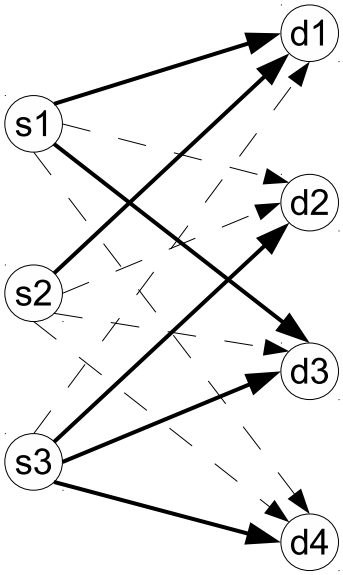

Fig. 2.2: A transportation problem and its solution (shown in bold), where sources are represented to the left and demand nodes to the right.

Assignment Problem - let us suppose we have a set of objects that we want to assign to a set of persons, provided that each object is only assigned to a single person, and vice-versa, and each assignment (object, person) may incur into a different cost. Then, the assignment problem is the one of finding the least cost assignment of all objects to all persons. The assignment problem is a special case of the transportation problem, but in this case besides of the number of persons and objects having to be the same the demand is always considered to be the unit. Practical applications of this problem appear in areas such as resource allocation, where it is necessary to assign machines to jobs/tasks, or persons to machines in a factory, for example. Another example of its application is the location of economical activities, (Koopmans and Berkmann, 1957). A good book on assignment problems is that of Burkard et al (2009). In Fig. 2.3 a graph with a solution for the assignment problem is represented.

Maximum Flow Problem - in this problem there are two special nodes, one is called the source node, from where some commodity is to flow until it reaches the other special node called destination or sink node. The objective is to find a path between these two nodes in such a way as to maximize the flow reaching the sink, provided

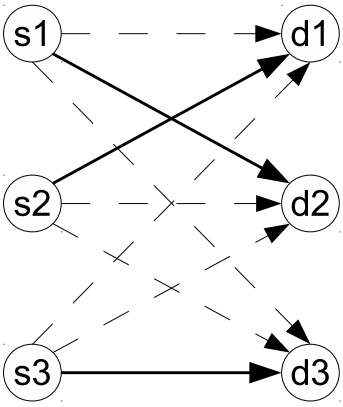

Fig. 2.3: Graph representing a feasible solution (shown in bold) to an assignment problem. To the left we have the origins and to the right the destinations.

that the capacity constraints are not violated. An example of such a network is given in Fig. 2.4, which is an adaptation of the one appearing in (Bertsekas, 1998).

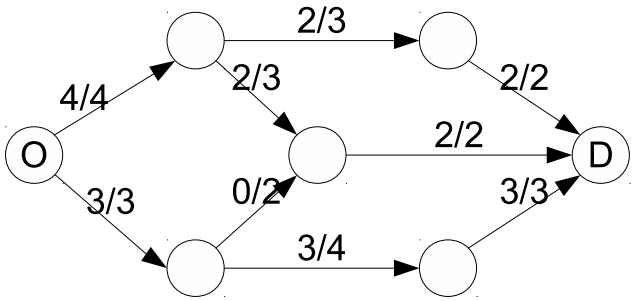

Fig. 2.4: A maximum flow problem with maximum flow. *O* identifies the origin node and *D* the destination node, and the numbers next to the arcs represent the flow and the capacity of the arc.

Furthermore, network flow problems can be further particularized accordingly to some major characteristics. Next, we briefly define some of those characteristics, provide a daily life practical example, and direct the reader to further reading:

- Commodity
  - Single-Commodity - a single type of commodity is to be flown in the arcs of a network, for example, when a certain brand of cars is to be distributed from the factory to the dealers (Ortega and Wolsey, 2003).
  - Multicommodity - several types of commodities, with different origins and destinations, are considered to flow in the arcs of the network at the same time, see Fig. 2.5. For example, a communication network, as we have at home, where different types of streams, data (computer), telephone, and TV

signal, are using the network simultaneously (Melo et al, 2005; Geoffrion and Graves, 2010).

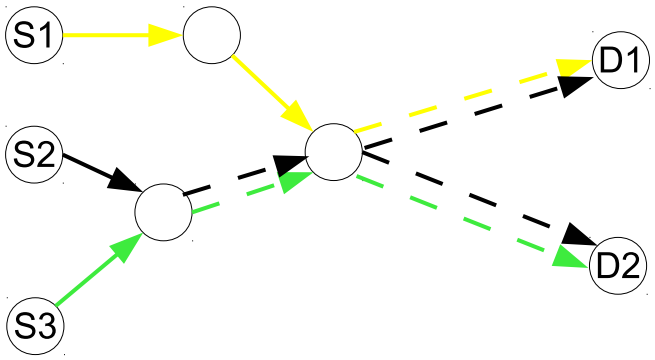

Fig. 2.5: Graph representing a multicommodity network. Dashed lines represent the flows sharing the same arcs, and different grey tones represent different types of commodities.

- Objective Function
  - Linear - a cost directly proportional to the flow on the arc (Vignaux and Michalewicz, 1991). Linear optimization problems are considered to be easier to solve than their nonlinear counterpart (Hochbaum, 2007).
  - Nonlinear - a nonlinear cost function is a function that is not directly proportional to the flow passing through an arc. One such case is the fixed-charge problem where a fixed cost is to be paid besides the cost depending on the flow. Nonlinear costs comprise concave, convex, and general nonlinear cost functions. The first are usually related to economies of scale and the latter, for instance, can be found in situations where the cost per unit changes accordingly to the consumption (we can find practical examples when paying for electricity, water, and so on), (Kim and Pardalos, 2000; Monteiro, 2005; Altiparmak and Karaoglan, 2007).
- Capacity
  - Capacitated - in the case it exists an upper and/or lower limit on the flow passing through an arc, on the demand, or on the supply, for example (Chen and Ting, 2008; Santos et al, 2010).
  - Uncapacitated - in the case no constraints exist on the flow in the arcs of the network, on the supply, and on the demand (Burkard et al, 2001; Ortega and Wolsey, 2003).

- Sources and Sinks
  - Single Source / Single Sink - when there is only one source and/or one sink to be considered in the problem. Examples of these problems happen when we consider a factory distribution network (single source), or a major supermarket company receiving all kinds of goods (single sink). Fig. 2.1 is an example of a simultaneous single source and single sink network (Fontes et al, 2003; Klose, 2008).
  - Multiple Sources / Multiple-Sinks - networks where the flow comes out of several sources and/or has several different destinations. These two situations can occur separately or in the same problem, for example, in timetabling problems where several teachers are to be scheduled to several classes, multiple-sources and multiple-sinks are considered. Fig. 2.2 shows a transportation problem with multiple origins and multiple destinations (Abramson et al, 1999; Chen et al, 2009).
- Time
  - Static - in the case where the problem characteristics do not change with time, that is, if the problem is to be solved having all characteristics known a priori (Aleskerov et al, 2003; Dang et al, 2011).
  - Dynamic - cases where the problem characteristics change with time; for example, where several time windows must be considered. Examples of time dependent problems can be found in scheduling of crews, vehicle routing with time windows or timetabling problems (Schaerf, 1999; Baykasoglu et al, 2006; Kliewer et al, 2012).

These characteristics are the most commonly used to define the nature of a network flow problem. Once its nature is known, the complexity of a problem can be identified and the researcher has all the means to decide upon a method to solve it. The next section briefly discusses the main guidelines of the theory of problem complexity.

2.3. Problem Complexity

One of the branches of the computational complexity theory focuses in classifying problems according to the difficulty they represent to solving methods, from a computational point of view. Problems may be classified as easy or hard to solve, depending on the

amount of computational resources needed in order for them to be solved. The best algorithm known to solve a problem is used to establish its complexity (Garey and Johnson, 1979).

Another branch of the computational complexity theory deals with the complexity of an algorithm which can also be accessed by the amount of computational resources needed by the algorithm to solve a given problem. Resources can be defined as the number of operations needed between the reception of an input and the final answer, also known as time complexity, or the memory needed to complete those operations. The time complexity is usually the one used in the classification. On the one hand, if the rate of growth of its computational time, given the size of a problem instance, can be described by a polynomial function, the algorithm is called efficient, and is classified as a *polynomial time deterministic algorithm*. On the other hand, if this time growth rate is comparable to an exponential function, then the algorithm is called inefficient, and is classified as an *exponential time algorithm*.

Problems can be classified in two classes: the P class, includes problems that can be solved with a polynomial time deterministic algorithm, in the worst case; and the NP class, which includes problems for which only non-deterministic polynomial time algorithms are known. When it is proven that a problem belongs to the P class, the problem is considered easy to solve, otherwise it is considered hard.

There are two more important classes to be characterized, NP-complete and NP-hard. A problem is said to be NP-complete if any other NP-problem can be reduced to it in polynomial time and whose solution may also be verified in polynomial time. This means that NP-complete problems can be viewed as at least as hard as any other NP-problem. However, some problems are known to be "Harder" to solve, and are classified as NP-hard problems, which includes all NP-problems at least as hard as NP-complete problems, i.e., all NP-complete problems can be reduced to a NP-hard problem in polynomial time.

Now, the million dollar question is whether or not $P = NP$? And, to the moment, it is a question still unanswered. However, it is strongly believed that $P \neq NP$ and many security protocols over the internet rely on it

NP-problems include P-problems and NP-complete problems, as long as $P \neq NP$, (Ladner, 1975), as illustrated in Fig. 2.6.

MCNFPs can be divided into two subdomains regarding the nature of the cost functions, linear or nonlinear ones. MCNFPs with linear costs are solvable in polynomial time, that is, they are considered easy to solve, and very efficient algorithms have already been developed to solve them, see e.g. (Bertsekas, 1998). Nonlinear cost functions can be

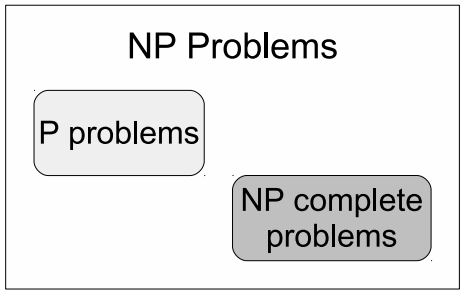

Fig. 2.6: Complexity classes, following Ladner (1975), as long as $P \neq NP$.

further divided into convex, concave, and general nonlinear ones. The work of Hochbaum (2007) provides an excellent discussion on complexity of convex network flow problems, updating Hochbaum (1993, 2005), specially on strong polynomial-time and polynomial-time algorithms to solve them.

The nonconvex case is a more complex one, although the great majority does belong to the NP-hard category (Garey and Johnson, 1979). In general, when concave costs are introduced in MCNFPs, the difficulty to solve them increases and they become NP-hard (Guisewite and Pardalos, 1990), even in the case of minimum cost network flow problems with fixed-charge cost functions where the ratio between flow costs and fixed costs is constant (Hochbaum and Segev, 1989).

Nonetheless, we provide references for some particular cases of concave MCNFPs more related to the problems hereby addressed that have been proven solvable by polynomial-time and strong polynomial-time algorithms, such as the: SSU concave MCNFP with only one demand node (Zangwill, 1968), the SSU MCNFP with a single nonlinear concave arc cost (Guisewite and Pardalos, 1993; Klinz and Tuy, 1993), the SSU MCNFP with two nonlinear concave arc costs and the MCNFP with two sources and one single nonlinear concave arc cost (Tuy et al, 1995b), the SSU MCNFP with a fixed number of sources and a fixed number of nonlinear concave arcs (Tuy et al, 1995a), and the SSU concave MCNFP with a fixed number of nonlinear cost arcs (Tuy, 2000).

General nonlinear MCNFPs are the most difficult to solve since neither convexity nor concavity properties can be explored. However, Lamar (1993) has developed a technique, which is discussed in Chapter 4, allowing the transformation of general nonlinear MCNFPs into concave MCNFPs.

2.4. Concave and Nonlinear Programming

Concave programming is one of the most well-studied problem classes in nonlinear optimization. There may exist several reasons why, but one of the most important ones is surely related to the many practical applications for this type of costs mainly because they are usually associated to real economic problems.

A concave minimization programming model can be, generally, given by:

$$\text{min:} \quad F(x) \tag{2.4}$$

$$\text{s.t.:} \quad x \in D, \tag{2.5}$$

where $D \neq \emptyset$ is a closed convex set in $\mathbf{R}^n$, and F is a real-valued concave function defined on some open convex set A in $\mathbf{R}^n$ that contains D, as given in (Benson, 2001). The objective is to find the global optimum of F in D.

There is an inherent difficulty in minimizing a concave cost function over a convex feasible region. Concave functions have a large number of local minima which are not global minima. This is the main reason why concave optimization problems are usually hard to solve in opposition to convex minimization problems where it is known that if a local minimum exists, then it is a global minimum. Guisewite and Pardalos (1990) have shown that the minimization of strictly concave cost functions on a network flow problem is NP-hard, and the same happens when the cost function considered is the fixed-charge.

Algorithms designed to solve convex programming problems generally fail when used to solve concave minimization problems. Nonetheless, concave minimization problems present some characteristics that can help to develop efficient algorithms to solve them. For example, the knowledge that if D is a compact set, i.e., a closed and bounded set, then we can conclude that the concave minimization program must have a global optimum which is an extreme point of D, (Benson, 1995).

In some minimization problems, general nonlinear cost functions can also be considered and they may have several local optima, making them hard to solve as well. The good news is that it has been proven in (Lamar, 1993) that NFPs with general nonlinear costs can be transformed into NFPs with concave costs, on an expanded network. Thus, methods developed to solve concave minimization problems can also solve minimization problems with more general nonlinear cost functions, although an increase on the problem size, with all the inherent complications, is expected and inevitable.

2.5. Solution Methods

In order to solve combinatorial problems, and network flow problems in particular, there is a wide range of available techniques that can be divided in two main classes: exact methods and approximate methods, heuristics being of the latter type.

An exact method, due to its exhaustive search of the solution space, guarantees that the solution found is an optimal solution for the problem. However, they usually require large amounts of computational resources and computational time since they explicitly or implicitly enumerate all feasible solutions of a problem (Michalewicz and Fogel, 2002). And, although small problem instances of some optimization problems can be solved quite quickly with these methods, when the size of the problem increases the number of possible solutions increases exponentially and enumerating them all would be too much time consuming. In some cases, the problem instances cannot be solved in a lifetime due to the large number of combinations to search for (Cordon et al, 2002). This fact is a major drawback because real-life applications can be quite large thus making it difficult to opt for the use of exact methods.

Heuristic methods, although are not guaranteed to find a global optimal solution for a problem, are usually able to find a good solution rapidly, perhaps a local optimum, and require less computational resources (Blum and Roli, 2003). This is the main reason for using heuristic methods when solving a combinatorial problem, and the reason why approximate methods are so popular.

The word *Metaheuristic* was first introduced by Glover (1986). Today this term is widely used and, in its essence, a metaheuristic is a series of procedures that can be used to define a heuristic algorithm to solve different types of optimization problems, with only a few modifications related to the nature and characteristics of the problem. Metaheuristics can be classified into many ways, please refer to (Blum and Roli, 2003) and to (Birattari et al, 2001) for more details. Herein, we only distinguish between *population-based* algorithms and *single-point* algorithms. Single-point search algorithms got the name from the fact that they work only with a single solution at a time. Tabu Search (TS), simple local search, and simulated annealing are some examples of this type of algorithms. Population-based algorithms work at the same time with a set of solutions, which are sometimes called “individuals”. ant colony and genetic algorithms are examples of population-based algorithms.

Next, we briefly review some of the most popular and well-known metaheuristics, starting with single-point based heuristics and then following to population-based heuris-

tics. This summary is intended to give an account of the existing techniques, and their main characteristics in order to sustain some decisions to be made afterwords in the development of the solution method chosen in this thesis.

Simple Local Search

The simple Local Search (LS) approach is amongst the most simple heuristics that can be implemented and used to solve combinatorial problems. This type of heuristic starts with the construction of an initial feasible solution. Then, this solution is iteratively improved, that is, at each iteration the algorithm searches for a better solution in some neighbourhood until some stopping criterion is reached, see Algorithm 1. All these three phases (initial solution, iterative improvement, stopping criterion) are very important since all have an impact on the quality of the final solution that is achieved and on the computational time required to obtain it (Blum and Roli, 2003).

Algorithm 1 Pseudo-code for Simple Local Search.

1: Generate an initial feasible solution X_0 for the problem at hand
2: Make $X \leftarrow X_0$
3: **while** ending criterion is not reached **do**
4: Apply local search operations: add, drop, swap, or combinations of these basic operations to X, and obtain X'
5: **if** $fitness(X') < fitness(X)$ **then**
6: $X \leftarrow X'$
7: **end if**
8: **end while**
9: Return X

The initial feasible solution can be constructed in many ways. In a facility location problem, for example, an initial feasible solution can be found by including all available sites in it, and then dropping one at a time until the pre-specified number of open facilities is met; or alternatively constructing it by adding a single facility at a time. After deciding on the technique to obtain an initial feasible solution X, the next step is to define the neighbourhood $N(X)$ of that solution. The neighbourhood of X is the set of all solutions that can be reached from X. In order to go from one solution X to another solution X' of $N(X)$, three basic operations can be considered: *add*, *drop*, and *swap*. In the add operation a solution component is added to the current solution. The opposite is the drop operation, where a component is dropped from the current solution. Finally, a swap can be an interchange between a component already in the solution that is going to be removed and a component (not in the solution) that is going to be added to the solution, or an interchange between two components already in the solution. For example, given

a solution S for the TSP, the swap operation between components of S represents the reversion of the order by which cities are to be visited.

From these basic operations many more can and have been defined. For instance, a p-swap operation, with $p > 1$, was used by Arya et al (2004). In a p-swap operation p components of the current solution are dropped, and p other components are added. Charikar and Guha (1999) have considered adding a component and dropping several ones. Ghosh (2003) defined and used add-swap, 2-swap, and permutation operations. In the permutation operation neighbourhood solutions differ in exactly two components.

Many stopping criteria can be defined, and they mainly depend on the author's preferences and on the problem being addressed. A LS algorithm can stop when a pre-specified number of iterations has been reached or when no better solution exists in the neighbourhood of the current solution. The latter one allows for the decrease of the algorithm's running time.

However, it should be noticed that such an algorithm stops at the first local optimum it finds. That is the reason why they often produce low quality solutions by themselves. Some changes have been introduced into the basic local search algorithm, in order to improve its performance, resulting in a new set of metaheuristics based on local search. Examples of these new local search methods are guided local search and iterated local search. Guided local search starts with an initial feasible solution that is improved by means of local search until a local optimum is found. Then, the objective function is iteratively changed in order to reduce the attractiveness of the local optima. Iterated local search differs from the previous only in the second phase. The algorithm performs a perturbation in the local optimum X obtaining a new solution X'. Then, by means of local search operations, another local optimum may be obtained.

Although local search algorithms can get "trapped" into a poor quality solution, they are very good at searching neighbourhoods. That is why many heuristics incorporate local search in some of their phases.

Local search algorithms have been used to solve many types of problems on their own or incorporated into other heuristics: location problems (Kuehn and Hamburger, 1963; Feldman et al, 1966; Wang et al, 2003; Arya et al, 2004), timetabling problems (Schaerf, 1999), network flow problems (Fontes et al, 2003), location-routing problems (Derbel et al, 2010; Duhamel et al, 2010; Nguyen et al, 2012), amongst others.

Tabu Search (Glover, 1986)

The tabu search is a single-point metaheuristic that tries to avoid getting trapped at a local optimum, by guiding the search trough the use of an adaptive memory structure, that is, through the use of the history of the search. Moreover, it also includes a *responsive exploration* of the search space. A tabu search approach consists of four main components: a short-term memory, a long-term memory, an intensification strategy, and a diversification strategy.

The *short-term memory* is used by means of a *tabu list* to escape from local optima and avoid cycling. Each time the algorithm visits a solution this solution is added to the tabu list. Whenever the algorithm moves to a new solution it first checks the tabu list so that no solution in the list is repeated. Sometimes, instead of the whole solution, only the most significant solution attributes are kept in the tabu list. The *long-term memory* is used throughout the search process, and collects information about common properties of good solution, so that the algorithm tries to visit solutions with different properties. Both types of memories can be classified into four principles: *recency*, *frequency*, *quality*, and *influence*. In the *recency-based memory*, the algorithm keeps track of the solution and the iteration it was last involved in. The *frequency-based memory* keeps a record of how often a solution (attribute) has been visited. The *quality* principle refers to the collection of components of good solutions in order to guide the search towards promising areas. Finally, the *influence* refers to the use of the most critical choices that were made during the search.

The intensification strategy is intended to concentrate the search near good solutions. The tabu search makes use of the components of good solutions to intensify the search in that area. The diversification strategy makes use of the moves that were made more frequently and guides the search towards moves not so frequent, in order to explore other areas of the solution space. These two phases are closely related to the allowed sizes of the tabu lists, since small tabu lists will force the algorithm to concentrate the search in solutions nearby the previous ones, while large tabu lists will push the algorithm to explore larger search areas including not yet explores ones. Algorithm 2 presents an example of the pseudo-code for the basic tabu search metaheuristic.

Applications of tabu search algorithms have been used to solve, for example, communication networks (Costamagna et al, 1998), assignment problems (Dell'Amico et al, 1999; Diaz and Fernandez, 2001), scheduling problems (Vasquez and Hao, 2001), location problems (Mladenovic et al, 2003), vehicle routing problems (Gendreau et al, 2008), hub covering problems (Calik et al, 2009), network pricing problems (Brotcorne et al, 2012), amongst others.

Algorithm 2 Pseudo-code for Tabu Search.

```
1: Generate an initial feasible solution X_0 for the problem at hand
2: Make X ← X_0
3: Initialize Tabu List as tbL ← ∅
4: while ending criteria is not reached do
5:    for every X' not tabu in the neighbourhood of X do
6:       Compute fitness(X')
7:    end for
8:    Identify best X'
9:    if fitness(bestX') < fitness(X) then
10:      X ← X'
11:   end if
12:   Update Tabu List
13: end while
14: Return X
```

Simulated Annealing (Kirkpatrik et al, 1983)

Simulated annealing uses a different strategy to avoid getting trapped into a local optimum. The main idea is that a solution is always accepted. If the candidate solution is better than the previous one, then it is immediately accepted; otherwise the solution may also be accepted, with some probability. Therefore, this method allows for moves that result in worse quality solutions in order to escape local optima. The algorithm starts by finding an initial feasible solution by means of a random generation or by using some more elaborated heuristic method. A parameter, called temperature, is used for the probability of acceptance of worse solutions. The temperature parameter is initialised to a high value, and then it is progressively decreased during the search process, therefore decreasing the probability of choosing non-improving solutions. When the temperature reaches 0, the algorithm converges to a simple iterative improvement algorithm (Kirkpatrik et al, 1983). The pseudo-code for this metaheuristic can be seen in Algorithm 3.

In this algorithm, the choice of the initial temperature, the cooling rate (known as cooling schedule), and the choice of the probability function to calculate the probability of accepting a non-improving solution (based on the current temperature) have a major influence in the algorithm performance. Many cooling schedules have been tried, and their nature can be arithmetic, geometric, quadratic, or heuristic. For the probability function there are also several options, but the Boltzmann-like distribution is the most commonly used one (Silva, 2003), and is given by equation (2.6).

$$p(\text{acceptance of } X') = \begin{cases} e^{\frac{f(X)-f(X')}{T_{it}}}, & \text{if } f(X) < f(X') \\ 1 & \text{otherwise,} \end{cases} \tag{2.6}$$

Algorithm 3 Pseudo-code for Simulated Annealing.

```
1: Generate an initial feasible solution X_0 for the problem at hand
2: Make X = X_0
3: Set the temperature to its initial value
4: while ending criteria is not reached do
5:    Find a neighbour solution X' from the current solution X
6:    if fitness(X') < fitness(X) then
7:       X ← X'
8:    else if fitness(X') ≥ fitness(X) then
9:       Calculate the probability of X' being accepted
10:      if Probability of being accepted > random() then
11:         Update current solution X ← X'
12:      end if
13:   end if
14:   Update the temperature according to the cooling function
15: end while
16: Return X
```

where T_{it} is the temperature value at iteration it, and $f(X)$ is the fitness of solution X.

Within the large range of optimization problems solved with simulated annealing, one can find, for example, TSPs (Skiscim and Golden, 1983), timetabling problems (Thompson and Dowsland, 1996; Abramson et al, 1999; Zhang et al, 2010), flowshop scheduling problems (Liu, 1999), assignment problems (Osman, 1995), routing problems (Lin et al, 2009), and reverse logistics network design (Pishvaee et al, 2010).

Genetic Algorithms (Holland, 1975)

Genetic algorithms were introduced by John Holland and his colleagues in the seventies. Initially, their objective was to study, in a formal way, the phenomenon of adaptation that occurs in the nature and to develop techniques to adapt these natural mechanics so that they could be implemented on a computer. The GA presented here is already an evolution of Holland's initial concept. Lately, this kind of approach has been the object of many studies and development. The GAs have been used for addressing various classes of problems: search, optimization, economy, immune systems, ecology, population genetics, evolution and learning, social systems, and many more (Mitchell, 1999).

The GAs are population-based procedures based on the genetic processes of biologic organisms and on natural selection principles. All living beings are made up of cells, each one of them containing a complete copy of the genetic model of the individual. This model appears in the form of *chromosomes*, each one composed of *gene* chains, which are made of a chemical substance named DNA. Generally, we can consider that a gene

encodes the characteristics of a being, like for example the colour of the eyes. During the reproduction, a combination or crossing of genes from the chromosomes of the parents occurs. Therefore, a new being is a combination of different chromosome structures. The generated being can suffer mutation and thus, part of the DNA may be altered, resulting in a gene different from those of the parents. Apart from the genetic process we know that populations evolve through generations. This occurs according to principles of natural selection and of the survival of the fittest. So, it does not come as a surprise that by imitating this process one hopes that a genetic algorithm can solve practical problems.

In a GA, each *chromosome* represents an *individual*, which in turn represents a possible solution to a problem. A group of individuals is designated by *population*, and represents the starting point of a GA. From this initial population a new one is to be generated by crossing the best existing individuals and by mutating them. The evolution to a new population is always made in the hope that the new population is better than the previous one and that this process leads to the generation of an individual that is an optimum solution or, at least, a good solution for the problem in hands. Fig. 2.7 represents the general structure of a GA.

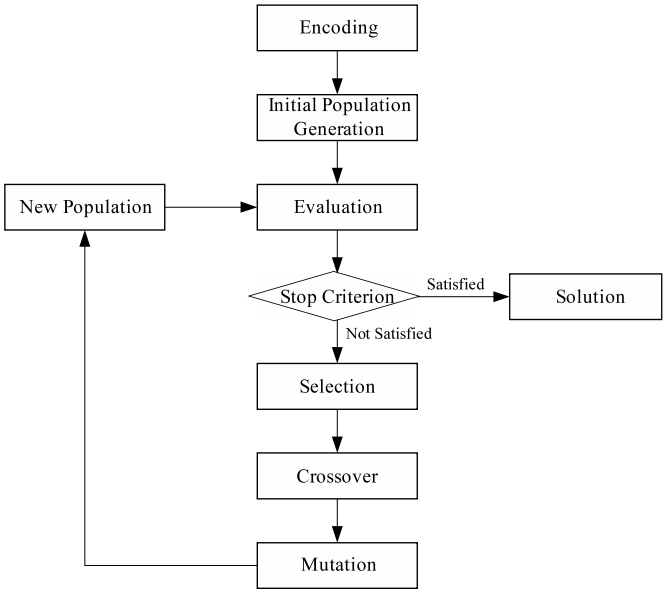

Fig. 2.7: Structure of a Genetic Algorithm.

The first step when applying a GA to a given problem is the encoding of the possible solutions. This is a key factor and it heavily depends on the problem being studied. Many of the implemented GAs and theoretical studies are based on binary encoding. In this case, each *chromosome* is represented by a string of bits, each bit having a zero or one value. A *gene* consists either on a single bit or on an association of adjacent bits. For example, if we want to maximize a function with three variables $F(x, y, z)$, we can represent each of

these variables by a 10-bits binary digit. Therefore, the resulting chromosome has three genes, and globally it is made of 30 bits. Other alternatives are also available, instead of zeros and ones, characters, real numbers, or objects more closely related to the problem can be used.

Once the encoding is defined it is necessary to generate an initial population. To do so the definition of the process to generate individuals must be set along with the size of the population. The number of individuals, although set initially to a certain value, may be changed during the experiences to try to achieve a better performance. The simplest way to create an initial population is to randomly generate the values of the string that constitutes the chromosome, in the domain of possible values. Another way is to use algorithms or heuristics so that the initial individuals are already reasonable solutions to the problem.

Next, it is necessary to define the genetic operators to be used. The two most popular operators are crossover and mutation. Crossover is made after the best individuals (*parents*) are chosen for reproduction. There are several techniques to perform crossover, but here we explain the three most commonly used:

- single-point crossover - a position in the chromosome is selected randomly. This is designated as *crossover point*. Two new individuals (*offsprings*) are created by switching the parent's genes that lay after the crossover point.
- two-points crossover - in this case the switched genes are the ones in the middle of crossover points previously defined.
- uniform crossover - in this case each gene is chosen from one of the parents according to a probability p.

None of the existing techniques has been proved to be the best, i.e. better for every problem. Thus, one must rely on past experience and computational experimentation to find the best one for the problem being solved. The mutation operator is used after the crossover and according to a predefined probability. This operator randomly selects a position in one of the chromosomes and changes its value. The idea is that by doing so a small uncertainty is assured in the search. This way none of the points in the search space has zero probability of being examined.

Finally, a new population is generated, usually with the same size as the previous one. This new population replaces the previous one. However, to avoid loss of information there is the possibility of maintaining some elements of the previous population. This strategy is designated by elitism, and the individuals to be kept are the ones with the

higher fitness. This process is iteratively repeated until a stopping criterion is satisfied. This criterion can be a pre-specified number of generations or solutions, or related to the solutions found, i.e. solution improvement or solution quality.

Genetic algorithms, are one of the most used heuristics in literature, and have been used to solve many different problems: jobshop scheduling problems (Varela et al, 2001; Gonçalves et al, 2011), location problems (Jaramillo et al, 2002), assignment problems (Chu and J.E., 1997; Shima et al, 2006), routing problems (Machado et al, 2002), space allocation (Silva, 2003), knapsack problems (Thiel and Voß, 1994), nurse rerostering problem (Pato and Moz, 2008), and loading problems (Gonçalves and Resende, 2012), just to mention but a few.

Ant Colony Optimization (Dorigo et al, 1991)

This metaheuristic is inspired in the communication between real ants while foraging for food sources, and relies on the reinforcement of components of good solutions found by artificial ants. We refrain ourselves from a thorough description of this metaheuristic since Chapter 3 is wholly dedicated to it.

This is just a popular subset of the ever increasing set of metaheuristics. As an example, recently two more optimization algorithms have been proposed for the first time, the bees algorithm (Pham et al, 2005), and the water drops algorithm (Shah-Hosseini, 2009). The former is based in the foraging behaviour of honey bees, while the latter is based in the way rivers and water courses find good paths between the source and their destination.

2.6. Summary

In this chapter, we have introduced network flow problems, the class of problems to which the problems studied in this thesis belong to. We have started by introducing some terminology that will be used herein, so as to make it easier for the reader to follow the work. A brief classification of NFPs was also introduced, as well as the discussion of complexity classification among optimization problems. Concave and nonlinear programming were briefly discussed. Some of the most popular Metaheuristic solution methods, among the large set of available heuristics, were also summarized. A working paper discussing network flow problems and reviewing the past and the most recent literature published on the subject, some of which also presented in Chapter 4, has been published Monteiro et al (2012c).

3

Ant Colony Optimization

3.1. Introduction

In the previous chapter, we have introduced and described in general terms network flow problems, as well as an overview of the most used metaheuristics to solve them. Now we turn our attention to the review and discussion of the methodology that we have chosen to use to solve them.

Therefore, this chapter describes and discusses ant based algorithms, and in particular the Ant System (AS), that is the first ant algorithm developed, on which the Ant Colony Optimization (ACO) metaheuristic is based. Please note that, from now on whenever we refer to ant algorithms, we mean algorithms based on the foraging behaviour of ants, since other types of ant based algorithms exist. Our main objective in this work is to use ACO procedures to solve two types of network flow problems: minimum cost network flow problems and hop-constrained network flow problems, which will be described in the chapters that follow.

The idea behind ant algorithms, rather than truly mimic the behaviour of real ants, is to adapt and use their communication style, which has been proven to be so good in nature. Artificial ants can then be seen and described as communicating agents sharing

some characteristics of the real ants, but also incorporating other characteristics that do not have a parallel in nature, (Solimanpur et al, 2004). The overall characteristics are what makes them fit to solve problems, if not optimally, at least by finding very good solutions. A real foraging ant spends all its life travelling between its nest and some food source. It does not then come as a surprise that the first problem solved with an ant algorithm, called ant system, was the Travelling Salesman Problem (TSP), a well-known combinatorial problem, where the shortest circuit (path) between a given set of cities, starting and ending at the same city, is to be found.

The very good results that were being achieved with ant algorithms pointed to the broadening of the definition of *path* therefore allowing the use of this method to solve other problems. Some adaptations of the algorithm had to take place, resulting in the so called ant colony optimization metaheuristic, which is based on the ant system. The definition of the ACO metaheuristic, as a series of generic guidelines that could be very easily adapted to almost all types of combinatorial optimization problems, allowed a boost in the use of this methodology and in the number of researchers and publications in the area. Since then, ACO procedures have been applied to solve a broad set of problems, including: network design Problems (Rappos and Hadjiconstantinou, 2004), assignment problems (Shyu et al, 2006; Bernardino et al, 2009), location problems (Baykasoglu et al, 2006; Pour and Nosraty, 2006; Chen and Ting, 2008), transportation problems (Musa et al, 2010; Santos et al, 2010), covering problems (Lessing et al, 2004; Crawford and Castro, 2006; Mehrabi et al, 2009), just to mention but a few in the area of combinatorial optimization. Curiously enough, although the TSP was the first problem to be solved by the AS and ACO metaheuristics, it still inspires researchers such as García-Martínez et al (2007) that have recently used ACO to solve a bi-criteria TSP or Tavares and Pereira (2011) that use the TSP to test an evolving strategy to update pheromone trails.

Although in general ACO algorithms achieve very good results, there are cases where an hybridization with other heuristics or metaheuristics, proves to be necessary. Therefore, in the past few years authors have developed hybrid algorithms between ACO and local search (Pour and Nosraty, 2006), simulated annealing (Bouhafs et al, 2006), post processing procedures (Crawford and Castro, 2006), and even with genetic algorithms as is the case of (Altiparmak and Karaoglan, 2007). This allowed ant algorithms to get even better results in problems too complex to be solved by a single heuristic method.

In the following sections we explore, in detail, the first ant algorithm, the ant system, and review some of the large number of interesting works that have been developed ever since.

3.2. Ant Colony Principles

Ant colony optimization principles are based on the natural behaviour of ants. In their daily life, one of the tasks ants have to perform is to search for food in the vicinity of their nest. While walking in such a quest, the ants deposit a chemical substance called *pheromone* in the ground. This is done with two objectives. On the one hand, it allows ants to find their way back to the nest, such as Hansel and Gretel in the fairytale. And on the other hand, it allows other ants to know the way they have taken, so that the others can follow them. The curiosity is that, because hundreds or even thousands of ants have this behaviour, if one could see the pheromone laid in the ground as a kind of light, the ground would be a large network with some of the arcs brighter than the others. And within the paths created by those arcs would surely be the shortest path between the nest and the food source. This behaviour can be seen as a kind of communication between the ants. If the path has a larger concentration of pheromone, this is probably due to its shorter length that has allowed ants to travel faster, resulting in a larger number of travels through the path therefore with much more ants depositing pheromone on it. This behaviour observed in ants, i.e. trail-laying and trail-following is the inspiring source of ACO (Dorigo and Stützle, 2004).

One of the first and most interesting experiences with real ants, in search for a reason for the choice ants make between paths to follow, was the experience made by Deneubourg and colleagues (Deneubourg et al, 1990; Goss et al, 1989). In this experience, there were two bridges connecting the nest of an Argentine ant species (*I. humilis*) and a food source, one bridge being longer than the other. The experiences were ran with a set of different ratios between the bridges size. The results showed that although at first the ants seemed to choose at random the path to follow, after a while they all converged to the shorter path. The investigators found that this was due to the amount of pheromone present in the path. If the number of ants walking in a certain path is high, then the pheromone quantity that will be deposited in that path is also high, which incentives other ants to follow it. One interesting observation was made when one of the paths was twice as long as the other. Although the majority of the ants followed the shorter path, there was always a small percentage that used the longer path. This fact was interpreted as a kind of exploration behaviour (Dorigo and Stützle, 2004). With these experiences, Deneubourg and colleagues (Deneubourg et al, 1990; Goss et al, 1989) where able to show that foraging ants can find the shortest path between their nest and some food source, by the use of a chemical substance called pheromone, that they deposit while walking. After these experiments, the authors proposed a stochastic model to describe what they had observed.

This was the first step leading to an optimization algorithm based on the ants behaviour. This algorithm was called *Ant System* and was firstly proposed in (Dorigo et al, 1991, 1996) to solve the travelling salesman problem.

3.3. From Biological to Artificial Ants: Ant System and Ant Colony Optimization

Marco Dorigo was the first author to propose an ant algorithm to solve combinatorial optimization problems in cooperation with his two colleagues Alberto Colorni and Vittorio Maniezzo (Dorigo et al, 1991; Dorigo, 1992). The first ant colony optimization algorithm to be developed was called ant system and was used to solve the travelling salesman problem, a well known NP-hard problem.

3.3.1. Ant System

In order to explain the ant system algorithm, and how it was used to solve the TSP, we will first define the Euclidean TSP having in mind that any other distance definition can be used.

In a TSP there is a set N with n cities, and a matrix D with the distances between each pair of cities. The distance between city i and city j is defined as:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \tag{3.1}$$

where x_i and y_i are the plane coordinates of city i. If the distance used is based on the euclidean distance definition then $d_{ij} = d_{ji}$. However, distances d_{ij} and d_{ji} are not always thus defined which means that D can be asymmetric, and that does not interfere with the AS structure. If we were to define the TSP in a graph, each city could be seen as a node and the path between pairs of cities could be seen as arcs of the underlying graph.

The objective of the TSP is to find the shortest tour between a set of cities, starting and finishing in the same city, going through all cities and visiting each city exactly once. This problem is very easily adapted to the idea of the ant system due to their similarity in concepts. Having defined the TSP, let us now proceed to the description of the AS algorithm.

An AS algorithm considers a single ant colony with m artificial ants cooperating with each other. Before the algorithm starts to run each arc linking two different cities is given

a certain quantity of pheromone τ_0. Also, the ants are created. The number of ants to be used is considered a critical decision because too many ants constructing solutions may induce the algorithm to converge quickly to suboptimal solutions, while a reduced number of ants will reduce the effect of communication between the ants. Dorigo et al (1996) have suggested the use of as many ants as the number of cities, for the TSP case.

The algorithm has two main phases, the construction of the tour/solution and the pheromone update. Other important decisions have to be made before the ants can start finding a solution, such as defining the structure (representation) of the solution, or the initial pheromone quantity to be given to each arc. These questions will be discussed further ahead.

In each iteration, and while the maximum number of iterations is not reached, each ant is randomly placed in a city, from the set of n cities. That city will be the starting point of the tour that is to be constructed by the ant. In order to solve the TSP, the number of ants used is usually the same as the number of cities, so that every city has the same probability of being the starting city of the tour. A solution to the TSP can be represented by a set of n consecutive cities. Therefore, at each step of the construction an ant has to choose the next city to travel to. This choice is probabilistic.

This choice is made by using a *transition rule*, the short expression for *random proportional transition rule*, which is based on the quantity of pheromone τ_{ij} and on the heuristic information η_{ij} of each arc. The transition rule quantifies the probability of ant k, positioned at city i, travelling to city j and it is given by:

$$P_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta}, \tag{3.2}$$

where η_{ij}, the *heuristic information*, is the inverse of the distance between city i and city j, i.e.

$$\eta_{ij} = \frac{1}{d_{ij}}, \tag{3.3}$$

J_i^k is the set of cities not yet visited by ant k while at city i, and α and β are parameters weighting the relative importance of the pheromone and of the heuristic information, respectively.

The heuristic information, which is a fixed value from the beginning of the algorithm, is also called the *visibility* of arc (i, j) and it can be seen as the information an ant has, in order to make its choice, if it can use its eyes (Afshar, 2005). Therefore, the closest cities, that is, the ones that the ant can see from where it is standing, will have a higher visibility

value, whereas the others will have a lower one.

The values α and β are two tunable parameters that weight the pheromone information and the heuristic information on the transition rule. On the one hand, if α has a very small value (close to 0) then the closest cities are more likely to be chosen, and the algorithm becomes a sort of greedy algorithm. On the other hand, if β is the one with a very small value then the algorithm will give priority to the pheromone information and faster convergence to poor solutions may arise. Therefore, these two parameters must be carefully chosen, since otherwise the algorithm may have a poor performance.

After building the solutions the pheromone values in the arcs are updated. The update is done in two phases. Just before the ants can deposit pheromone in the arcs of their solution, the algorithm applies an *evaporation rate* ρ, with $\rho \in]0, 1]$, to the pheromone in each arc, as given in equation (3.4). This operation simulates the natural process of evaporation preventing the algorithm from converging too quickly (all ants constructing the same tour) and getting trapped into a local optimum. The value of the evaporation rate indicates the relative importance of the pheromone values from one iteration to the following one. If ρ takes a value near 1, then the pheromone trail will not have a lasting effect, potentiating the exploration of the solutions space, whereas a small value will increase the importance of the pheromone, potentiating the exploitation of the search space near the current solution.

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t). \tag{3.4}$$

The length S^k of each tour is then calculated and the ants will be allowed to deposit pheromone in every arc of their tour. The pheromone quantity to be deposited in each arc,$\Delta\tau_{ij}^k(t)$, is inversely proportional to the quality of the solution of each ant and proportional to the number of ants that incorporate the arc in its solution, as can be seen in equations (3.5) and (3.6).

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^k(t). \tag{3.5}$$

$$\Delta\tau_{ij}^k(t) = \begin{cases} \dfrac{Q}{S^k(t)} & \text{if } (i,j) \text{ belongs to the solution of ant } k, \\ 0 & \text{otherwise,} \end{cases} \tag{3.6}$$

where Q is a positive proportionality parameter and $S^k(t)$ is the length of the tour constructed by ant k at iteration t.

In a small problem instance, this update will impose a reduction of the search space thus converging to one where the optimal solution components will have the highest values in the matrix. However, in large instance problems it is known that stagnation is likely to happen, driving the solution to a suboptimal solution rather than to an optimal one. This is why pheromone evaporation is so important.

The previous steps are performed until some stopping criterion is reached, which can be a fixed number of iterations, as was the case, but it could also be the setting of a bound on running time or even the number of solutions evaluated.

The best values for the parameters used in ant algorithms depend both on problem characteristics and on the orientation to input on the search of the solutions space. Recall, for instance, that the larger the value of ρ the higher the priority given to exploration. Therefore, before any setting on the parameter values, decisions on the orientation of the search have to be made. Then, the algorithm must be run several times in order to establish the best values. For this particular algorithm and problem, the parameter values found to be the best were the following: $\alpha = 1$, $\beta = 5$, $\rho = 0.5$, $m = n$, $Q = 100$, and $\tau_0 = 10^{-6}$.

This first experience with Ant System was proposed by Dorigo in his Doctoral Dissertation (Dorigo, 1992). His experiences with the AS originated three variants called *ant-density*, *ant-quantity*, and *ant-cycle*, (Dorigo et al, 1991; Dorigo, 1992). The main difference between them is that whereas in the ant-cycle version (the one we have explained) the pheromone update is done after all the ants have constructed their tour, in the ant-density and in the ant-quantity the update is done immediately after a move from one city to another is made. In ant-density the pheromone quantity to be deposited is the same for all arcs Q_1, (see Eq. 3.7), whereas in the ant-quantity the value depends on the length of the arc d_{ij} (see Eq. 3.8).

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q_1 & \text{if the } k\text{th ant chooses arc } (i,j), \\ 0 & \text{otherwise,} \end{cases} \tag{3.7}$$

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \dfrac{Q_2}{d_{ij}} & \text{if the } k\text{th ant chooses arc } (i,j), \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}$$

It has been proven later, in (Badr and Fahmy, 2004), that Ant algorithms converge with probability one to a stable path. Nonetheless, the results obtained with the ant-cycle algorithm were much better than the ones obtained with the other two algorithms, which were abandoned. The *ant-cycle* algorithm is now known as Ant System.

3.3.2. Improvements to the Ant System

The AS algorithm achieved good results for small instances but not so good for larger ones. Therefore, changes were subsequently introduced in order to improve the results obtained.

Elitist AS - provides pheromone reinforcement to all the arcs belonging to the best tour found since the beginning of the algorithm (Dorigo et al, 1996). Therefore, the updating rule becomes:

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) + \Delta\tau_{ij}^{best}(t), \tag{3.9}$$

where,

$$\Delta\tau_{ij}^{best}(t) = \begin{cases} \frac{e}{S^{+}} & \text{if } (i,j) \text{ belongs to the best solution found so far,} \\ 0 & \text{otherwise,} \end{cases} \tag{3.10}$$

$best$ stands for the best solution, S^{+} is the length of the best solution, and e is a positive integer parameter.

Rank-based AS - the arcs belonging to the $r-1$ best ants have a pheromone reinforcement, as well as the arcs belonging to the best solution since the beginning of the algorithm (Bullnheimer et al, 1997). The pheromone quantity deposited is dependent on the rank of the ant:

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{r-1}(r-k)\Delta\tau_{ij}^{k}(t) + \Delta\tau_{ij}^{best}(t). \tag{3.11}$$

Max-Min AS - introduces four changes to the original AS: only the arcs on the best tour in an iteration or the arcs on the best tour since the beginning have their pheromone updated; the pheromone in each arc is bounded to be in the interval $[\tau_{min}, \tau_{max}]$; the initial pheromone trail in each arc is equal to τ_{max}; finally, if there is no improvement in the tours for some consecutive iterations, the pheromone trails are reinitialized. For further details see (Stützle and Hoos, 1997) or Section 3.3.4.3.

Ant Colony System (ACS) - was another successful improvement on the AS, and was introduced by Dorigo and Gambardella (1997). The ACS brought four modifications to the original AS. Firstly, the introduction of a new transition rule, called *pseudo-random transition rule*:

$$j = \begin{cases} \arg\max_{u \in J_i^k} \{[\tau_{iu}(t)] \cdot [\eta_{iu}]^\beta\} & \text{if } q \leq q_0, \\ J & \text{if } q > q_0, \end{cases} \tag{3.12}$$

where q is a random variable uniformly distributed over $[0, 1]$, $q_0 \in [0, 1]$ is a parameter, and J is a city (node) to be randomly chosen from the set of cities not yet visited by ant k based on the probability function. The probability function is similar to the one used in the AS, see Eq. (3.2), the only difference being that $\alpha = 1$, as given by Eq. (3.13).

$$P_{ij}^k(t) = \frac{[\tau_{ij}(t)] \cdot [\eta_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}.(t)] \cdot [\eta_{il}]^\beta}. \tag{3.13}$$

This way, either city j to be added to the solution is randomly chosen, although based on the pheromone values, or it is deterministically chosen as the one maximizing the product between the pheromone trail and the heuristic information. Parameter q_0 measures the behaviour of the algorithm in the sense that small values for q_0 will lead to a more probabilistic algorithm, while large values will make it more deterministic. Secondly, the pheromone trail update rule is global and thus only allows the best ant, from the beginning of the run, to reinforce the pheromone values in its arcs, as given in:

$$\tau_{ij}(t+1) = (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}^{best}(t), \tag{3.14}$$

where $\Delta\tau_{ij}(t)$ is the inverse of the length of the best solution.

$$\Delta\tau_{ij}^{best}(t) = \frac{1}{S_{best}}, \tag{3.15}$$

Thirdly, after each step of the construction of the algorithm, the ant updates the pheromone in the respective arc. This is done by initially evaporating the pheromone value of the arc and then depositing a small quantity, proportional to τ_0,

$$\tau_{ij}(t) \leftarrow (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0. \tag{3.16}$$

This local updating rule favours exploration by decreasing the desirability to incorporate the arc in the solution. Therefore, the more ants travel through that arc, the less desirable it will appear to the following ants. Finally, the fourth modification is the use of a candidate list. This list will only include some of the closest cities

provided that they are not yet in the partial solution. These cities are investigated first, and only then other cities are to be examined.

3.3.3. Ant Colony Optimization

Meanwhile, and following these new improvements to the AS, the most important development is the description of the ant colony optimization metaheuristic in (Dorigo and Di Caro, 1999; Dorigo et al, 1999). The ACO, which is described in Algorithm 4, is made of general guidelines for the development of ant algorithms to solve combinatorial optimization problems.

Algorithm 4 Pseudo-code for Ant Colony Optimization.

```
1: Initialize parameters
2: Initialize pheromone trails
3: Create ants
4: while Stopping criteria is not reached do
5:     Let all ants construct their solution
6:     Update pheromone trails
7:     Allow Daemon Actions
8: end while
```

The main difference from the basic structure of the AS algorithm is the introduction of a *Daemon*. The daemon can perform operations that use global knowledge of the solutions, thus having a very active and important role in the algorithm, in contrast to the AS algorithm where each ant was supposed to deposit pheromone in its solution despite what the other solutions were like. This is a task that has no equivalence in the nature. The daemon can, for example, control the feasibility of each solution or give an extra pheromone quantity to the best solution found from the beginning of the run of the algorithm, or even to the best solution in the current iteration. These last operations were already mentioned in previous algorithms but never attributing its responsibility to a main entity in the colony.

Another important feature, frequently used by authors on ant colony optimization algorithms is the introduction of local search procedures following the construction of the solutions. This is an optional feature that has been proven to be very important in the exploitation of the search space near to good solutions, leading almost always to better performances of the ACO.

In this section, we have given a brief historical perspective of the idea that lead to the development of ant algorithms and on the most important and following developments. The number of investigators working with and the number of ant algorithms, have been

growing exponentially, specially following the definition of the ACO metaheuristic. It would therefore, be impossible to give an accurate account of all of them.

3.3.4. The Building Blocks of an Ant Algorithm

Ant colony algorithms have a set of characterising features that can be considered as their step stones or building blocks. These features must always be specified, preferably, when describing an ant algorithm:

- method chosen to construct the solution,
- heuristic information,
- pheromone updating rule,
- transition rule and probability function,
- parameters values, and
- termination condition.

It becomes obvious that the different techniques that can be developed for each of them, when combined with each other, result in a large diversity of ant colony algorithms, each of which more adequate to a certain class of problems. Within the vast literature on the subject, different proposes can be identified to either improve earlier results or simply to solve a new type of problems. In the previous sections we have discussed, with some degree of detail, the AS that was used to solve the TSP. In this section, and for each of these building blocks, we review some of the extremely large number of techniques previously proposed, since it is impossible to track all the work that has been done ever since the early stages of ant algorithms. Nonetheless, the reader is always referred to the works that will be discussed in this section, for further details, as well as to the references therein.

3.3.4.1. Constructing a Solution

The construction of a solution, along with its representation, is one major aspect of an ant algorithm as it is with any other heuristic methods, because it will influence the rest of the procedures to be defined. Thus, it plays a crucial role on the success of the algorithm. Besides, it is common knowledge that it has a great effect on the running time of the ACO

algorithm (Neumann and Witt, 2010). Therefore, if the construction is badly chosen, the probability of a bad performance is high.

One critical decision is whether the algorithm must construct a feasible solution or it is allowed to construct unfeasible ones. This decision alone can have several outcomes, such as:

- allowing the construction of unfeasible solutions and then creating an additional procedure to fix them. This may involve a significant running time effort;
- allowing the construction of unfeasible solutions and then discarding or penalizing unfeasible solutions. In this case, if the number of unfeasible solutions is high then, given the lack of choice, the algorithm may converge too quickly to a suboptimal solution;
- if only feasible solutions are allowed, then the construction procedure may be too complex leading to large running times.

These examples illustrate that, as someone has pointed out before but within another context, *"There is no such thing as a free lunch"*[1].

The construction of a solution is influenced by many aspects, such as the problem being solved, the constraints to be considered, the representation chosen for the problem, the investigator preferences, and so on.

Alaya et al (2004) solve a multidimensional knapsack problem where a decision on a subset of objects, satisfying a few resource constraints, has to be made in order to maximize the total profit. The solution for this problem only requires the choice of a subset of objects to be introduced in the knapsack, with no order specified, and can then be represented as a string of object identifiers. Each ant starts the construction of their solution by randomly choosing an object to be put in the knapsack. Then, objects are added to the solution, by using a transition rule as defined in Eq. (3.2), as long as they do not violate any resource constraint. This way a feasible solution is always obtained.

Rappos and Hadjiconstantinou (2004), in order to design two-arc connected flow networks, use two types of ant colonies sharing information about their pheromone levels. This problem is about configuring a network in order to satisfy demand nodes, provided

[1] We are referring to the *No Free Lunch Theorem* defined by (Wolpert and Macready, 1995, 1997). In it, the authors conclude that different algorithms perform differently while solving the same problem and that each algorithm has to be tuned with information of the problem to be solved, in order to have a good performance. When we use the expression here, we mean that hard work has to be done to choose the best information to incorporate into the algorithm so as to have good results.

that an extra arc is considered to keep the network flowing in the case that one of the arcs in the network fails. The solution for this problem is constructed in two phases, each of which solved by a different type of ants. One ant colony is inhabitated by flow ants and the other colony by reliability ants. The number of flow ants is the same as the number of demand nodes and, although they all start constructing their solution from the source node, each ant is assigned to reach just one specific demand node. When all flow ants have constructed their partial solutions, reaching their demand node destination, the network is created. The next step involves the reliability ants whose objective is to decide upon an extra arc, called reliability arc, to be added to the solution. For every flow ant a reliability ant is created and associated with each arc visited by the flow ant. Therefore, for each flow ant there is a set of reliability ants, as many as arcs in the solution of the flow ant. The objective of a reliability ant is to find an alternative path from the root node to the same demand node of the flow ant as long as it does not use a particular arc, from the ones used in the solution of the flow ant. This ACO algorithm provides a single feasible solution at each iteration, which is only entirely defined when all partial solutions of the flow ants have been assembled together, and the extra arc found by the reliability ants is identified.

Baykasoglu et al (2006) solve a dynamic facility layout problem, where each ant has to decide, for each period t, the location of the n departments considered. The authors use a string with size $t \times n$ to represent the final solution, where the first n consecutive values identify the department locations for the first period, the second n consecutive values give their locations for the second period, and so on. Therefore, to construct a solution, the ants have to choose $t \times n$ elements of the type (*department, location*), accordingly to the pheromone levels, such that within a time period no pair (*department, location*) is repeated, thus guaranteeing the construction of a feasible solution.

In (Crawford and Castro, 2006) partitioning and covering problems are solved with ACO algorithms. In this case, given a set of columns and rows, the objective is to choose a subset of columns covering all rows while minimizing cover costs. The solution is represented by a subset of columns. This implies a different approach, from the ones we have been mentioning before, because the solution components are represented by nodes and not by arcs, a fact that simplifies the calculations. The construction is straightforward. Each ant starts with an empty set of columns. Then, the ant adds columns one at a time, based on pheromone values, until all rows are covered. Solutions constructed in this way, can be unfeasible in the partitioning case because a row may be covered by more than one column. That is why post processing procedures, that will try to eliminate redundant columns, are added afterwards in order to turn unfeasible solutions into feasible ones.

In a transportation problem with N supply nodes and M demand nodes, it is known that a solution has, at most, $N + M - 1$ arcs. This observation allowed Altiparmak and Karaoglan (2006) to decide on allowing each ant in their algorithm to be able to choose $N + M - 1$ arcs to construct a feasible solution. Each ant starts by randomly choosing an arc from the set of available arcs A, and proceeds the construction by adding, one at a time, the remaining arcs by using pheromone information. The arcs in the set of allowed arcs to be chosen to enter the solution are defined by the demand and supply nodes that have not exceeded already their demand and supply, respectively. In this case, the notion of path is not applied since arcs are chosen arbitrarily, as long as they satisfy demand and supply constraints.

The single source capacitated facility location problem deals with the location of a set of facilities, each with limited capacity, and the allocation of a single facility to each customer so as to satisfy customers demand. Facilities are chosen such that the total cost is minimized. Chen and Ting (2008) propose an Ant Colony System with two types of ants, location ants and assignment ants, to solve it. Therefore, there are two different solution representations and constructions. Location ants select the facilities to be opened, and their solutions are not uniform, that is each ant can open a different number of facilities, accordingly to:

$$f_a = \left\lfloor \frac{\sum_i d_i}{\sum_j (s_j/m)} \right\rfloor + U[0, r], \tag{3.17}$$

where f_a is the number of facilities to be opened by ant a, d_i is the demand of customer i, s_j is the capacity of facility j, m is the number of available locations for the facilities, and r is a pre-specified integer constant with a value between the first term of the sum in Eq. (3.17) and m. After the selection of the facilities to be opened, assignment ants assign each customer to one and only one facility but they do not acknowledge, at least in this phase, whether the solution is feasible or not, from the supply capacity point of view. The unfeasible solutions are dealt with by using penalties in the local search phase.

3.3.4.2. Heuristic Information

The heuristic information, also known as visibility, is an extra information available to ant algorithms which is usually referred to as a kind of local information. Originally used as the inverse of the length of the arc between two cities in the TSP, see Section 3.3.1, it has suffered several mutations throughout the hundreds of approaches that have been developed since. The heuristic information is a very effective value when associated to the

correct exponent, and as it usually is a value between 0 and 1, the larger the exponent the lower will be its impact on the probability function and on the choice of the components to integrate the solutions constructed by the ants.

Lessing et al (2004) studied the influence of the heuristic information, in the performance of ant algorithms when solving set covering problems. Two types of heuristic information are studied, static heuristic information, where the values are calculated once at the beginning of the algorithm, and dynamic heuristic information, whenever the values are recalculated at each construction step of each ant, as in the case of the ant-density algorithm. The different heuristic information values used are based on the Column Costs

$$\eta_j = \frac{1}{c_j}; \tag{3.18}$$

(normalized) Lagrangean Costs

$$\eta_j = \frac{1}{C_j}; \tag{3.19}$$

Cover Costs and (normalized) Lagrangean Cover Costs, respectively, where $card_j(S)$ is the number of rows covered by column j

$$\eta_j = \frac{card_j(S)}{c_j}, \tag{3.20}$$

and

$$\eta_j = \frac{card_j(S)}{C_j}; \tag{3.21}$$

Marchiori and Steenbeck Cover Costs,

$$\eta_j = \frac{1}{(c_j/cv(j,S)}, \tag{3.22}$$

where $cv(j,S)$ is the sum, for all rows covered by column j but not covered by any other column in $S \setminus \{j\}$, of the minimum cover costs;

Marchiori and Steenbeck Lagrangean Cover Costs with Normalized Costs,

$$\eta_j = \frac{1}{(C_j/cv(j,S)}; \tag{3.23}$$

and finally lower bounds, where the heuristic information for column j is the inverse of the cost of the lower bound obtained by tentatively adding column j;

Each of these heuristic information types was tested with four different ant algorithms: min-max AS, ACS, a hybrid between min-max AS and ACO, and the approximate nondeterministic tree search ACO algorithm. The results obtained suggest that different types of heuristic information should be used for different types of ant algorithms.

Reimann and Laumanns (2006) use savings values as the heuristic information instead of the usual inverse of the arc cost,

$$\eta_{ij} = S_{ij}, \tag{3.24}$$

in an ACO algorithm to solve capacitated minimum spanning tree problems. The savings $S_{ij} = c_{i0} + c_{0j} - c_{ij}$ are related to the cost difference obtained by merging the subtrees of node i and j, previously linked directly to the source node 0. In this case, the larger the savings associated to an arc the higher probability of that arc being selected.

A capacitated fixed-charge location problem aims at deciding on the supply facilities that must be opened such that they can satisfy all the customers demand at the lowest possible cost. Venables and Moscardini (2006) developed an ACO based algorithm that defines and uses the information of a matrix T_{ij} called the total opportunity matrix. This matrix uses the sum of the differences between the cost c_{ij} of each arc (i, j) and both the lowest supplying cost c_{i^*j} from facility i and the lowest supplying cost c_{ij^*} to customer j, i.e. $T_{ij} = (c_{ij} - c_{ij^*}) + (c_{ij} - c_{i^*j})$. The visibility is set to be the facility visibility and is defined as the sum on the customer indices of the total opportunity cost,

$$\eta_i = \frac{1}{\sum_{j=1}^{n} T_{ij}}. \tag{3.25}$$

The lower the T_{ij} the higher the visibility of the arc and probability of it being chosen.

In (Altiparmak and Karaoglan, 2007) the heuristic information is based on the concave nature of the arcs costs in the transportation problem to be solved. In this case, the heuristic information takes into account not only the cost of the arc c_{ij} but also the flow of that arc x_{ij}, and is given as

$$\eta_{ij} = \frac{1}{(c_{ij}/\sqrt{x_{ij}})}. \tag{3.26}$$

Pour and Nosraty (2006) have solved the NP-hard plant/facility location problem with an ACO algorithm. In this problem, there is a set of existing facilities p_i and a set of locations where new facilities x_j are to be located and the objective is to locate these new facilities, such that the sum of the costs between the facilities is minimized, and each

location is assigned a single facility. In this algorithm, the heuristic information used by the authors is defined as the inverse of the product of distances d_i and costs f_i between existing facility i and all new facilities x_j, taking the form of

$$\eta_{ij} = \frac{1}{f_i \cdot d_j}. \tag{3.27}$$

This way, nearest and lower cost facilities have a better heuristic value.

The minimum weight nodes cover problem is solved by Shyu et al (2004) with an ACO algorithm where the heuristic information is defined for pairs of the type $(node, ant)$. In it, the heuristic information is defined as the local preference of ant k to choose node j to enter the solution, which is translated into the ratio between the number of arcs linked to node j, but not yet covered by ant k, and the weight of node j. Thus, this heuristic information is not static since its value changes with each step of the construction of the solution, and also from solution to solution, since ants may construct different solutions.

Crawford and Castro (2006) calculate the value of the heuristic information in a dynamic fashion, to solve partitioning and covering problems. At each step of the construction of the solution, the algorithm computes the heuristic information as the per unit cost of covering an additional row, as given bellow

$$\eta_j = \frac{e_j}{c_j}, \tag{3.28}$$

where e_j is the number of additional rows that are covered by node j when it is added to the partial solution already constructed.

In the Cell Assignment Problem (CAP) a set of cells must be linked to a set of switches such that each cell is associated to exactly one switch but switches may be linked to several cells. Shyu et al (2006) define two heuristic information matrices, instead of the usual single one, to be used in an ACO algorithm developed to solve the CAP. These matrices are associated with the choice of the switch to move to when located at a certain cell, and vice-versa. The former decision uses a heuristic information function based on the inverse of the partial costs. This heuristic information is dynamic since whenever an arc (c_i, s_j) linking a cell c_i and a switch s_j is included in the partial solution, the heuristic value for that arc will be updated with the inverse of the partial solution cost constructed to the moment. This update is performed at each step of the construction procedure. Therefore, the higher the partial cost the lower the value of the heuristic information η_{c_i,s_j}. Whenever an ant is located at a certain switch it must choose to which cell to move to. In order to do so, the heuristic information used is the call volume associated with each cell, thus η_{c_i} is defined for cells rather than for arcs. Therefore, the larger the call volume the higher the

value of the heuristic, thus favouring cells with high call volumes to be handled first.

3.3.4.3. Pheromones and the Laws of Attraction

It has been mentioned earlier in this chapter, it is a good idea to set some bounds in the values allowed to be taken by pheromones. At some point on the run of an ACO algorithm, the values of the pheromones in the components of the solution may be extremely small, almost prohibiting the choice of those arcs, or extremely large which will lead to the construction of the same solution, over and over again. To prevent that from happening, as already mentioned, setting upper and lower bounds on the pheromones may be a solution. The first work to introduce this mechanism was (Stützle and Hoos, 1997). By proving the following proposition:

Proposition 1. *For any τ_{ij} it holds:*

$$\lim_{t\to\infty} \tau_{ij}(t) \leq \frac{1}{\rho F(S^{opt})}, \tag{3.29}$$

where ρ is the evaporation rate and $F(S^{opt})$ is the objective function value of the optimum solution. Since the optimum solution is not known a priori, it must be estimated by using the value of the best solution S^{best} found to the moment. This implies that whenever this solution is updated, the value of τ_{max} is also updated. Therefore, τ_{max} can be defined as given in Eq. 3.30.

$$\tau_{max} = \frac{1}{\rho F(S^{best})}. \tag{3.30}$$

Having established the maximum allowed value for the pheromone, the setting of τ_{min} followed. Two observations were very important in this case: after stagnation, the probability of constructing the global best solution is significantly higher than zero and better solutions have a good chance to be found near it; the construction of a solution is much influenced by the relative difference between the maximum and minimum values of the pheromones. After stagnation, and assuming that a solution has n components to be chosen, the probability of constructing the best solution is given by $p_{best} = p_{dec}^n$, where p_{dec}^n is the probability to choose the right component at each of the n steps of the solution construction. Furthermore, on average, avg , ants have $n/2$ components to choose from, which together with the probability function, results in

$$p_{dec} = \frac{\tau_{max}}{\tau_{max} + (avg - 1)\tau_{min}}. \tag{3.31}$$

Since the result we want is to find a valid value for τ_{min} we can write it as

$$\tau_{min} = \frac{\tau_{max}(1 - p_{dec})}{(avg - 1)p_{dec}}. \tag{3.32}$$

This value depends on the value of τ_{max}, therefore, whenever a new global best solution is found τ_{min} must also be updated.

Other approaches have been suggested by other authors. For example, Venables and Moscardini (2006) and Altiparmak and Karaoglan (2007) both define the upper bound τ_{max}, as in Eq. (3.30). The minimum pheromone value allowed is given by a fraction of the maximum pheromone value allowed,

$$\tau_{min} = \tau_{max}/a, \tag{3.33}$$

where a is a parameter value dependent on the size of the problem. It is easy to see that τ_{min} and τ_{max} are not static values changing whenever a new best solution is found.

Another mechanism also based on pheromone trails is the identification of stagnation. Altiparmak and Karaoglan (2007) use a two phase reinitialization scheme. On the one hand, if more than 50% of the arcs in a transportation network have pheromone values equal to τ_{min}, then τ_{max} and τ_{min} are updated according to the global best solution and all values in the pheromone matrix are set to τ_{max}. On the other hand, if the global best solution has not been updated for 50 iterations, then 10% of the population is randomly generated and will replace the worst solutions.

Blum and Blesa (2005) propose an ACO algorithm to solve arc-weighted k-cardinality tree problems, where the pheromone values are in the interval $[0, 1]$, following the HyperCube Framework defined in (Blum et al, 2001). In order to define an usable value for each limit, the minimum and maximum pheromone values are as given in Eq. (3.34).

$$[\tau_{min}, \tau_{max}] = [0.001, 0.999]\,. \tag{3.34}$$

The algorithm also incorporates a so-called convergence factor cf, defined according to Eq. (3.35), in order to estimate the degree of convergence.

$$cf = \frac{\sum_{a \in A(S_k^{it})} \tau_a}{k \cdot \tau_{max}}, \tag{3.35}$$

where k is the cardinality of the problem, A is the set of arcs belonging to the best k-cardinality tree of the iteration, τ_{max} is the maximum pheromone value, and finally $\tau_{a)}$ is

the pheromone value for arc a. By definition cf is a value always between 0 and 1, and the closer cf is to 1, the quicker the system converges because the probability to construct again S_k^{it} is closer to 1. When this happens pheromone values and the best solution are reset.

Bui and Zrncic (2006) address the degree-constrained minimum spanning tree problem. In order to do so, they define the maximum and the minimum value allowed for pheromone levels based on the differences between the cost M of the most expensive arc and the cost m of the cheapest arc, as follows

$$\tau_{max} = 1000 \cdot \frac{(M-m)+(M-m)}{3}, \tag{3.36}$$

and

$$\tau_{min} = \frac{M-3}{3}. \tag{3.37}$$

Whenever the pheromone value of an arc exceeds τ_{max} it is not reset to τ_{max}, as usual, but rather adjusted to $\tau_{max} - \tau_{ij}^{init}$, where the initial pheromone value for arc (i,j) is given by $\tau_{ij}^{init} = (M - c_{ij}) + (M-m)/3$. In a similar way, $\tau_{ij} = \tau_{min} + \tau_{ij}^{init}$ whenever the pheromone value falls bellow τ_{min}. This way, as some of the original information is maintained it is expected that the ant still recognizes good arcs and bad arcs.

Bin et al (2009) use lower and upper bounds for the pheromone values in the arcs of the routes found by the ants, for the vehicle routing problem, which are dependent on the distance d_{0i} between the central supply node 0 and each customer node i. The bounds are given by

$$[\tau_{min}, \tau_{max}] = \left[\frac{Q}{\sum_i d_{0i}}, \frac{Q}{\sum_i 2d_{0i}} \right], \tag{3.38}$$

where Q is a parameter. These bounds are calculated only once, at the beginning of the algorithm. The algorithm incorporates a mutation operator, in a similar fashion to genetic algorithms, to try to include arcs other than the ones with higher pheromone value, chosen by influence of the probability function. Given two parent tours from a solution, two customers, one from each tour, are randomly selected and exchanged. If this operation turns out to result into unfeasible solutions, then they are fixed by using a repairing mechanism. Thus, two new feasible solutions are always created.

3.3.4.4. Pheromone Update

In the definition of the ACO metaheuristic the pheromone update has been defined to be performed after all the ants have constructed their solutions. Although it is the recommended/suggested method, it has not been proven to be the best choice for all problems. In fact, different pheromone update schemes have been provided in the literature differing in three key aspects: the moment at which pheromones are updated, the pheromone quantity to be deposited and evaporated, and which ants are allowed to deposit pheromone in their trails.

The work of Talbi et al (2001) is one of those cases where an alternative approach has proven to achieve good results. In order to solve a Quadratic Assignment Problem (QAP), the pheromone update instead of reinforcing the components of the best solution found, as is usually done, reinforces every solution $F(S)$ taking into account both the value of the best ($F(S^*)$) and the value of the worst ($F(S^-)$) solutions found, as follows

$$\tau_{ij} = (1-\rho) \times \tau_{ij} + \frac{\rho}{F(S)} \times \frac{F(S^-) - F(S)}{F(S^*)}. \tag{3.39}$$

The intention is to weaken the reinforcement, preventing a quick convergence, due to the unusual large number of ants depositing pheromone on their solutions.

A different approach is that of (Rappos and Hadjiconstantinou, 2004) that was developed to design flow networks that are two-arcs connected, that is, that can continue to satisfy the customers demands if any single arc in the network is removed. In this problem arcs are undirected. Having in consideration the nature of the problem, the authors decided to make a distinction between two types of pheromone values associated to each arc. One, $T_e(ij)$, is called the *arc trail* and is related to the fixed cost that has to be paid for using that arc. The other one, $T_f(ij)$, is called the *flow trail* and is related to the cost of the flow passing through the arc. Flow ants, which can detected and reinforced both pheromone types, are created, as well as reliability ants, that can only detect and reinforce arc pheromone. In each iteration a single solution is produced by the flow ants, and then the solution is made reliable by adding an extra arc by the reliability ants. Both flow trails and reliability trails are updated, at the end of the corresponding construction phase, by initially performing a reduction on the pheromone of all arcs. Then, each reliability ant adds:

$$\Delta T_e(ij) = \frac{1}{b_{ij}} \tag{3.40}$$

to each arc on its solution, regarding the arc pheromone trail, where b_{ij} is the fixed cost

associated to arc (i, j). Each flow ant adds the following quantities to the arc and flow pheromone trails, respectively, on the arcs of its solution, provided that fixed-costs are paid only once

$$\Delta T_e(ij) = \frac{1}{b_{ij}} \text{ and } \Delta T_f(ij) = \frac{1}{c_{ij}d_j}, \tag{3.41}$$

where c_{ij} is the flow cost per unit and d_j is the demand of node j. Reliability ants do not deposit pheromone on flow trails since the extra arc that they add to the solution does not carry any flow.

In (Alaya et al, 2004), where multidimensional knapsack problems are solved, the pheromone update is done in such a way that the quantity deposited in each component of the solution includes information about the difference between the objective function value of the best solution of the iteration $F(S^{it})$ and of the global best solution $F(S^*)$,

$$\Delta\tau_{ij} = \frac{1}{1 + F(S^*) - F(S^{it})}. \tag{3.42}$$

Therefore, the closer the solution is to the global best solution, the higher the quantity of pheromone deposited.

Two pheromone updating rules are used in (Shyu et al, 2004), a global and a local one, as follows. At the end of each iteration, and after evaporation is applied, the pheromone quantity on the nodes of the incumbent solution S^* is reinforced with an amount $\Delta\tau_{ij}$ inversely proportional to the total weight of the nodes in the solution

$$\Delta\tau_{ij} = \frac{1}{\sum_{j \in S^*} w_j}. \tag{3.43}$$

Each time the ant adds a node into its solution the local pheromone updating rule is applied as given by

$$\tau_i \leftarrow (1 - \varphi)\tau_i + \varphi\tau_0, \tag{3.44}$$

where τ_0 is the initial pheromone laid in every node and φ is the evaporation rate applied locally. This latter rule has the objective of preventing the ants of always choosing the most significant nodes. Eswaramurthy and Tamilarasi (2009) have also used a similar global and local updating procedure, but considering arcs instead of nodes. It should be noticed that while Shyu et al (2004) solve the minimum weight nodes cover problem, Eswaramurthy and Tamilarasi (2009) solve the job shop scheduling problem.

In (Solimanpur et al, 2005), all ants are allowed to deposit pheromone, however, the deposited amount is larger for the solutions closer to the global best solution. No evaporation rate is considered. The amount of pheromone $\Delta\tau_{ij}^k$ to be deposited by ant k in arc (i, j) is given by

$$\Delta\tau_{ij}^k = \lambda \cdot \frac{F(S^*)}{F(S^k)}, \tag{3.45}$$

where $F(S^k)$ is the solution of ant k, and λ is a scaling factor that must be chosen appropriately such that a quick convergence to a local optima may be avoided. This method clearly encourages search along the vicinities of the global best solution in the hope that a better one can be found nearby.

In order to solve the arc weighted k-cardinality tree problem, Blum and Blesa (2005) define a pheromone update rewarding the ants associated with the following three solutions: the best solution in the current iteration S_k^{ib}, the best global solution to the moment S_k^{gb}, and the restart-best solution S_k^{rb}, that is, the best solution found at the restart of the algorithm. The reinforcement is then not based on the fitness of the solution, that is, the corresponding value of the objective function, but rather on the value of the convergence factor cf, see Eq. (3.35). Each of these three solutions is attributed a different weight k_{ib}, k_{gb}, and k_{rb}, defined in the same manner as above, such that their sum equals 1. The schedule the authors have applied is dependent on cf in such a way as to increase the value of k_{rb} and decrease the value attributed to k_{ib} with the increase of cf, if $cf < 0.99$. The value of the evaporation rate ρ parameter is also dynamic, decreasing with the increase of cf. When a global convergence has been reached, that is, when $cf \geq 0.99$ then the only solution being updated is S_k^{gb}, that is $k_{gb} = 1$. This happens because the pheromone values are to be reset and this is the only solution to be maintained. The update of pheromone values is performed as usual, i.e., initially it is evaporated and then the following pheromone quantity is added for each arc a,

$$\xi_a = k_{ib}\delta(S_k^{ib}, a) + k_{rb}\delta(S_k^{rb}, a) + k_{gb}\delta(S_k^{gb}, a), \tag{3.46}$$

where $\delta(S_k, a) = 1$ if arc a belong to the solution tree S_k, and 0 otherwise.

Following the work of Bin et al (2009), Yang et al (2007) use an ant-weight pheromone updating strategy based in the one of the ant-density algorithm, described in Section 3.3.1, in an ACO based algorithm used to solve the bus network design problem. The idea behind it is to incorporate both local and global information about solutions. Therefore, every ant k deposits the following pheromone quantity in its solution components

$$\Delta\tau_{ij}^{k} = \frac{Q}{K \times L} \times \frac{D^{k} - d_{ij}}{m^{k} \times D^{k}}, \tag{3.47}$$

where Q is the usual proportionality constant parameter, L is the sum of the lengths of all routes in the solution, K is the total number of routes in the solution, D^k is the length of route k, d_{ij} is the distance between customer i and customer j, and m^k is the number of customers visited in route k. Note that, a solution is only entirely defined when all routes constructed are assembled. The first component $\frac{Q}{K \times L}$, represents the global pheromone increment and it depends on the total length of the solution and on the number of routes, representing a compromise between the total cost and the number of vehicles used. The second component $\frac{D^k - d_{ij}}{m^k \times D^k}$, represents the local pheromone increment and is related to the cost contribution of arc (i, j) to the kth route, which increases as d_{ij} decreases.

3.3.4.5. Transition Rule and Probability Function

This may be considered the characteristic that has less changed with the evolution of ant algorithms. Its initial structure, as given in Eq. (3.2), is almost always used in the works of the researchers in the area. Nonetheless, different methods have been introduced mainly associated to the high complexity of the problem to be solved.

The probability function used in (Bouhafs et al, 2006) to calculate the probability of visiting customer j when in customer i, for a capacitated location-routing problem, also incorporates the savings value γ_{ij}, for visiting customer j from customer i:

$$P_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta} \cdot [\gamma_{ij}]^{\lambda}}{\sum_{j \in J_i^k} [\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}^{\beta}] \cdot [\gamma_{ij}]^{\lambda}}, \tag{3.48}$$

where J_i^k is the set of costumers not yet visited by ant k (in its solution) and that by being chosen do not violate any constraint. The transition rule used is an adaptation of (3.12), in order to incorporate the new component γ_{ij} of the probability function. The savings value is computed once at the beginning of the algorithm as follows

$$\gamma_{ij} = d_{i0} + d_{i0} - g \cdot d_{ij} + f \cdot |d_{i0} - d_{0j}|, \tag{3.49}$$

where g and f are parameters, d_{ij} is the distance between nodes i and j, and 0 is the starting node.

Afshar (2005) proposes a new transition rule for ant colony optimization algorithms, that is given by:

$$P_{ij}^{k}(t) = \frac{\alpha\tau_{ij}(t) + \beta\eta_{ij}}{\sum_{j \in J_i^k}[\alpha\tau_{ij}(t) + \beta\eta_{ij}]}. \tag{3.50}$$

The strategy is defined to prevent a domination of the pheromone trails in the ants decision, by incorporating an additive form instead of the usual multiplicative form. This way, the author expects both pheromones and heuristic information to have an active role in the decision. This new transition rule comes with a modification of the heuristic value, which is a simple scaling procedure given by:

$$\eta_{ij}^{s} = \frac{\eta_{ij}}{\max(\eta_{ij})}, \tag{3.51}$$

making every value to be between zero and one regardless of problem size. The transition rule proved to be able to overcome the stagnation problem.

The probability function developed in (Maniezzo, 1999) for the quadratic assignment problem is used within an algorithm developed to solve single row layout problems by Solimanpur et al (2005). The function also presents an additive scheme but eliminates the necessity of the parameter β associated to the heuristic value

$$P_{ij}^{k}(t) = \frac{\alpha\tau_{ij}(t) + (1-\alpha)\eta_{ij}}{\sum_{j \in J_i^k}[\alpha\tau_{ij}(t) + (1-\alpha)\eta_{ij}]}. \tag{3.52}$$

In this case, it is clear that α must be a number between zero and one, and not any positive number as was the case of the original method where α is a positive number. Therefore, if one wishes to prioritize the pheromone information one is implicitly decreasing the importance of the heuristic information, and vice-versa, and there is only one value for which they have the same weight, which is 0.5.

Blum and Blesa (2005) introduced some changes to the transition rule defined for the ACS (see Section 3.3.2), in order to solve k-minimum spanning tree problems. An ant starts its solution by randomly choosing the first arc to enter the solution tree. Then, at each step of the construction, the next arc a to be added is chosen deterministically if $q \leq 0.8$, and probabilistically if $q > 0.8$, according to Eq. (3.53):

$$a = \begin{cases} \arg\min\left\{\dfrac{\tau_a}{w(a)} : a \in A_{NH}(S_{t-1})\right\} & \text{if } q \leq 0.8, \\ l & \text{if } q > 0.8, \end{cases} \tag{3.53}$$

where τ_a is the pheromone in arc a, $w(a)$ is the weight of arc a, $A_{NH}(S_{t-1})$ is the set of all arcs that do not belong to solution S_{t-1} and have exactly one end-point in S_{t-1}, and

where

$$l = \begin{cases} \dfrac{\tau_a / w_a}{\sum\limits_{a'} \in A_{NH}(S_{t-1})} \tau_{a'} / w_{a'} & \text{if } a \in A_{NH}(S_{t-1}), \\ 0 & \text{otherwise.} \end{cases} \tag{3.54}$$

This rule assigns equal weight to the pheromone value and the heuristic value, hereby represented by $1/w_a$, by eliminating parameters α and β from the exponents of the pheromone and heuristic value, respectively. Given that the probabilistic rule is only triggered whenever a random number $q > 0.8$, the search for solutions is, in 80% of the cases, concentrated on relatively good areas.

3.3.4.6. Parameter Values

The setting of an ant colony based algorithm can take a long time in order to produce some useful results, as is the case for other algorithms. Furthermore, a set of parameter values has also to be defined:

- α - parameter related to the weight of the pheromone concentration in the probability function;
- β - parameter weighting the relative importance of heuristic information in the probability function;
- ρ - pheromone evaporation rate, where $\rho \in]0, 1]$, measures the information that is to be transported to the next iteration;
- Q - parameter weighting the quantity of pheromone to be deposited in each component of the solution;
- τ_0 - initial pheromone value to be deposited in every solution component;
- number of ants in the colony;
- stopping criterion - the number of iterations to be performed, the number of solutions to be evaluated, maximum allowed running time, and so on.

For each algorithm developed, there can be other parameters to be set, for example, if bounds are imposed on the pheromone values, a p_{best} parameter, as well as, the limit values have to be defined. There are other cases where differences in the definition of the

probability function, or the type of ant used, require more parameters. We feel that there is no need to report on those parameters in a section of their own as they tend to be unique for each algorithm. Nonetheless, we believe that almost every work that was reviewed in this chapter reports to have tried several combinations of parameter values before the final choice was made.

3.3.5. Multiple Ant Colonies

If a single colony of ants can solve a hard problem, what about a set of colonies? As with many other heuristics, ant algorithms are very adequate to be used in parallel. Problems where they can have a significant contribute are, for example, multi-criteria problems, with each colony solving the major problem from one criterion point of view.

Gambardella et al (1999a) use multiple ant colonies to deal with the vehicle routing problem with time windows. In it, a fleet of vehicles has to provide service to their customers such that two objectives are satisfied: the minimization of the number of routes and the minimization of total travel time. The minimization of the number of routes is considered to take precedence over the minimization of time. Each of these objectives is to be handled by an ant colony. The ACS-VEI is the ant colony that handles the minimization of the number of vehicles (routes), while the ACS-TIME is the colony that optimizes the travelling time of the solutions found by ACS-VEI. Each ant colony has its own pheromone matrix. The algorithm starts by finding an initial feasible solution with a nearest neighbour heuristic. Then, the ACS-VEI will try to decrease, by one vehicle, the number of vehicles used in that initial solution. After that, the ACS-TIME colony will try to optimize this new solution. If the new feasible solution improves the global best solution S^*, then S^* is updated and in the case that the number of vehicles is lower than it was in the previous global best solution the two colonies will be disintegrated. The algorithm, based on this new S^*, will work in cycle until some stopping criterion is met. The authors tested the algorithm using a set of benchmark problems, having been able to improve, both in terms of time and performance, upon previous results.

Multi-colony ant algorithms can also be used to favour the exploration of different regions of the search space. Colonies having a similar behaviour can be defined such that, initially, each is devoted to a specific region, exchanging information, from time to time, with the other colonies. Obviously, the exchange of information becomes a major issue, if not *the* major issue, and the following questions may arise:

"What, when, and how often are the procedures to transfer information between them?"

Krüger et al (1998) have tried to answer to the *what* question, and have concluded that a complete exchange of the pheromone matrices between the colonies is not the best approach. Middendorf et al (2002) completed this study by comparing four methods of exchanging information, all of them based on the transfer of a single or of few solutions. The first strategy identifies the best solution amongst the several colonies and sends it to all the other colonies. The second strategy considers a virtual ring formed by the colonies. Each colony will then send its best solution to its neighbour colony. The third strategy also considers the existence of a virtual ring but instead of using only one solution, each colony uses the information about their m best solutions. Each pair of neighbour colonies finds the best m solutions between both of them, and these are the ones used to update the pheromone matrix. These solutions are called migrants. Finally, the fourth strategy is a combination of the previous two methods,i.e., besides sending its best solution to its neighbour colony there is also a migrants exchange. Strategy one and two can be seen graphically in Fig. 3.1. Strategy two was considered to be the best approach because although, at first, colonies may diverge towards different regions on the search space, they tend to slowly converge to the same region, after a certain number of iterations. This gives them time to explore different regions and then to combine the best parts of the solutions, before convergence. These authors also address the question of: *"How many colonies do we have to use in the algorithm?"* If the problem is multi-criteria, the answer may be almost immediate and intuitive, one colony for each criterion. But, when using homogeneous colonies it can be tricky to choose the adequate number in order to solve the problem, as a large number of colonies also implies the growth of the computational time. Middendorf et al (2002) recommend the use of a moderate number of colonies, which will depend, of course, on the problem to be solved and its size. These authors test their algorithms with both the TSP and the QAP, having concluded that a problem such as the QAP does not profit from a multi-colony approach as opposed to the TSP that does.

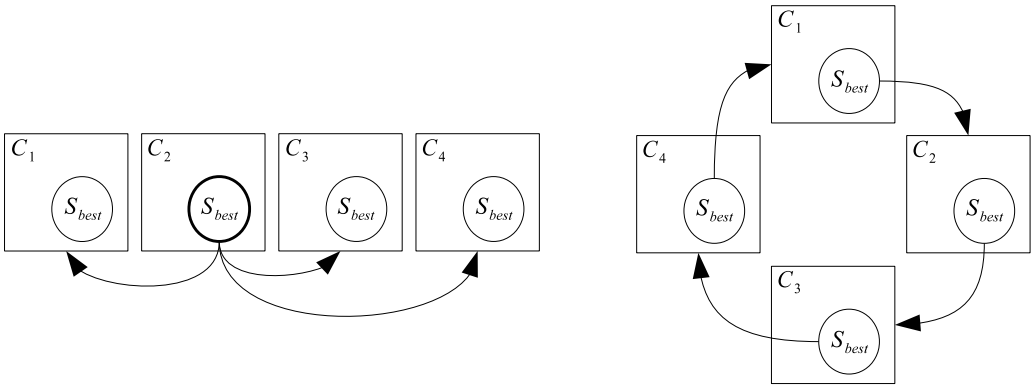

Fig. 3.1: Examples of the first (on the left) and of the second (on the right) exchange of information strategies for an algorithm comprising four ant colonies.

Talbi et al (2001) use a set of parallel ant colonies, each colony with a single ant, using an ant system, in an attempt to solve large combinatorial problems, and choose the QAP to test it. The solution is represented as a permutation of n integers, one for each object to be located. The colonies are set to work in parallel and there is a daemon to centralize information. Therefore, at each iteration, the daemon distributes the single existent pheromone matrix amongst the ants in each colony. The ants construct their solution and apply a tabu search in order to improve it. Then, each of the colonies sends both the solution and its frequency matrix back to the daemon. The frequency matrix keeps track of all the previous assignments and is to be used in the diversification phase. After identifying the best solution the daemon is the one updating the pheromone matrix, accordingly, and also the frequency matrix. New solutions are sent to the colonies by the daemon when the diversification phase starts, based on the frequency matrix that has been collected throughout the algorithm and that now is functioning as a long term memory.

A similar approach has been done by Chen and Ting (2008) to solve single source capacitated facility location problems using two different types of ant colonies. The algorithm that Chen and Ting have developed is called Multiple Ant Colony System (MACS) and is an extension of the one developed by Dorigo and Gambardella (1997). The choice of two colonies of ants is related to the problem being solved. On the one hand, the decision to locate (open) a set of capacitated facilities has to be taken. On the other hand, customers have to be assigned to one and only one of those facilities in order to have their needs satisfied. The objective is to minimize the sum of fixed costs incurred with the establishment of the facility and of the transportation costs incurred with the shipment of the commodity from the facility to the customer. The first phase is the location of facilities, which is determined by the location ants. These ants use a heuristic function, which is given by the ratio between the capacity s_j of each facility and its fixed-charge f_j, i.e., $\eta_j = s_j/f_j$. Location ants deposit pheromones in facility candidate sites. Therefore, although the probability function used is of the type of Eq. (3.2), only one index is used. The number of facilities opened by each ant differs because it is calculated based on the average number of facilities needed to satisfy the demand, plus a uniformly distributed random number, generated between 0 and the number of facility candidate sites. The second phase, the customer assignment phase, is determined by the selection ants. The heuristic information used by this colony of selection ants is the shipping cost (c_{ij}) between site i and customer j. Selection ants deposit pheromone in arcs between facilities and customers and the probability function is thus the same as in Eq. (3.2). Pheromone update is done locally after each ant constructs its solution and globally after local search is performed. At the end of each iteration the best solution is identified and local search is performed on it. As the assignment of customers to facilities is done such that no

consideration is taken for the feasibility of the solution, the iteration best solution may be unfeasible. If this is the case, a penalty cost is added to the cost function value of the solution. Next, insert moves and swap moves are alternatively performed until the best possible feasible solution is found. Swap moves are performed between customers assigned to different facilities, and insert moves are performed on customers assigned to facilities that have exceeded their capacity. If necessary, another facility may be opened to reassign that customer, but only if the facility fixed cost is smaller than the penalty value. Another heuristic is proposed by the authors, resulting in the hybridization between a Lagrangian Heuristic (LH) and a modification of the MACS, explained above. The LH is used to set the locations of the facilities to be opened and the selection part of the MACS is used to assign customers to facilities. The quantity of pheromone deposited by the ants is based on the values of the best solution found so far and on the best and on the worse solutions of the current iteration. Both algorithms were applied to benchmark problems and LH-MACS was found to be the best of the two and also to have outperformed other heuristics reported in the literature, in respect to the solution quality.

3.3.6. Hybridization

Although Ant colony optimization algorithms are known to behave fairly well for combinatorial optimization problems, it is also known that when the size of the problem or its complexity begin to increase, the algorithm may converge to a solution still far from the optimum. It does not come as a surprise then, that researchers have been trying to deal with this handicap by hybridizing ACO with some other heuristics / metaheuristics. In this section, we present some of the studies that have inspired us in the research we have conducted in the pursuit of a method, involving ACO, that we could use to solve the problems already mentioned in Chapter 1.

One of the most problematic issues with heuristic methods is getting trapped into local optima. If the search is performed by disregarding the search nearby good solutions, the optimum value may never be found because the algorithm is always jumping from one region to another region of the search space. Whereas, if an intensive search nearby good solutions is made then, the algorithm may converge too soon and concentrate all the effort in a poor area. Therefore, if exploration and exploitation are carelessly thought of, one may only by chance get to a region where an optimum solution or a very good solution is to be found. The trade-off between exploration and exploitation of the search space must then reach an equilibrium to take advantage of the great potential of these heuristics in order to achieve good results.

Ant colony algorithms suffer from that same problem, especially due to the number of solutions evaluated at each iteration, which may lead to an early convergence towards a poor solution. Nonetheless, ACO algorithms have a set of parameters that may be set and controlled in order to leave less to chance, as was already discussed in Section 3.3. Hybridization between ACO and other heuristics is also another hypothesis to think of when trying to improve its performance. But then the question is whether to use ACO as the main framework or as an auxiliary mechanism, and that will depend on the problem to be solved and whether the objective is to incentivise exploration or exploitation.

Recently, in (Bernardino et al, 2009), a hybridization between an ant colony optimization and local search is used to solve the Terminal Assignment (TA) problem. In the TA problem there is a set of terminals with an associated capacity and a set of concentrators. The objective is to minimize the distances between concentrators and terminals, provided that each terminal is assigned to a single concentrator and that no terminal capacity is overflowed. The algorithm proposed by the authors is based on that of Gambardella et al (1999b), which was used to solve the quadratic assignment problem. The main difference of this hybrid ACO is that the information about the pheromone quantity laid in each path is used to modify the solutions that were obtained previously, instead of producing new solutions. All the pheromone trails are initialized with the same random amount of pheromone. Pheromones are updated only by the ant that has constructed the global best solution. In each iteration, solutions are modified based on the pheromone quantities, in order to perform an exploration of the search space. After a terminal c, in the solution, has been randomly chosen and a number x randomly generated between 0 and 1, if x is lower than a certain parameter q, then the best concentrator is chosen, that is, the one with the maximum pheromone value. However, if x is higher than q, then the concentrator to which the terminal is to be linked to, is chosen with a probability proportional to the pheromone trail values. The following step is to apply local search, which is performed by using swaps between two randomly chosen terminals and the two concentrators they are linked to. An intensification phase is applied if the best solution found in the search is improved. After the pheromone has been updated, the diversification phase is applied. The diversification is introduced in the algorithm by means of a pheromone trail reinitialization, if the solution is not improved for a certain number of consecutive iterations. From the comparisons of the hybrid ACO with genetic, tabu search, and hybrid differential evolution algorithms it was found that the hybrid ACO was able to obtain better solutions for large problems. Nevertheless, the fastest algorithm was the tabu search.

In (Crawford and Castro, 2006), the authors solve both set covering and set partitioning benchmark problems with ACO algorithms, and with hybridizations between ACO and constraint programming techniques, such as forward checking and full lookahead,

and post processing procedures. In the Set Partitioning Problem (SPP) there is a set C of columns and a set *R* of rows, with each $c \in C$ covering a subset of rows. The objective of the SPP is to minimize the sum of the columns costs provided that each row is covered by one, and only one column whereas, in the Set Covering Problem (SCP) each row must be covered by at least one column. In the ACO solution construction procedure, performed by each ant, each column $c \in C$ is a component of the solution (such as arcs are the components in the TSP problem). Therefore, pheromone is added to components rather than to connections, and the quantity to be deposited is proportional to the ratio between the number of ants using that column in their solution and the number of ants with better solutions. The authors use dynamic, rather than a static, heuristic information that evaluates the partial solution and calculates the cost of covering a single row with the new column. Additionally, a post processing procedure is added to the developed ACO, and it consists of eliminating columns if they cover rows that can be covered by other existing columns and/or replacing columns in the solution by more advantageous columns. In the forward checking technique, each time a column is to be added to the ant solution, it is investigated first if it is prone to create conflicts with the next columns to be chosen, that is, to violate a constraint of the problem. In the full lookahead technique, besides checking the next columns to be chosen, conflicts with the columns already in the solution are also checked. This is very important, specially to the SPP, because it may be used to check the possibility of constructing unfeasible solutions, and to avoid it. Results showed that the hybridization between ACO and Post Processing achieved better results although none of the algorithms was able to reach the best known solution so far.

Rappos and Hadjiconstantinou (2004) use ACO to design two-arcs connected flow networks. In this problem one wishes to find a minimum cost two-arcs connected network satisfying customer flow requirements. The two arcs connected network is such that if an arc is removed from the network, the network is still able to satisfy its customers. The authors refer to this problem as the Reliable Network Design Problem (RNDP). This problem has a fixed and a variable component cost making it harder to solve due to its concave nature. The supply network is seen as a set of paths each of which starting at the source node and ending at one of the customer nodes. This allows for a modification in the representation of the solution. In this algorithm a single ant is not able to find a complete solution for the problem. A solution is only available through the partial solutions of a group of ants. The ants within a group are divided into two types, *flow* ants and *reliability* ants. Initially, the number of *flow* ants is the same as the number of customer nodes in the problem. Each of these ants will find a path from the source node to a customer node. After all the *flow* ants have constructed their solution, *reliability* ants are created. For each *flow* ant solution, the algorithm creates as many *reliability* ants as

the number of arcs in that particular solution. Their goal is to find a path from the same source node to the same customer node provided that the arc, to which the *reliability* ant is associated to, is not used in the solution. This procedure will provide the extra arc needed, whenever one of the solution arcs cannot be used. The pheromone trail will also be of two kinds, *flow* trail (T_f) and *arcs* trail (T_e). The former one is related to the fixed cost incurred by using the arcs, and the latter one relates to the flow passing through the arc. *Flow* ants can *smell* and deposit both types of pheromones. *Reliability* ants can only *smell* and deposit *arc* pheromone because their solution is only related to the extra arc that is included in the solution and that has no flow, incurring only in fixed costs. The amount of pheromone deposited in each arc, for the two types of pheromones, is inversely proportional to the corresponding cost. Pheromone trails are updated when all the ants of the group have finished searching for their part of the solution. When an ant is in a node and wants to decide where to go to next, there are two alternative ways to do it. Rappos and Hadjiconstantinou (2004) use a method to choose the arc to travel similar to the one in the ant colony system. The heuristic value is thus defined as the inverse of the total costs incurred by using the arc, i.e. fixed and variable costs, while the probability value for travelling through an arc depends on the type of ant. Recall that the probability function, as defined in Eq. (3.2) depends solely on the pheromone value and on the heuristic value. *Flow* ants use both types of pheromone to represent the pheromone value in the probability function, see Eq. (3.55), while *reliability* ants can only use *arc* pheromone, T_e.

$$F(T_f, T_e) = (\frac{1}{T_f} + \frac{1}{T_e})^{-1}. \tag{3.55}$$

Networks with 12 up to 50 nodes were solved with the ACO algorithm and gap results were all inferior to 4%, when compared to the optimal values obtained with a commercial solver.

Another ACO algorithm was developed by Shyu et al (2006) to solve the Cell Assignment Problem (CAP). This ACO algorithm incorporates daemon actions, see Section 3.3.3, having a greedy local search approach nature. In the CAP problem there is a set of existing cells to be assigned to one and only one switch, from the set of available switches. Because handoffs, i.e. transfers from one cell to another one, between cells assigned to different switches can occur, handoff costs are associated to every pair of existing cells. Handoff costs are proportional to the frequency of handoffs between cells, which is a number known in advance. Cabling costs are also incurred from linking cells and switches. Costs are defined per time unit. The number of calls that a cell can handle is known as is the handling capacity on the number of calls each switch can cope with. The goal of

the CAP is to minimize the total cost incurred with the assignment of cells to switches, in the conditions specified above. The authors have defined the construction of a solution similar to a walk in the search space, where an ant steps from a cell to a switch and then from that switch to another cell, and so on, defining a continuous walk instead of a set of unrelated arcs. Thus, two decisions concerning the construction of the solution take place and are associated with two different heuristic information matrices. Firstly, when located at a certain cell ants must choose to which switch to move to. This decision is associated to a heuristic information based on the inverse of partial costs. Therefore, as partial costs increase the information heuristic decreases. Secondly, when located at a certain switch ants must choose to witch cell to move to. The heuristic information used in this case is the cell call volume, and it aims at favouring the choice of high call volume cells, i.e. such cells enter first into the solution. The cells and switches available at each step of the solution construction guarantee that ants always construct feasible solutions. Pheromone update takes place at the end of the iteration and only the best ant is allowed to deposit pheromone in its solution path. Whenever a solution S improves the global best solution, a local search is performed in an attempt to improve it further. The algorithm searches for a feasible reassignment for each cell in solution S. The chosen reassignment is the one representing the largest saving in total costs. The results obtained proved to perform better both in terms of solution quality and running time when compared to three other metaheuristics (Simulated Annealing, Tabu Seach, and Memetic Algorithm). However, the ACO algorithm was only able to improve upon the solution quality when compared to three heuristic algorithms, one based upon a greedy strategy and the other two based upon clusters of cells.

In (Bouhafs et al, 2006), the algorithm used to solve the Capacitated Location-Routing (CLR) problem hybridizes an ant colony system with Simulated Annealing (SA). The objective of this problem is to minimize routing and location costs while trying to locate distribution centres (facilities) and satisfying customers demands. Some constraints are defined for the problem: distribution centres have a capacity limit; customers are to be visited only once in order to get their demands satisfied; and each vehicle starts and finishes its route at the same distribution center. The authors solve the problem in two phases, which are repeated until a specified number of iterations is reached. In the first phase, the SA component of the algorithm solves the facility location problem by locating the distribution centres (DC) and by assigning customers to each DC. In the second phase, the best routes are defined by using the ACS. The algorithm starts by randomly choosing one of the distribution centres to be opened while all the others remain closed. If the DC can satisfy all the customers without violating its capacity constraint, the ACS algorithm is then used to find the route associated to the DC configuration providing then a solution,

S, to the CLR problem. This is the algorithm initialization. The location phase, tries to improve the previous solution, with the use of add and swap moves. Swap moves are used to open a closed DC and to close an opened one, if the swapping costs justify it. This operation has the objective of maintaining the number of opened DCs constant. After a possible swap is made, which is always the best one that can be identified regarding the savings on costs, the ACS is performed so that routes are recalculated. Add moves are performed after all the configurations using the same number of DCs have been explored. An add move, as is defined by Bouhafs et al (2006), randomly adds another DC to the solution and assigns to it the customers which are closer to this new DC than to previously existing ones, provided that the DC's capacity is not exceeded. The new location solution S', provided by a swap or by an add move, is then accepted if it represents a lower cost, i.e. if $cost(S') < cost(S)$, and if $e^{-(cost(S')-cost(S))/T} > \mu$, where T is the temperature and $\mu \in [0, 1]$ is a randomly uniformly distributed value. If S' is accepted then, the routing optimization ACS algorithm is performed in order to recalculate the best routes. The ACS algorithm starts by positioning m ants, one at each customer. Since each customer is associated with a particular DC, each ant is also associated with the DC's set of customers. An ant positioned at a given customer constructs a route by choosing the next customer to visit while the capacity of the vehicle is not exceeded. This is performed until all customers have been visited. Bouhafs et al (2006) use local pheromone update during the construction of the solution, i.e., the pheromone is updated while the ant is constructing the solution. The probability distribution used in this algorithm is slightly different from the one in Eq. (3.2), since the savings value γ_{ij}, obtained from visiting customer j after customer i, is also considered:

$$P_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta} \cdot [\gamma_{ij}]^{\lambda}}{\sum_{j \in J_i^k} [\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}^{\beta}] \cdot [\gamma_{ij}]^{\lambda}}, \tag{3.56}$$

where J_i^k is the set of costumers not yet visited by ant k and that by being chosen do not violate any constraint. When all ants have constructed their solution, the pheromone is added to the arcs in the best solution and in the global best solution. These steps are repeated while the temperature is being progressively decreased until it reaches a minimum value. The algorithm stops if the minimum temperature value or if the maximum number of allowed iterations is reached. The authors solved a set of 11 benchmark problems and have compared their results to the previously known. This hybrid algorithm finds better solutions for eight of them and matches the best known for the remaining three problems. Running times are not provided.

The hybridization of an ACO and a Genetic Algorithm (h_GACO) has proven to be quite effective to solve transportation problems with concave cost functions in (Altipar-

mak and Karaoglan, 2007). The ACO procedure is embedded in the GA to create new offspring, which are generated through crossover and mutation. Given two parents x_1 and x_2 the crossover operator is defined such that half of the arcs in x_2 but not in x_1 are selected using the transition rule in order to be inserted into solution x_1. The transition rule used is the usual, but the heuristic value is adapted to the concave nature of the transportation costs taking into account the unit transportation cost $c_{ij}/\sqrt{x_{ij}}$, where c_{ij} is the cost of the arc and x_{ij} is the flow passing through the arc. The heuristic information is defined as $\eta_{ij} = 1/(c_{ij}/\sqrt{x_{ij}})$. The mutation operator, also uses the ACO procedure but only to change one arc in each solution to be mutated. The arc entering the solution must be one of the first 10% of the arcs not present in the solution, sorted in ascending order of unit variable cost. Pheromone trails are limited by maximum and minimum limits, $\tau_{max} = 1/(F(S^b) \cdot \rho)$ and $\tau_{min} = \tau_{max}/(|N| + |M|)$, where ρ is the evaporation rate, N is the set of supply nodes, and M is the set of demand nodes. A reinitialization is performed in the algorithm whenever more than 50% of the pheromone trails are equal to τ_{min} or when the current best solution has not been updated for 50 iterations. In the first case τ_{max} is recomputed using the best solution known and the pheromone matrix is set to τ_{max}. In the second case, 10% of the worst solutions are replaced with randomly generated solutions. The results obtained were compared with the ones obtained by other heuristics in literature (GA, ACO, SA, Local Search Algorithm, Adapted Tabu-Simulated Annealing), for a set of 14 benchmark problems, having as a stopping criterion a limited running time. The results were favourable to the h_GACO heuristic, having outperformed the other methods in all cases but one.

3.3.7. Books and Surveys

Reviews are very important, specially when someone is starting on a new research area. Therefore, we could not finish this section without referring to some of the detailed and comprehensive reviews. For a good review on early ant colony optimization historical applications we refer to (Cordon et al, 2002). The reader may also find the review of Mullen et al (2009) very interesting, since the authors review the application of ant algorithms in fields such as digital image processing, data mining and other machine learning techniques. In this chapter, we have omitted multi-criteria combinatorial optimization problems, because they are not part of the problems we want to study in the work that follows. Also, they are part of a very specific area of investigation because, having to account for more than one objective value implies that there is a set of good solution, called the Pareto Set, that are superior to the remainder. A good work reviewing this type of problems is provided in (García-Martínez et al, 2007), where the authors, besides providing a

survey on previous works also solve a set of instances of the bi-criteria TSP with several ACO algorithms, in order to be able to compare them and discuss their characteristics.

Although a little out-of-date, due to the large number of works that have seen the broad daylight after they have been published, (Bonabeau et al, 1999) and (Dorigo and Stützle, 2004) are still important references regarding ant based algorithms, since they provide excellent explanations on ant algorithms and their evolution. The first book gives us an insight on the general social insect behaviour with particular emphasis on ant algorithms. The second book is fully dedicated to ant colony algorithms, and surveys several applications of ACO in fields such as scheduling, machine learning and bio-informatics. It also discusses some theoretical findings and is an excellent guide to everyone who wishes to implement ant algorithms.

3.4. Summary

In this chapter we have presented an overview of the literature regarding Ant Colony Optimization. Initially, the Ant System algorithm, the first ant algorithm developed to solve combinatorial problems, was described. Some major developments, that have shortly followed this algorithm were also introduced. Then, a literature review was made initially regarding different approaches developed for each component of ant algorithms. Algorithms using several ant colonies were also discussed, as well as hybrid approaches where the ACO was either the major procedure or an auxiliary procedure embedded in order to mainly perform exploitation. A literature survey, based on this chapter, has already been published as a working paper (Monteiro et al, 2012a).

4

Minimum Cost Network Flow Problems

4.1. Introduction

Minimum Cost Network Flow Problems (MCNFP) have a major role in optimization since they have many practical applications, such as supply chains, logistics, transportation and facility location. Costs can take several forms but the ones we are interested in are concave costs, usually associated with economies of scale, discounts, or start-up costs (Guisewite and Pardalos, 1990), which are much more realistic than the linear ones often found in literature. The minimization of a concave function over a convex feasible region, defined by the linear constraints of the problem, makes it much more difficult to solve, therefore more appealing. In this chapter, we concentrate our attention in the study of the special case of the single source uncapacitated nonlinear MCNFP. We start by introducing and discussing the problem definition and formulation. Then, we present an overview and review some methodologies that have been used in recent years in order to address MCNFPs. The last sections are devoted to the heuristic methodology here developed and to the discussion of the results achieved and reported here.

4.2. Concave Minimum Cost Network Flow Problems

Let us start by defining the concave minimum cost network flow problem. This problem is a special case of the MCNFP class where concave cost functions are considered. Concave cost functions arise in many economical situations where economies of scale apply, i.e., when the increase in the input leads to an increase on the overall costs but with a marginal cost decrease. A frequent example of this situation is the toll charge. A vehicle pays its toll charge regarding the vehicle class to which it belongs, whether it travels at its full capacity or not. Therefore, the toll cost per unit transported decreases with the increase of the number of transported items. Other examples of situations where concave cost functions have to be accounted for include the setting of networks of facilities, such as a network of bank branches, that besides the initial costs incurred with the opening of facilities and equipment have also to include operating costs, see (Monteiro and Fontes, 2006).

A minimum cost network flow problem with a general concave cost function can be formally defined as follows.

Given a directed graph $G = (N, A)$, where N is a set of n nodes and A is a set of m available arcs (i, j), with $i \in N$ and $j \in N$, a concave minimum cost network flow problem is a problem that minimizes the total concave costs g_{ij} incurred with the network while satisfying the nodes demand d_j.

Considering the notation summarized bellow,

n - number of nodes in the network
m - number of available arcs $(i, j) \in A$
d_j - demand of node $j \in N$
x_{ij} - flow on arc $(i, j) \in A$
y_{ij} - decision variable assuming the value 1 if arc $(i, j) \in A$ is chosen and 0 otherwise
g_{ij} - concave cost of arc $(i, j) \in A$
u_{ij} - upper limit on flow through arc $(i, j) \in A$
l_{ij} - lower limit on flow through arc $(i, j) \in A$

the mathematical model for the concave MCNFP can then be written as follows in Model 1.

The objective is to minimize the total costs defined in (4.1), provided that the demand is satisfied, stated by the *flow conservation constraints* (4.2), and that the arcs *capacity*

Model 1 A mixed-integer mathematical programming model for the general concave MCNFP.

$$\text{min:} \quad \sum_{(i,j)\in A} g_{ij}(x_{ij}, y_{ij}) \tag{4.1}$$

$$\text{s.t.:} \quad \sum_{\{i|(i,j)\in A\}} x_{ij} - \sum_{\{k|(j,k)\in A\}} x_{jk} = d_j, \forall_{j\in N}, \tag{4.2}$$

$$0 \leq x_{ij} \leq u_{ij}, \quad \forall_{(i,j)\in A}, \tag{4.3}$$

$$x_{ij} \geq l_{ij} y_{ij}, \quad \forall_{(i,j)\in A}, \tag{4.4}$$

$$y_{ij} \in \{0,1\}, \quad \forall_{(i,j)\in A}. \tag{4.5}$$

constraints in (4.3) and (4.4) are not violated. Constraints (4.5) refer to the binary nature of the decision variables y_{ij}. Regarding the demand, d_j takes a negative or positive value depending on whether j is a source or a demand node, respectively. We assume that the total source demand equals the total sink demand, thus $\sum_{j\in N} d_j = 0$. Sometimes neither upper nor lower bounds are established for the flows in the arcs, therefore the problem is considered uncapacitated which mathematically is translated as $u_{ij} = +\infty$ and $l_{ij} = 0$.

Regarding concave cost functions, they can take several forms but the most popular ones used in literature are $b_{ij} \cdot x_{ij} + c_{ij}$, also known as fixed-charge functions. A square root cost function, $b_{ij} \cdot \sqrt{x_{ij}}$, has also been considered in (Altiparmak and Karaoglan, 2006), $-a_{ij} \cdot x_{ij}^2 + b_{ij} \cdot x_{ij}$ is the cost function used in (Dang et al, 2011), while Fontes and Gonçalves (2007) use $-a_{ij} \cdot x_{ij}^2 + b_{ij} \cdot x_{ij} + c_{ij}$, and Burkard et al (2009) define the cost function as $-a_{ij} \cdot x_{ij}^{\alpha} + b_{ij} \cdot x_{ij} + c_{ij}$ with $\alpha \in [0,1]$, just to mention but a few possibilities.

MCNFPs with linear costs are solvable in polynomial time, that is, they are considered easy to solve, see Section 2.3. As both the cost function and the constraints are linear they can be solved by linear programming methods, such as the simplex method. When costs have a nonlinear nature then the complexity of the problem may increase exponentially. This is the case of the concave MCNFP which is known to belong to the class of NP-Hard problems, (Guisewite and Pardalos, 1991a).

4.2.1. Characteristics of Concave MCNFPs Solutions

Some properties and results have already been established regarding MCNFPs with concave costs, which can help us to characterize a solution for this problem.

Let us start by defining a feasible solution (or feasible flow) in this context. A feasible solution for the concave MCNFP is a solution that does not violate neither Eq. (4.2) nor

Eq. (4.3). The set of all feasible solutions is denominated by the feasible region of the problem. Among all feasible solutions, there are some that cannot be obtained by a linear combination of any of the other solutions. These special solutions are called extreme solutions, also known as local optimal solutions.

Lozovanu (1983) observed that if a feasible solution exists for concave MCNFPs, then an optimal solution must occur at a vertex, i.e. an extreme point, of the convex polyhedron defined by the problem constraints (4.2) and (4.3). Also, since it involves the minimization of a concave cost function in a convex polyhedron, a local optimum may not be a global optimum. Thus, in order to find the global optimum solution for this problem, the set of all extreme points in the convex polyhedron has to be searched for.

Furthermore, if the function has a finite global minimum on the feasible region, then there is an extreme solution that is an optimal solution (Eggleston, 1963).

4.2.2. Complexity Issues

Establishing what really defines the complexity of an optimization problem is a very important issue mainly because it will allow the researcher to choose an adequate method to solve it. For example, if the MCNFP instance to be solved is considered easy, an exact method, such as simplex or branch-and-bound, can be used, whereas if it is considered hard then a heuristic method is probably more adequate as it can provide a fairly good solution in a reasonable amount of time.

It follows a discussion of the main characteristics of MCNFPs that define its complexity. The intention is not to provide the mathematical proofs for the results, as they can be found in the references provided within this section, but rather to provide an insight on the subject.

Fixed-charge and Variable Costs: the special case of the fixed-charge problem

Problems with fixed-charge costs are a special case of concave optimization. These functions are characterized by a discontinuity at the origin which is caused by considering a fixed cost for using an arc besides this routing cost, which is proportional to the flow passing through the arc. The special case of the single source uncapacitated MCNFP with fixed-charge costs has been proven to be NP-hard (Hochbaum and Segev, 1989). Furthermore, fixed-charge problems may be simpler or harder to solve accordingly to characteristics that have been argued by some authors. One such characteristic is the ratio between fixed (F) and variable (C) costs $\frac{F}{C}$. On the one hand, Kennington and Unger

(1976) claim that the difficulty to solve fixed-charge problems increases with this ratio. On the other hand Palekar et al (1990), by observing the results obtained by both (Kennington and Unger, 1976) and (Barr et al, 1981), disagree with them suggesting that only ratios with intermediate values are difficult to solve. Their reasoning is that if the ratio is very small or very large the problem is easier to solve either because fixed costs are negligible thus transforming the problem into a linear one, or because the problem reduces to the one of minimizing fixed costs. The problem with such a classification is that the true value of these ratios can only be calculated after the problem has been solved, although some approximations have been proposed. Nonetheless, Palekar et al (1990) proposed a *difficulty ratio* that can be calculated *a priori*. The difficulty ratio (DR) depends on the sum of all demands D, on the average of the variable $\bar{c}$ and of the fixed $\bar{f}$ costs, and on the number of source n_1 and demand nodes n_2, and it is given as $DR = \bar{f}(n_1 + n_2 - 1)/\bar{c}D$. The interested reader is referred to Palekar et al (1990) for more details.

The Density of the Network

The density of a network is usually measured by the ratio of the available and of the potential existing arcs in a network. It comes straightforward that the closer this measure is to 1 the denser is the network, whereas the closer it is to 0 the sparser the network is. Therefore, it is easy to conclude that the denser the network the harder the problem is to solve, because the number of feasible solutions increases and so does the computational time needed to enumerate all of them in case of an exact method.

Capacitated Arcs versus Uncapacitated Arcs

Regarding the capacity of arcs in a network, both versions of the concave MCNFP, capacitated and uncapacitated, are known to be NP-hard. But, although capacitated problems are usually harder to solve, there are some methods that can benefit from them. Such is the case of linear underestimation methods that can benefit from the capacity constraint by using it to improve the precision of the linear estimation in the subintervals defined for the linear approximation.

The Density of Nonlinear Arcs

The number of nonlinear cost arcs is a major factor affecting the difficulty to solve a nonconvex MCNFP (Tuy et al, 1995a). The larger the number of nonlinear arcs, the harder the problem becomes. Some problems with a small number of nonlinear arc costs

have been proven to be solvable in polynomial time see, e.g., (Guisewite and Pardalos, 1993).

Another characteristic affecting the difficulty of concave MCNFPs is the concavity of the cost function. Guisewite and Pardalos (1991b) provide a study on how the form of the concavity affects the complexity of these problems. The authors use functions with the following form $\alpha_{ij}x_{ij}^{\beta_{ij}}$. They were able to provide evidence that on the one hand, the number of local optima increases with the decrease of β_{ij}, and on the other hand, that the larger the set from which to draw the value of α_{ij}, the smaller the set of local optima.

The Density of Demand Nodes

Four types of nodes can be defined in a network problem. Source nodes, which are the nodes from where some commodity is to flow from and thus, they have a negative demand value. Transshipment nodes, which are nodes with no demand that may be incorporated in the network, if it contributes to lowering the costs. Demand nodes and sink nodes, which are nodes to which the commodity has to be delivered to and thus, their demand has a positive demand value. The difference between them being that the former can act also as transshipment nodes, while the latter are terminal nodes. In network flow problems, demand nodes are usually the ones contributing to the complexity of a problem. In addition, problems with several source nodes can be transformed into problems with a single source node. Therefore, the size of a problem, and consequently one of the many aspects contributing to the difficulty in solving it, is usually related to the number of demand nodes.

4.3. The Single Source Uncapacitated Concave Minimum Cost Network Flow Problem

We now define a special case of MCNFPs, the Single Source Uncapacitated Minimum Cost Network Flow Problem (SSU MCNFP). Let us start by pointing out the differences towards the main class of MCNFP.

Firstly, this problem does not impose a constraint on the flow passing through the arcs of the network other than being nonnegative. Therefore, this in an uncapacitated version of the MCNFP. The imposition of some capacity in the arcs of the network may make sense in situations where, e.g., the number of trucks available for the distribution of the commodity is restricted, thus limiting the availability of capacity. But, if the number of

trucks that can be used is unrestricted, then we can send an unlimited amount of commodity, and this is the case that is being considered in this work.

Secondly, we consider the existence of a single source node from where the commodity flows and that all other nodes are demand nodes. In this case, no transshipment nodes exist thus, the set of nodes in the network is made up of one source node and the rest are considered demand nodes. The case of a single source is a common situation as it may represent a single warehouse or factory where commodities have to flow from in order to get to their final destination, i.e., customers. A single source can also represent a starting point in time in production and inventory planning models, where the total production for a specific time window can be represented by the total flow in the model, and where sinks may represent intermediate production requirements for the time window considered, see e.g., (Guisewite and Pardalos, 1991a).

Concave MCNFPs have the combinatorial property that if a finite solution exists, then there exists an optimal solution that is a vertex (extreme point) of the corresponding feasible domain (defined by the network constraints). Single source uncapacitated concave minimum cost network flow Problems have a finite solution if and only if there exists a direct path going from the source to every demand node and if there are no negative cost cycles, otherwise an unbounded negative cost solution would exist (Lozovanu, 1983). Therefore, for the SSU MCNFP, an extreme flow is a tree rooted at the single source spanning all demand nodes. For detail and proofs see (Zangwill, 1968).

The special case of the fixed-charge SSU MCNFP has been proven to be NP-hard in (Hochbaum and Segev, 1989). Later on, Guisewite and Pardalos (1990, 1991a) also proved that the general SSU concave MCNFP is NP-Hard for acyclic networks where all nodes have bounded total degrees less than or equal to 3.

However, a few special cases of concave MCNFPs have been proven solvable by polynomial-time and strong polynomial-time algorithms, such as: the SSU concave MCNFP with only one demand node (Zangwill, 1968); the SSU MCNFP with a single nonlinear concave arc cost (Guisewite and Pardalos, 1993; Klinz and Tuy, 1993); the SSU MCNFP with two nonlinear concave arc costs and the MCNFP with two sources and one single nonlinear concave arc cost (Tuy et al, 1995b); the SSU MCNFP with a fixed number of sources and a fixed number of nonlinear concave arcs (Tuy et al, 1995a).

Tuy et al (1995a) and Tuy (2000) state that the difficulty of concave MCNFPs depends on the number of arcs with nonlinear costs and prove the strong polynomial-time solvability of SSU MCNFP problems where only a subset of k arcs have concave cost functions, while the remaining ones have linear cost functions. Nonetheless, in Horst and Thoai (1998) it has been empirically shown, by using the same problem, that the difficulty

in solving concave MCNFPs increases with the number of arcs having concave costs.

In summary, the SSU concave MCNFP, where all arcs have a nonlinear concave cost, is a NP-hard problem that can be used to model a larger number of practical applications, especially in economics. Hence, our motivation to study and to solve it in this thesis.

4.3.1. Problem Definition and Formulation

The SSU concave MCNFP considers the establishment of a network between a single source, from where the commodity is to flow, and a set of demand nodes, at minimum cost provided that the node demands are satisfied. As we are considering the uncapacitated version of the problem, there are no bounds on the arcs capacity. Formally, the problem can be defined as follows.

Consider a directed graph $G = (N, A)$, where N is a set of $n + 1$ nodes, with one source node t and n demand nodes $j \in N \setminus \{t\}$, and $A(\subseteq N \times N \setminus \{t\})$ is a set of m available arcs (i, j). Since there is only one source node, the number of available arcs m is at most $n \cdot (n + 1)$. Consider a commodity that flows from the single source t to the n demand nodes $j \in N \setminus \{t\}$, each node requiring a demand d_j to be satisfied. Let the decision variables x_{ij} be the amount of flow routed through arc (i, j) and y_{ij} be a binary variable assuming the value 1 if arc (i, j) is chosen and 0 otherwise. Consider $g_{ij}(x_{ij}, y_{ij})$, representing the cost incurred with arc (i, j), is given by the sum of the cost of using arc (i, j) with the cost of routing flow x_{ij} through it. A single source uncapacitated concave minimum cost network flow problem is a problem that minimizes the total costs $g(X, Y)$ incurred with the network while satisfying the demand of all nodes.

Considering the notation summarized bellow,

t	-	source node
n	-	number of demand nodes
m	-	number of available arcs in the network
d_j	-	demand of node $j \in N \setminus \{t\}$
x_{ij}	-	flow on arc (i, j)
y_{ij}	-	decision variable assuming the value 1 if arc $(i, j) \in A$ is chosen and 0 otherwise
g_{ij}	-	cost of arc (i, j)

the mathematical model for the single source uncapacitated concave minimum cost network flow can then be written as follows:

Model 2 A mixed-integer mathematical programming model for the SSU MCNFP.

$$\text{min:} \quad \sum_{(i,j)\in A} g_{ij}(x_{ij}, y_{ij}) \tag{4.6}$$

$$\text{s.t.:} \quad \sum_{\{i|(i,j)\in A\}} x_{ij} - \sum_{\{k|(j,k)\in A\}} x_{jk} = d_j, \quad \forall_{j\in N\setminus\{t\}}, \tag{4.7}$$

$$x_{ij} \leq My_{ij}, \quad \forall_{(i,j)\in A}, \tag{4.8}$$

$$x_{ij} \geq 0, \quad \forall_{(i,j)\in A}, \tag{4.9}$$

$$y_{ij} \in \{0,1\}, \quad \forall_{(i,j)\in A}. \tag{4.10}$$

The objective in this problem is to minimize the total costs incurred with the network as given in Eq. (4.6). Constraints (4.7) are called the *flow conservation constraints*. The first term of these constraints, $\sum_{\{i|(i,j)\in A\}} x_{ij}$, represents the flow entering node j and the second term, $\sum_{\{k|(j,k)\in A\}} x_{jk}$, represents the flow leaving node j. Therefore, the flow conservation constraints state that the difference between the flow entering a node and the flow leaving the same node must be the demand of the node. Constraints (4.8) guarantee that no flow is sent through an arc, unless it has been chosen as part of the network. Here, M is a positive large number (say $M \geq \sum_{j\in N\setminus\{t\}} d_j$). Constraints (4.9) and (4.10) refer to the nonnegative and binary nature of the decision variables.

We assume that the commodity stored at the source node t equals the sum of all the demands $d_j \in N \setminus \{t\}$, that is,

$$\sum_{j\in N\setminus\{t\}} d_j + d_t = 0, \tag{4.11}$$

where d_t is the demand of node t. As t is a source node, its demand is represented by a negative value, hence the form of Eq. (4.11).

4.3.2. Solution Characteristics

Let us start by defining a feasible solution in this context. A solution to the MCNFP is considered feasible if and only if it satisfies both the arc capacity constraints and the flow conservation constraints.

An uncapacitated MCNFP has a finite solution if the solution satisfies the following characteristics:

- The solution has a single path coming from the source(s) node(s) to each of the demand nodes.
- There is no negative cost loop, otherwise an unbounded negative cost solution would exist.

These observations are very important to distinguish feasible solutions from all the other solutions when defining a solution methodology, especially if we are using heuristic solution methods. But, how to distinguish between a feasible solution (flow) from an extreme solution, i.e. a local optimum? If we can find a feasible solution that is not the convex combination of any other feasible solution, then we have found our extreme solution. Furthermore, Eggleston (1963) established another very important result. If the objective function has a finite global optimum on the feasible region, then there exists an extreme flow that is an optimal flow.

Proposition 2. *For a single source uncapacitated concave MCNFP a flow is an extreme flow if and only if it is a tree rooted at the single source, that is, if it is an arborescence, (Zangwill, 1968).*

4.3.3. Transforming MCNFPs into SSU Concave MCNFPs

In this section, we present some transformations that can be made to MCNFPs in order to transform them into single source uncapacitated concave MCNFPs. Thus, proving that this special case can actually be used for addressing general concave MCNFPs.

Please note that it is not our objective to present the proofs of the following propositions. However, references are provided for the interested reader.

Let us start by converting the *multi source* problem into the *single source* problem.

Proposition 3. *Every multiple source (uncapacitated) concave MCNFP defined on a graph $G = (N, A)$ with n nodes, from which s are source nodes, and m arcs can be transformed into a single source (uncapacitated) concave MCNFP on an expanded graph $G = (N', A')$ with $(n + 1)$ nodes and $(m + s)$ arcs.*

This proposition has been proven by Zangwill (1968). The transformation is made by adding a fictitious source t and s arcs linking each of the original sources to the fictitious source t. The original source will have a zero demand.

Next, it follows the transformation of a *Capacitated* problem into an *Uncapacitated* one, due to Wagner (1958).

Proposition 4. *Every capacitated concave MCNFP on a graph $G = (N, A)$ with n nodes and m capacitated arcs can be transformed into an equivalent uncapacitated concave MCNFP on an expanded graph $G = (N', A')$ with $(n + m)$ nodes and $(m + m)$ arcs.*

In order to convert a capacitated problem into an uncapacitated one, each capacitated arc (i, j) with lower and upper capacities given by $0 \leq l_{ij} \leq u_{ij} \leq \infty$ must be replaced by a source node s and two uncapacitated arcs (s, i) and (s, j). The flow of the new arcs is given by $x_{si} = u_{ij} - x_{ij}$ and $x_{sj} = x_{ij} - l_{ij}$, and the cost is given by $g_{si}(x_{si}) = 0$ and $g_{sj}(x_{sj}) = g_{ij}(l_{ij} - x_{sj})$, respectively. The demands of nodes i and j is now given by $d_i + u_{ij}$ and $d_j - l_{ij}$, respectively, where d_i represents the demand of node i.

Finally, Lamar (1993) discusses the transformation of general nonlinear cost functions into concave cost functions. Problems with general nonlinear cost functions are harder to solve, because neither the characteristics of concave problems nor the characteristics of convex problems may be explored in order to develop a solution approach.

Proposition 5. *Every MCNFP defined on a graph $G = (N, A)$ with all or some arcs having general nonlinear cost functions can be transformed into a concave MCNFP on an expanded graph $G = (N', A')$.*

This is a more recent result obtained by Lamar (1993). The main idea is to start by, initially, substituting the general nonlinear cost function by a piecewise linear function having the same value for integer values of the arc flow. In this piecewise linear function, each segment must be either concave or convex. Then, arcs are rearranged such that all piecewise linear convex functions are joined together and are replaced by a single piecewise linear convex function. The same is true for all piecewise linear concave functions. Furthermore, a result by Jensen and Barnes (1980) allows an arc with a piecewise linear convex cost with k segments to be converted into a set of k parallel arcs with linear costs. Therefore, each arc of the original graph having a nonconvex and nonconcave cost function with k segments can be converted into an arc having a concave cost in series with a set of parallel arcs each having a linear cost.

Having presented these results we can now characterize the solutions of SSU concave MCNFP from the ones we know which are already established for concave MCNFP. It is known that the minimization of a concave function over a convex polyhedron defined by the network constraints has a large number of local optima that are not global optima. It is then important to know the characteristics of the problem, as well as, of the solutions in order to help in the construction of a solution methodology.

4.4. Solution Methods for MCNFPs

Most of the works developed around concave MCNFPs consider fixed-charge costs, that is cost functions having a fixed cost component and a linear routing component. Other works considering nonlinear concave routing costs (Guisewite and Pardalos, 1991a; Horst and Thoai, 1998; Smith and Walters, 2000) do not include a fixed component.

As far as we are aware of, only a few works consider concave cost functions made of nonlinear concave routing costs and fixed costs simultaneously, which are those by Burkard et al (2001),Fontes et al (2003, 2006b,a); Fontes and Gonçalves (2007), and Dang et al (2011). This is the main reason why the review of previous works is mainly on the fixed-charge problem.

As it has been mentioned before, SSU concave MCNFP belong to the class of NP-hard problems. Exact solution methods are usually not very efficient in these cases, because they make use of implicit or explicit enumeration of the vertices (local minimums) of the convex polyhedron defined by the flow conservation constraints. Nonetheless, exact methods are very important in the sense that they provide us with optimal values, even if it is only for small problem instances, and also due to the theoretical advances they unravel.

The most popular methods to solve NP-hard problems are heuristic methods. Low usage of computational memory and computational time are their most attractive characteristics, although they may provide only a local optimum solution. Heuristic methods may be classified in various ways, but here we distinguish between them based on the number of solutions evaluated: single-point or multi-point algorithms. Generally speaking, a single-point algorithm evaluates a single solution in each phase of the search. These algorithms are usually coupled with a local search procedure in order to provide a better solution. Examples of such heuristics are Simulated Annealing and Tabu Search. Multi-point heuristics, in opposition, analyse a set of solutions in each phase/iteration and usually combine the best parts of some solutions in order to create new solutions. Examples of these are evolutionary algorithms, such as genetic algorithms, and ant colony optimization algorithms. Furthermore, hybrid algorithms are also very popular because they usually join forces between methods focused in the exploration of the search space and methods, such as local search, more focused in the exploitation of the search space.

In the following sections we review works on each of the above mentioned categories of solution methods.

4.4.1. Exact Solution Methods

The most common technique associated with exact methods to solve MCNFPs is Branch-and-Bound (BB). Branch-and-bound procedures are very popular in the fields dedicated to the development of exact algorithms. Usually, these procedures start with a main problem dividing it into smaller subproblems which, in turn, are further divided into smaller subproblems, and so on. For example, in binary programming this is usually achieved by fixing the value of some variable to 1 for one subproblem and fixing it to 0 for the other. This is called the branching phase. For each subproblem A, upper and lower bounds to the true cost function value have to be computed, which is known as the bounding phase. In BB a subproblem is also called a node of the branching tree. If the lower bound value to subproblem A is worse than any upper bound of some other subproblem B, then A is said to have been conquered (fathomed) and A can be safely discarded, also known as pruned. This means that A will no longer be included in the branching process. These are the main concepts of BB.

However, a major drawback can easily be perceived for this method, it may need hundreds and hundreds of branching steps. This is why fathoming takes such a major role in BB. Fathoming can be achieved in three different ways: by discarding a node with a worse bound, when compared with the incumbent solution; when the relaxation of the problem (used to calculate bounds) has no feasible solutions; or if the solution to the relaxation is feasible and better than that of the incumbent solution. Branch-and-bound algorithms differ from each other in the techniques that are used in its three phases.

Cutting planes are commonly used to try to overcome the problem of extensive branching. The idea of cutting planes is to add a constraint to the relaxation of the original problem in order to reduce the feasible region, without eliminating any feasible solution to the original problem. Therefore, this method is commonly used in BB in order to refine the calculation of bounds for the solution cost, thus reducing the number of extreme points to be searched for.

Given these basic concepts, we now review some exact methods that were developed through the years, in order to solve MCNFPs.

The branch-and-bound procedure developed by Soland (1974), is still very popular and used by other authors as a basis for their own branch-and-bound methods. The idea is to use linear underestimation by convex envelopes and to use rectangles defined by the capacity flow constraints to partition the search space. In Soland's algorithm the branching procedure starts by considering the rectangle defined by the capacity flow constraints C. Then, a subset $C^a \subset C$ is partitioned into two subrectangles C^b and C^c such that

$C^b \cup C^c = C^a$. This way, a subproblem at a node b has its domain defined by both the rectangle C^b and the flow constraints. The bounding procedure corresponds to the computation of a lower bound on the optimal solution found in the subrectangle C^a. This lower bound is obtained by solving a linear relaxation of subproblem C^a. Horst and Thoai (1998) consider the capacitated version of concave MCNFPs where a fixed number of arcs have concave flow costs and the other arcs have linear costs. A BB algorithm based on the work of Soland (1974) is developed leading to improvements of lower bounds. This algorithm differs from Soland's in two ways: in the way rectangles are subdivided, turning them into integral rectangles of approximately the same size, and in the way branching arcs are chosen, in this case from the set of arcs with nonlinear costs. Problems with 20, 40 and 50 nodes are solved considering three concave cost functions for some of the arcs in the problem. It was found that as the number of nonlinear arc costs increases the computational time increases at an astonishing rate. A survey on MCNFPs with a fixed number of arcs with nonlinear costs can be found in (Tuy, 2000).

In (Gallo et al, 1980) a BB algorithm is developed to solve SSU concave MCNFPs. In the problems to be solved the authors consider nonnegative separable concave cost functions for all arcs, however only some of the n nodes are demand nodes, the others being merely transshipment nodes. The BB algorithm initially starts with only the source node and the branching part of it is performed by adding arcs extending the current subtree. Then, lower bounds are obtained for each BB node by using linear underestimation of the arcs costs for demand nodes not satisfied. The authors solve one problem with 4 and another with 10 demand nodes and consider that all arc costs are given by $\alpha_{ij} \cdot x_{ij}^{\beta}$. Latter on, Guisewite and Pardalos (1991a) improve these lower bounds by projecting them on the cost of extending the current path.

Another BB procedure is proposed in (Fontes et al, 2006a) to optimally solve SSU concave MCNFPs considering fixed-charge and nonlinear concave second-order complete polynomial cost functions. At each node of the BB procedure, a lower bound for the cost of the solution is found by making use of a modified relaxation of the state space of the DP developed in (Fontes et al, 2006b). The relaxation only guarantees that the number of used arcs is the correct, i.e. $n - 1$ where n is the number of nodes. However, any arc may be used several times. The BB procedure is as follows. Given a lower bound solution, a branching arc (i, j) is chosen, and two branches are identified and analysed, one where the arc is deleted from the solution and the other where it is forced to be in the solution. If, when (i, j) is deleted from the solution, any demand node is disconnected from the solution tree, then that branch is discarded, otherwise lower and upper bounds are obtained. After analysing that branch, the algorithm steps into the other branch, where (i, j) is fixed as part of the solution and again upper and lower bounds are calculated. The

choice of the BB tree node to go to next is made by selecting the node with the largest gap between the corresponding lower bound and the best upper bound available. An upper bound is computed as explained in (Fontes et al, 2003), see Section 4.4.2 .

Branch-and-cut algorithms are hybrids between the Branch-and-bound technique and cutting planes. In these algorithms the problem is initially solved without the integer constraint and, while the optimal solution has non-integer variable values, a cutting plane algorithm is applied to identify linear constraints that, although violated by the non-integer variable, are satisfied by all the other variables. These constraints are added to the problem which is solved once more. These two steps are repeated until an integer solution is found or until no more cutting planes can be identified. If no more cutting planes can be applied to the subproblem, the algorithm continues with the branch-and-bound procedure and new cutting planes are then identified for the subproblems generated.

Ortega and Wolsey (2003) solve the uncapacitated fixed-charge network flow problem with a branch-and-cut algorithm by extending the cutting planes previously used to solve uncapacitated lot sizing problems, and applying them to a commercial optimization routine of software Xpress. The problem is formulated as a mixed integer problem where binary variables y_{ij}, associated to the use of arc (i, j) are considered. Four dicut-inequalities are defined as follows: simple dicut, mixed dicut, simple inflow-outflow, and mixed dicut with outflow inequalities. However, only dicut-inequalities and simple inflow-outflow inequalities are used, due to their performance in preliminary tests. Another feature therein introduced was the use of a dynamic active node set list for the dicut inequalities. Single-commodity and multicommodity problems have been solved.

Linearisation of the cost function is also very popular among the methods used to solve fixed-charge problems when formulated as mixed-integer problems.

In (Kennington and Unger, 1976) a linear relaxed version of the fixed-charge transportation problem is used. In it, the integrality constraints are relaxed and the usual fixed-charge objective function is replaced by $d_{ij} \cdot x_{ij}$ with $d_{ij} = c_{ij} + f_{ij}/u_{ij}$, where u_{ij} represents the flow capacity of arc (i, j). This relaxation is used to obtain bounds for the solution of the original problem, which are later strengthened using Driebeek penalties (Driebeek, 1966) which are used in the branching and fathoming phases of a BB algorithm.

A recent work on fixed-charge MCNFPs is that of Rebennack et al (2009), where the authors propose a continuous bilinear formulation from which an exact algorithm, based on the algorithm developed earlier in (Nahapetyan and Pardalos, 2007, 2008), is derived. The fixed-charge function is modified by introducing binary variables y_{ij}, defined for all arcs, that take the value 1 if x_{ij}, the flow on arc (i, j), is between a given small value ϵ_{ij}

and the capacity of arc (i, j), and 0 if x_{ij} is between 0 and ϵ_{ij}. The relaxation of these variables results on a continuous bilinear network flow problem with the following cost function

$$f(x, y) = \sum_{(i,j)\in A} \left(c_{ij}^{\epsilon_{ij}} x_{ij} + \left(s_{ij} - \frac{s_{ij}}{\epsilon_{ij}} x_{ij} \right) y_{ij} \right), \tag{4.12}$$

where A is the set of all available arcs, c_{ij} is the variable cost of arc (i, j) ,and s_{ij} is the fixed cost of arc (i, j). The algorithm defined for this new formulation proved to converge in a finite number of steps.

Another work on Fixed-Charge (FC) problems is the one of (Adlakha et al, 2010), where the authors make use of the relaxation of the binary constraints on the y_{ij} value, which was initially proposed in (Balinski, 1961). The optimal solution of this relaxation, has the property that the value per unit flow in each arc becomes

$$C_{ij} = c_{ij} + \frac{f_{ij}}{min(s_i, d_i)}. \tag{4.13}$$

The relaxed problem becomes a linear one. The objective function value of the optimum solution for this new problem provides the FC with a lower bound to the total flow costs, while the objective function value for the FC provides an upper bound. Then, based on the differential of the fixed costs for the FC and for the relaxed problem, the algorithm iteratively chooses demand nodes to be provided with all their demand by a single supply, adjusting the rest of the network by eliminating the most expensive arc. The bounds are then tightened until an optimum value is reached and both bounds have the same value. Although the authors provide a numerical example to illustrate the branching procedure, they do not provide computational results.

Fontes et al (2006b) use an exact method involving DP to optimally solve SSU concave MCNFPs with four cost functions: linear, fixed-charge, and second-order polynomials both with and without a fixed-charge component. The state space graph is gradually expanded by using a procedure working in a backward-forward manner on the state space graph. The dynamic part of the algorithm is related to the identification of only the states needed for each problem being solved. The DP procedure has as many stages as the number of nodes $n + 1$ in the problem to be solved, and each stage is made up of a set of states $s_i \equiv (S, x)$ such that each state considers a subset S of the set of nodes W and some root node x, with $x \in S$. Therefore a stage is given by the cardinality of S. The algorithm starts from the final state where all demand nodes are considered along with the root node t, (W, t). Then, it moves backwards, until some state already computed is reached, identifying possible states in the way. Then, it moves forward, through already computed states, until a not yet computed state (S, x) is reached. The algorithm continues

this backward-forward procedure until the last stage (W, t) is reached and no more moves can improve its cost, and thus an optimal solution has been found.

It is not the purpose to explore further exact methods, since the methods we are interested in are heuristic methods. The latter will be the subject of the next section.

4.4.2. Heuristic and Approximate Solution Methods

In the previous section, we reviewed methods, to solve concave NFPs, that guaranteed to find an optimal solution. Now, in this section, we review approximation methods, that is the ones that do not guarantee an optimal solution, although sometimes they can find it. These methods are divided into two classes: heuristic algorithms, developed to find local optima; and methods to solve approximations of the original problem, generally in order to find upper and/or lower bounds for the optimum solution. Approximation algorithms are based on several techniques: metaheuristics, linearisation of the cost function, reformulations of the problem, branch-and-bound, and hybrids between the former techniques. In this section, we review works of all these classes. No special order is intended, although when grouped by solution technique they are sorted in chronological order.

Burkard et al (2001) develop a dynamic programming algorithm, to solve the SSU concave MCNFP, based on linear approximations of the cost function, where concave costs are given by $c + bx_{ij} + ax_{ij}^d$, with $d \in [0, 1]$ and where a, b, c, and d might or might not depend on the arc (i, j). The authors develop a DP algorithm to solve it and prove that with the use of approximated linear cost functions the method converges towards an optimal solution. The method is only adequate to networks where nodes have small degrees. Therefore, although they are able to solve problems with 1103 nodes they may only have up to 2203 arcs.

Kim and Pardalos (1999) developed a technique called Dynamic Slope Scaling (DSS) in order to solve the well-known NP-hard fixed-charge network flow problem. Given an objective function of the type $f(x) = \sum_{(i,j)}(c_{ij}x_{ij} + s_{ij})$, where c_{ij} represents the flow variable cost, and s_{ij} represents the fixed cost, the idea behind it is to find a linear factor that can represent the variable and fixed costs at the same time. In order to find the linear factor a slope and an initial value have to be determined. The slope of the line connecting the origin and $(\hat{x}_{ij}, f(\hat{x}_{ij}))$ is given by

$$\bar{c}_{ij}(\hat{x}_{ij}) = c_{ij} + s_{ij}/\hat{x}_{ij}, \tag{4.14}$$

if the arc has a positive flow, and by a very large number, otherwise. The initial value

for the slope can be set to either $\bar{c}_{ij}^0 = c_{ij}$, and fixed costs are disregarded, or to $\bar{c}_{ij}^0 = c_{ij} + s_{ij}/u_{ij}$, where u_{ij} represents the upper limit on the capacity of arc (i, j). The former initial slope value can make the algorithm return a solution far away from the optimum value, but it is known that it will adjust itself quickly when the fixed costs are incorporated on the cost function in the following iterations. When the initial slope is calculated, CPLEX is used to solve the new linear formulation obtained finding an initial solution, let us say $\bar{x}_{ij}^0$. The cost function to be used in the next iteration is then updated by substituting $\hat{x}_{ij}$ in Eq. (4.14) by $\bar{x}_{ij}^0$, for arcs with a positive flow. Two alternative updating schemes were provided when $x_{ij} = 0$, either $\bar{c}_{ij}$ is updated to the maximum or to the latest value obtained, in the history of the iterations, when a positive flow occurred in the previous iteration. The algorithm follows these two steps, solving the linear problem and updating the cost function, until two consecutive iterations return the same solution. The DSS procedure was tested on small, medium and large scale problems. It was proven then that the second updating scheme had a better performance for large scale problems, although the solutions found were still far away from the optimal ones. Nonetheless, the DSS had a very good behaviour for the other sizes. Later on, Kim and Pardalos (2000) extend the use of DSS by incorporating a local search scheme, called Trust Interval, to solve concave piecewise linear NFPs. An adaptation of the DSS technique, coupled with a local search procedure, was also used to solve the problem of bank-branch location and sizing with fixed-charge costs in (Monteiro and Fontes, 2006).

A recent work on MCNFPs is the one of (Nahapetyan and Pardalos, 2007) where the authors consider a concave piecewise linear cost function. The problem is transformed into a continuous one with a bilinear cost function, through the use of a nonlinear relaxation technique. First, the problem is formulated as a mixed integer program, by introducing the usual binary variables y_{ij}^k associated to the fixed costs, where k identifies the linear segment of the cost function. Then, the binary nature of y_{ij}^k and constraint $x_{ij}^k \leq M y_{ij}^k$ are replaced with $y_{ij}^k \geq 0$ and $x_{ij}^k = x_{ij} y_{ij}^k$, respectively, where x_{ij} is the flow in arc (i, j). The relaxed problem is then solved with a dynamic slope scaling method, based on the one proposed in (Kim and Pardalos, 1999, 2000) and explained above.

The same authors approximate the fixed-charge cost function of NFPs by a concave piecewise linear function in (Nahapetyan and Pardalos, 2008). The problem is transformed into a continuous one with a bilinear cost function. This approach is considered a novelty because fixed-charge functions are usually approximated to linear functions. One of the cost functions represents a line connecting the origin and some point $(\varepsilon_{ij}, f(\varepsilon_{ij}))$, and is defined as $f_{ij}^{\varepsilon_{ij}}(x_{ij}) = c_{ij}^{\varepsilon_{ij}} x_{ij}$ if $x_{ij} \in [0, \varepsilon_{ij}[$. The other one is defined as $f_{ij}^{\varepsilon_{ij}}(x_{ij}) = c_{ij} x_{ij} + s_{ij}$ for $x_{ij} \in [\varepsilon_{ij}, \lambda_{ij}]$, where λ_{ij} is the capacity of arc (i, j) , and $c_{ij}^{\varepsilon_{ij}} = c_{ij} + s_{ij}/\varepsilon_{ij}$ with c_{ij} as the flow cost and s_{ij} the fixed cost. Although the arcs are

capacitated, this problem can be transformed into an uncapacitated one by substituting the capacities with a sufficiently large M (constant). The problem thus formulated, a value for $\varepsilon_{ij} \in [0, \lambda_{ij}]$ is chosen and the problem is then solved by the DSS algorithm developed in (Nahapetyan and Pardalos, 2007). At the end of each iteration, every flow variable x_{ij} is tested in order to verify if its value is within $]0, \varepsilon_{ij}[$. If so, ε_{ij} is decreased by a constant $\alpha \in]0, 1[$, such that $\varepsilon_{ij} \leftarrow \alpha\varepsilon_{ij}$, otherwise the algorithm stops and the best found solution is returned. They were able to improve upon the results of the original DSS (Kim and Pardalos, 1999).

Upper Bounds (UBs) based on local search are obtained in (Fontes et al, 2003) to solve SSU concave MCNFPs. The local search is based on swaps of arcs and is performed repeatedly with different initial solutions, this way avoiding getting trapped into a local optimum. Given an initial feasible solution, and for every subtree T_y in the solution, the local search procedure tries to put T_y "under" another node k that does not belong to that subtree. If a new solution, thus constructed, has a better cost, the UB is updated and the procedure continues to the next subtree. When no more reductions in the cost can be found the algorithm stops. The initial feasible solution is provided by a Lower Bound (LB), found by a relaxation of the state space of a DP recursion, and it consists of a network supplying a set of demand nodes. This set of supplied nodes can be, and usually is, only a subset of all demand nodes. Supplied nodes are added to the set of fixed-nodes and the rest are added to a temporary nodes list. Then, the temporary nodes are appended to the solution tree. Each temporary node k is selected, one at a time, and the arc linking it to the LB tree is identified as the one representing the lowest cost for the path linking the source node and node k. This action is performed until the set of temporary nodes is empty, and a new improved solution is found. The algorithm is tested over a set of randomly generated problem instances, and over a set of problems satisfying the triangle inequality, and considers two cost functions, a fixed-charge and a second-order concave polynomial. The algorithm provides very good solutions, although not all of them optimal.

Lower bounds for SSU concave MCNFPs, derived from state space relaxations, are given in (Fontes et al, 2006c). The state space relaxation associated with a DP recursion can be translated into a reduction on the number of states, by forcing constraints in the linear programming formulation to appear as variables of the DP. The authors provide a new relaxation adding a new constraint to the q-set relaxation forcing the solution, of a problem with $n+1$ arcs, to have exactly n arcs. The bound obtained is further improved by penalizing demand nodes not fully supplied. These LBs are later on used in the bounding phase of a Branch-and-Bound procedure given in (Fontes et al, 2006a).

Kim et al (2006) introduced tabu search strategies into the basic DSS having improved upon the results of the basic DSS developed in (Kim and Pardalos, 1999). The new algorithm is called Enhanced DSS and has three phases. The first phase runs the basic DSS and whenever the best solution to the moment is updated, the arcs with the largest changes on the flow, when comparing to the previous iteration, are added to a set called the inspiration set ξ. Also, a record of the frequency of appearance of each arc with a positive flow is incremented. After reaching one of the DSS stopping criteria, the intensification phase follows. In it, some arcs are chosen to be tabu, accordingly to the frequency of their appearance in the previous solutions, and others, including the ones in ξ, are added to a candidate arcs list α which will be the ones allowed to enter new solutions. Once these sets are identified, the DSS phase is run again. The initial linearisation factors for those arcs not in α, to be used in the DSS phase that follows the intensification phase, are the same linear factors associated with the arcs of the most recently improved best solution. At the end of the intensification phase, the third phase, the diversification phase, takes place in order to explore new regions on the search space. Arcs appearing not so frequently are added to the candidate list β, based on information about the reduced costs, the amount by which $\bar{c}_{ij}$ has to be reduced in order for arc (i, j) to enter the solution. Tabu and non-tabu lists are also maintained during this phase. The DSS phase follows once more using the reduced costs as an initial linearisation factor. Both the intensification and the diversification are run a fixed number of times. The tabu mechanisms introduced in this DSS were inspired by the tabu search heuristic previously developed in (Sun et al, 1998) to solve fixed-charge transportation problems which, to the moment, and along with the one of (Glover et al, 2005), and more recently of (Aguado, 2009), is still one of the most efficient heuristic methods to solve such an NP-hard problem.

A hybrid between simulated annealing and tabu search with an adaptive cooling strategy is the algorithm used by Altiparmak and Karaoglan (2006) to solve the concave cost transportation problem, where the cost function is proportional to the square root of the flow $c_{ij}\sqrt{x_{ij}}$. After the generation of an initial feasible solution, swap moves between an arc in the solution and an arc not in the solution are applied in order to improve the solution. An arc is added to the solution, thus creating a cycle. As in a pivot move on the network flow simplex algorithm, in order to maintain the feasibility of the solution the flow of the arcs in the cycle is increased or decreased, as needed, accordingly to the flow on the arc to be dropped from the solution. The set D of arcs from the cycle whose flow must be adjusted by being decreased is identified and the arc $(k, l) \in D$ with the least amount of flow is the one to be dropped from the solution. The tabu procedure is incorporated in the algorithm in the form of two tabu lists, one keeping track of the arcs leaving the solution and another one keeping track of the arcs entering the solution. This

way, the number of arcs to be tested decreases, and consequently the computational time also decreases. The adaptive cooling schedule is based on a ratio between the temperature at the previous iteration and 1 minus the cubic root of the temperature, allowing for a slower temperature decreasing rate. The heuristic thus defined was able to improve upon the computational results obtained with its basic forms tabu search, simulated annealing, and adapted simulated annealing heuristics and also upon results reported in the literature.

Genetic algorithms are based on the evolution of species and the main idea is to take a set of solutions, which are called a population or generation, and to combine the best of them, following the maxima *"the survival of the fittest"*, in order to generate new improved solutions. A mutation factor is also usually incorporated.

Smith and Walters (2000) provide a heuristic method based on Genetic Algorithms to find minimum cost optimal trees on networks and apply it to the solution of SSU concave MCNFPs. The cost functions considered are concave given by the square root of the flow. Randomly generated feasible trees are considered for the initial population. The authors stress out the problematic of generating feasible trees specially in the mutation and the crossing of parents and propose a technique for each. Accordingly to their fitness value, two parents at a time, T_1 and T_2, are chosen to reproduce thus creating two new trees. In order to accomplish that, a bipartite graph is created by overlapping T_1 and T_2. The children have a common structure constituted by the parents common arcs. The number of arcs unique to each parent is the same. Therefore, these arcs are chosen in pairs, one from each parent, and one of them is attributed to one child and the other to its sibling, with a probability of 0.5. If at least one child is not a tree the crossing process is repeated until both of them are. The mutation operator is applied to a subset of the population, and is defined so that one arc is randomly chosen to be substituted by another one in such a way as to maintain the solutions feasibility, that is, so that the solution is still a tree. The authors solve two types of problems, one is the maximization of total length of the tree and the other the minimization of costs of a water distribution network. The cost functions used in the water distribution problem are related with the diameter of the water pipe which is proportional to the square root of its capacity. The sizes of the problems range from 15 to 35 nodes. Although no results are reported regarding the computational time spent by the algorithm in order to solve the problems, the average number of iterations/generations needed to find the optimal solution for the maximization problem and the best solution for the minimization problem are provided.

Fontes and Gonçalves (2007) use a genetic algorithm coupled with a local search procedure, which was called HGA, to solve the SSU MCNFP with general concave costs. Random keys are used to encode the chromosome, as they allow all solutions generated

by crossover to be feasible solutions. In order to create a new generation of solutions, the algorithm selects the top 15% of chromosomes, regarding their fitness value, which are directly copied onto the next generation. The mutation operator used is not a traditional one since the bottom 15% of chromosomes of the next generation are randomly generated chromosomes. Finally, the remaining chromosomes to integrate the next generation are created by applying a 70% probability crossover operator. The crossover between two parent solutions is performed by considering a gene at a time. The algorithm generates a vector with as many random numbers (in the interval $[0, 1]$) as genes in a chromosome. Every random number on that vector is tested and if its value is lower than a certain probability, say 70%, then the gene of the offspring is drawn from the best parent, otherwise it is drawn from the other parent. The local search procedure operates through swap operations between arcs already in the solution and arcs not in the solution. Arcs (i, j) belonging to the solution tree are sorted and considered in descending order of nodes priority. Then each arc (k, j) outside the solution tree, is considered in descending order of priority, and the first one that does not introduce a cycle in the solution is the one chosen to substitute the leaving arc (i, j). When compared with results in the literature, the HGA was able to improve upon upper bounds provided by a heuristic algorithm based on local search, as well as running times.

In a transportation network design problem the objective is to choose among a given set of projects in order to optimize objectives such as, for example, the minimization of total travel time, subject to resource constraints. Poorzahedy and Rouhani (2007) solve transportation network design problems by proposing seven hybrid algorithms based on a previously developed ant system (Poorzahedy and Abulghasemi, 2005) and on three improvements with notions borrowed from genetic algorithms, simulated annealing and tabu search. The first improvement introduced modifies the way pheromones are updated, allowing only the three best solutions to contribute to the pheromone update. The second improvement is based on evolutionary algorithms and it allows mutation to take place under some conditions. The mutation is applied in substitution of the construction phase, and it occurs in the middle of the run of the algorithm, that is to say, in the fifth iteration since the algorithms are allowed to run only 10 iterations. The 3 best solutions of each of the previous four iterations, are retained. These solutions will be used to calculate the frequency of appearance of each project. Then, the 2 best solutions of the previous iterations are also retained along with the 2 best solutions of them all, with repeated solutions allowed. Repeated solutions identify the least, or next least, frequent project and substitute it with the most, or next most, frequent project provided that the solution is still feasible. The solutions thus found are considered new solutions and the algorithm continues to the next step, the pheromone update. The last improvement, applied from the second itera-

tion onwards, is based on simulated annealing concepts and its purpose is to reduce the computational effort of computing net benefits by decreasing the probability of solutions with low levels of energy, as opposite to the usual simulated annealing. The seven hybrid algorithms are constructed by incorporating into the ant system different combinations of these three improvements, as well as incorporating each one on its own. The algorithms were applied to a real-size traffic-network of a city in Iran and the algorithm incorporating all three improvements achieved the best results of them all.

More recently, Dang et al (2011) developed a deterministic annealing algorithm for the capacitated version of the concave MCNFP, that can be used to solve both single source and multiple source cases. The use of a Hopfield type barrier function is a notion borrowed from the theory of neural networks, and is used to cope with the lower and upper bounds on the capacities of the arcs. Each arc (i, j) is associated to a Hopfield-type barrier field thus allowing the capacity constraints to be incorporated into the objective function. The barrier parameter has a behaviour similar to the temperature on the simulated annealing, decreasing towards zero, from a large positive number. The linear constraints, the flow conservation constraints, are dealt with by using lagrangean multipliers, to incorporate them into the objective function. This way, a Lagrange and barrier function is obtained. Numerical results are provided, for problems with from 5 up to 12 nodes, for both a linear cost function $b_{ij} \cdot x_{ij}$ and a concave second order polynomial function $-a_{ij} \cdot x_{ij}^2 + b_{ij} \cdot x_{ij}$. The authors were able to improve upon results reported in the literature and upon results obtained with a specific minimization function of MATLAB.

4.4.3. Test Problems

In order to test the algorithm that was developed to solve SSU concave MCNFPs, regarding its effectiveness and efficiency, a set of problems taken from the literature is used. These problems are available and can be downloaded from (Beasley, 2012). This way a comparison between the results obtained with the ACO algorithm and the ones obtained with other methods can be made. Next we will describe the general characteristics of this set of problems, but an extensive description can be found in (Fontes et al, 2003).

The problems are divided into ten groups $\{g_1, g_2, \ldots, g_{10}\}$, with different ratios between variable and fixed costs, V/F, since it has been proven in (Hochbaum and Segev, 1989) that the values of such ratios are the main parameter in defining problem difficulty for fixed-charge MCNFPs. It is important to notice that every arc in these problems has associated a concave cost. For each of these groups we can find three problem instances identified as A, B, and C. Furthermore, the problems are also classified regarding the

number of nodes considered which varies within $\{10, 12, 15, 17, 19, 25, 30, 40, 50\}$. For problems with 10, 12, 15, 17, 19, 25, and 30 nodes, all groups are considered, while for the remaining only the first five groups are available. For further details on these problems please refer to (Fontes et al, 2003). Therefore, there is a total of 240 problems to be solved.

Three types of polynomial cost functions, which is motivated by the easiness of approximation of any cost function by a Taylor Series, are considered:

Type I: A first-order polynomial cost function, representing a linear routing cost and a fixed-charge cost,

$$g_{ij}(x) = \begin{cases} b_{ij} \cdot x_{ij} + c_{ij}, & \text{if } x_{ij} > 0, \\ 0, & \text{otherwise,} \end{cases} \tag{4.15}$$

Type II: A second-order polynomial cost function representing a routing concave cost but without the fixed charge component,

$$g_{ij}(x) = \begin{cases} -a_{ij} \cdot x_{ij}^2 + b_{ij} \cdot x_{ij}, & \text{if } x_{ij} > 0, \\ 0, & \text{otherwise,} \end{cases} \tag{4.16}$$

Type III: A complete second-order polynomial cost function, incorporating both routing concave costs and fixed-charge,

$$g_{ij}(x) = \begin{cases} -a_{ij} \cdot x_{ij}^2 + b_{ij} \cdot x_{ij} + c_{ij}, & \text{if } x_{ij} > 0, \\ 0, & \text{otherwise,} \end{cases} \tag{4.17}$$

where a_{ij}, b_{ij}, and $c_{ij} \in \mathbb{Z}^+$. Since three types of cost functions are considered, the number of problem instances to be solved increases to 720. The concavity of the cost functions of Type II and III, given by a_{ij}, is defined such that it takes the maximum value guaranteeing that the cost functions are nondecreasing and that an equilibrium between all arcs costs is reached, thus increasing the number of local optima solutions and making the problem harder to solve. Furthermore, cost function of Type III in addition to the nonlinear concave cost also incorporates a fixed-charge component.

In addition to the downloaded test set of problems, we have also generated larger size problem instances with $\{60, 80, 100, 120\}$ nodes where we consider five different

groups, with three problem instances each. Therefore, in total, we have generated 60 new problem instances. Please note that problems with cost functions of Type II and III cannot be solved by CPLEX. All the problems used herein, the former and the newly generated ones, have the source node always located at an extreme point of the grid, as problems with this characteristic are known to be harder to solve (see e.g. (Fontes et al, 2003; Dahl et al, 2006)).

4.5. An Ant Colony Optimization Algorithm Outline for the SSU Concave MCNFP

In this section, we develop a heuristic method, based on ant colony optimization, to solve SSU concave MCNFPs. As we have already mentioned before, ACO algorithms are characterized by a set of decisions that have to be accounted for, such as the construction of the solution or the value of the parameters used. We present and discuss each of these decisions, starting with the two major decisions to be made which are the representation and the construction of the solution.

4.5.1. Representation of the Solution

As ACO is a probabilistic constructive method, that builds a solution by adding to it one component at a time, it can be applied to any sort of combinatorial problem. The big question is how to represent the problem so that ants can be used to solve it (Dorigo and Stützle, 2004). Therefore, the first and most important decision to be made is the representation of a solution to the problem being solved, because a poor representation can lead to not so good solutions. As we have already mentioned, a feasible extreme solution for the SSU MCNFP with concave costs is a set of existing arcs forming a tree, in other words a connected graph without cycles. In our ACO algorithm, each ant solution consists of a number of directed arcs that equals the number of demand nodes, provided that all nodes are connected and that the solution has no cycles. Bellow, in Fig.4.1, is an example of a solution for a problem with five demand nodes.

In this case, the overall amount to flow in the network is 53. Demand nodes 1 and 2 are directly satisfied by the source node t, with a flow of 10 and 8, respectively. Nodes 4 and 5 receive their demand via node node 3. In this case, node 3 gets an inflow of 35, keeps its demand, which is 22, and then sends the rest to the other nodes, that is, a flow of 5 and 8 to demand nodes 4 and 5, respectively.

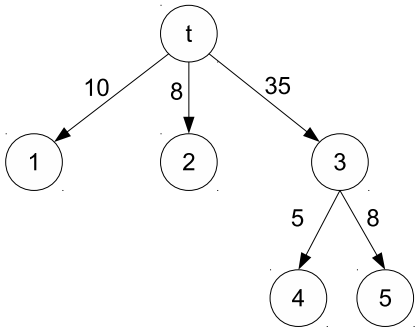

Fig. 4.1: Example of a solution tree for the SSU MCNFP.

4.5.2. Construction of the Solution

The method that is used to construct solutions for the SSU MCNFP guarantees that a solution is always feasible. All ants begin their solution construction at the source node. Initially, an ant selects an existing arc linking the source t and one of the demand nodes $i \in N \setminus \{t\}$. Then, the ant selects another arc, from the set of available arcs linking the source or one of the demand nodes already in the partial solution to another demand node not yet considered. This last step is performed until all the demand nodes are in the solution. Therefore the admissibility of the solution is guaranteed. The choice of the arc entering the solution is made using the probability function defined bellow:

$$P_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{(i,j)\in A} [\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}, \tag{4.18}$$

where τ_{ij} is the pheromone present in arc (i, j) at the current iteration; η_{ij} is the heuristic information for the problem; $\alpha > 0$ and $\beta > 0$ are both parameters weighting the relative importance of the pheromone value and the heuristic information, respectively. The heuristic information, which in this case is given by

$$\eta_{ij} = \frac{1}{c_{ij} + b_{ij}}, \tag{4.19}$$

represents local knowledge of the problem. In the TSP, that was originally the problem used in (Dorigo et al, 1991), its value is given by $\eta_{ij} = \frac{1}{c_{ij}}$, where c_{ij} represents the cost (length) of the arc, thus increasing the importance of shorter arcs. In the case of the problems we are solving, the cost functions are represented by polynomials of degree one or two, see Section 4.4.3, where two or three coefficients are used. The cost is undoubtedly the local information to be made available to the ants, since the goal is the minimization of costs. We have attempted several values for it as the objective functions varied their form and the number of cost coefficients. The coefficient a_{ij} did not contribute positively

to the solutions found, that is, when incorporated in the denominator of η_{ij} the solutions were worst. Then, we were left with b_{ij}, the variable cost, and with c_{ij}, the fixed-charge. We have already discussed the implications of the ratio between variable and fixed costs, therefore it is not with surprise that we have observed that none of them achieve better results on its own, where applicable.

We now present an example of the construction of a solution. Let us consider a network with five nodes, one source node and four demand nodes. The demand for each demand node is given by $\{20, 65, 20, 15\}$. The following matrix presents the existing arcs:

		to				
		t	1	2	3	4
from	t	-	0	1	0	1
	1	-	-	0	1	0
	2	-	1	-	1	1
	3	-	0	1	-	0
	4	-	0	1	1	-

Each ant starts at the source node t. Therefore, the source is the first node to be included in the solution, Fig. 4.2.

Fig. 4.2: Each ant starts at the source node t.

Then, based on the pheromone concentration values, an arc linking the source node t and one of the demand nodes, is chosen. In this case, only the arcs $(t, 2)$ and $(t, 4)$ can be chosen. Let us suppose arc $(t, 2)$ is the one entering the solution. At this moment, the solution tree has only one arc and two nodes, source node t and demand node 2, as can be seen in Fig. 4.3.

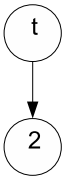

Fig. 4.3: Arc $(t, 2)$ enters the solution tree.

Next, the ant will choose an arc linking either node t or node 2 to another demand node not yet in the solution. Please note that, we only consider arcs with outflow from nodes t

and 2. This way, we know that no cycle can be introduced in the solution. Observing the table given above, we can see that the arcs that can be chosen are $(t, 4)$, $(2, 1)$, $(2, 3)$ and $(2, 4)$. Now, let us assume that the chosen arc was arc $(2, 1)$. The subtree configuration is given in Fig. 4.4.

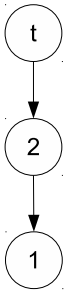

Fig. 4.4: Arc $(2, 1)$ enters the solution tree.

The arcs that can be chosen by the ant to enter the solution are now arcs $(t, 4)$, $(1, 3)$, $(2, 3)$ and $(2, 4)$. If we follow these steps, until no more demand nodes are out of the solution tree, we have constructed a feasible solution and we arrive to a solution such as the one in Fig. 4.5.

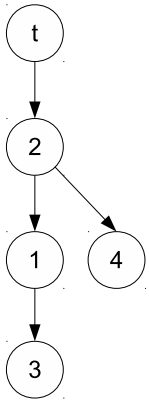

Fig. 4.5: A possible feasible solution tree.

After the construction of the tree, the flow in each arc has to be computed. Only then is the solution complete, as can be seen in Fig. 4.6.

Our method for constructing a solution is very straightforward, which allow us to save some running time. With this method the ant does not walk through a continuous path as in the TSP. Nonetheless, many are the problems where that can not be achieved either because it may turn out to have a poor performance or due to the particular structure of the problem.

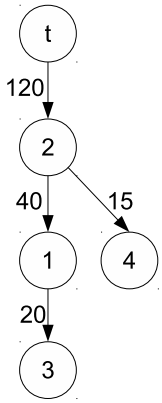

Fig. 4.6: The final solution.

4.5.3. Pheromone Update

After the construction of all the solutions for iteration T, the total cost of the network is computed and the best ant is identified, as the one with the lowest cost of them all. Then, the algorithm updates the pheromones. The pheromone update is performed according to the following update function:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \Delta\tau_{ij}. \tag{4.20}$$

Initially, and simulating the natural process of evaporation, the pheromone values are reduced in every existing arc $(i, j) \in A$. This is represented by the first component of Eq. (4.20), that is $(1 - \rho) \cdot \tau_{ij}$, where ρ represents the pheromone evaporation rate and $\rho \in]0, 1]$, and τ_{ij} is the pheromone quantity in arc (i, j). The value of the evaporation rate indicates the relative importance of the pheromone values from one iteration to the following one. If ρ takes a value near 1, then the pheromone trail will not have a lasting effect, while a small value will increase the importance of the arcs a lot longer. The second component, $\Delta\tau_{ij}$, represents the pheromone quantity to be deposited in arc (i, j) , and is given by:

$$\Delta\tau_{ij} = \begin{cases} \dfrac{Q}{g(S)} & \text{if } (i, j) \text{ belongs to solution S,} \\ 0 & \text{otherwise,} \end{cases} \tag{4.21}$$

where S is the best solution found at the current iteration, $g(S)$ is its corresponding cost, and Q is a positive proportionality parameter.

4.5.4. Pheromone Bounds

Initially, the algorithm starts by depositing an equal amount of pheromone in all arcs $(i, j) \in A$, so that every arc has the same chance of being chosen

$$\tau_{ij} = \tau_0, \forall (i, j) \in A. \tag{4.22}$$

At each iteration, after the pheromone update is performed a check is done to find out if its value is bounded in the interval $[\tau_{min}, \tau_{max}]$, following the work of (Stützle and Hoos, 1997). The initial pheromone bounds are only set at the end of the first iteration, after the best solution is identified.

The τ_{max} value depends on the cost of the best solution found so far G^* and on the pheromone evaporation rate ρ, see Eq. (4.23).

$$\tau_{max} = \frac{1}{\rho \cdot G^*}. \tag{4.23}$$

The τ_{min} value depends on the upper bound for the pheromone value τ_{max} and on a parameter p_{best}, as given in Eq. (4.24),

$$\tau_{min} = \frac{\tau_{max} \cdot (1 - \sqrt[n]{p_{best}})}{(\frac{n}{2} - 1) \cdot \sqrt[n]{p_{best}}}, \tag{4.24}$$

where p_{best} is the probability of constructing the best solution. Since both τ_{min} and τ_{max} depend on the cost of the best solution found so far, they have to be updated each time the best solution is improved. After each pheromone update a check is made to ensure that pheromone values are within the limits. If, on the one hand, some pheromone value is bellow τ_{min}, it is set to τ_{min}. On the other hand, if some pheromone value is above τ_{max}, it is set to τ_{max}.

4.5.5. Algorithm

In Algorithm 5, we provide an outline of the hybrid ACO algorithm that we have developed to solve the SSU MCNFP with concave costs, HACO1.

The algorithm starts by initializing the necessary parameters, such as pheromone trails and the best global solution. Then, each ant will construct its solution, by using the method described earlier in Section 4.5.2. After all ants have constructed their solutions, the algorithm identifies the best solution found by the ants at the current iteration, S^i. This

action is usually identified as a daemon action since it uses the global knowledge of what has happened in the iteration. The local search procedure, described in Section 4.5.6, is then applied to a subset W of the solutions found at the current iteration. At the end of the local search a new best solution is returned if one is found; otherwise the iteration best solution, previously found, is returned. Then, this solution S^i is compared with the best global solution S^g. If S^i has a lower cost, then the best global solution is updated, $S^g \leftarrow S^i$; otherwise S^g remains the same. The next step is to update pheromone trails, first by evaporating the trails, using the evaporation rate ρ, and then by adding to each component of solution S^i a pheromone quantity inversely proportional to the cost of the solution.

Before the algorithm starts the next iteration, all pheromone trails are checked for violations to either the upper or the lower pheromone bounds. If any arc has a pheromone value in this situation, it is corrected accordingly. The algorithm runs until the maximum number of iterations allowed $maxIter$ is reached.

4.5.6. Local Search

Although the first experimental results in the test set achieved good solutions, as it can be seen in Section 4.6.1, there was still some room for improvement. Therefore, we have developed a local search procedure in order to further improve the results.

After all ants have constructed their solutions S_a, where $a = \{1, 2, \ldots, n\}$, and after the best solution of the iteration S^i is found (see Algorithm 5), the local search procedure takes place. Five ants are selected from the current iteration to perform local search on their respective solutions. One is always the ant that has found the best solution for the current iteration, while the other four are randomly selected from the remaining $n - 1$ ants. The local search allows the algorithm to search, in the neighbourhood of a particular solution, for another solution that might have a lower cost. Given a solution S for the SSU MCNFP, the 1-opt neighbourhood of S, denominated $\mathcal{N}(\mathcal{S})$, consists of all solutions S' that can be obtained by swapping an arc $(i, j) \in S$ with another arc $(k, j) \notin S$, i.e. $\mathcal{N}(\mathcal{S}) = \{S' : S' = S \setminus \{(i, j)\} \cup \{(k, j)\}\}$, provided that certain conditions are observed, as explained next. Whenever an arc is removed from S we have two disjoint graphs, T^1 and T^2. The choice of the arc $(k, j) \notin S$ is made from the set of arcs satisfying $k \in T^1$ if $j \in T^2$ and $k \in T^2$ otherwise. This way only arcs that do not introduce a cycle into the solution are considered. The first candidate to improve the cost is the one chosen for the swap. Furthermore, the arcs leaving and the arcs entering S are considered in a specific order. Candidate arcs are removed, one at a time from S, in ascending order of

Algorithm 5 Pseudo-code for the proposed Ant Colony Optimization algorithm (HACO1).

Initialize $\tau_{ij} \leftarrow \tau_0, \forall (i,j) \in A$
Initialize τ_{min}, τ_{max}
Initialize $S^g \leftarrow \emptyset$, $S^i \leftarrow \emptyset$, $G(S^g) \leftarrow \infty$, and $G(S^i) \leftarrow \infty$
Set ρ, Q, α, β, and p_{best} values
Create ants
while it $\leq$ maxIter **do**
  for all ants a **do**
    Let a construct solution S_a
  end for
  Identify $S^i \leftarrow \{S : min\{G(S_1), \ldots, G(S_n)\}\}$
  Construct $W = \{S^i \cup 4$ randomly chosen $S_a\}$
  Apply local search to all solutions $S \in W$ and return W'
  Identify $S^{LS} \leftarrow \{S \in W' : min\{G(S)\}\}$
  $S^i \leftarrow \{S : min\{G(S^i), G(S^{LS})\}\}$
  Update $G(S^i)$ accordingly
  if $G(S^i) < G(S^g)$ **then**
    $S^g \leftarrow S^i$
    $G(S^g) \leftarrow G(S^i)$
    Update τ_{max} and τ_{min}
  end if
  Evaporate pheromone values $\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \forall (i,j) \in A$
  Reinforce pheromone values $\tau_{ij} \leftarrow \tau_{ij} + \frac{Q}{G(S^i)}, \forall (i,j) \in S^i$
  for all arcs $(i,j) \in A$ **do**
    if $\tau_{ij} < \tau_{min}$ **then**
      $\tau_{ij} \leftarrow \tau_{min}$
    else if $\tau_{ij} > \tau_{max}$ **then**
      $\tau_{ij} \leftarrow \tau_{max}$
    end if
  end for
  $it \leftarrow it + 1$
end while
return S^g and $G(S^g)$

pheromone. In opposition, candidate arcs are added to the solution in descending order of pheromone. We are then trying to replace arcs with lower pheromone values with arcs with higher pheromone values. The first cost improving solution is accepted and the search continues with the following arc to be removed from S. Therefore, the local search we have defined is a greedy one.

The pseudo-code for the local search procedure is given in Algorithm 6.

Algorithm 6 Pseudo-code for the proposed Local Search procedure that was incorporated into the ACO algorithm developed.

1: $W = \{S^i \cup 4$ randomly chosen $S_a\}$
2: **for all** $S \in W$ **do**
3: Sort all $(i, j) \in S$ in ascending order of τ_{ij}
4: **for all** arcs $(i, j) \in S$ **do**
5: Identify P as the set of all arcs $(k, j) \notin S$ that can replace $(i, j) \in S$ without forming a cycle
6: Sort $(k, j) \in P$ in descending order of τ_{ij}
7: **for** each arc $(k, j) \in P$ **do**
8: $S' = S \backslash \{(i, j)\} \cup \{(k, j)\}$
9: **if** $G(S') < G(S)$ **then**
10: $S \leftarrow S'$
11: GOTO next $(i, j) \in S$ //Line 4
12: **end if**
13: **end for**
14: **end for**
15: **end for**
16: **return** All five improved solutions and their respective costs

Given a solution S to be improved, we start by sorting the arcs in S in ascending order of their pheromone value. For each of these arcs we try to find an alternative one that improves the cost of the current solution. In order to do so, we find all the arcs entering node j that can replace the current one while maintaining solution feasibility, i.e. without forming a cycle. We attempt to replace the original arc, starting with the ones with a higher pheromone concentration. If one of the replacements improves the cost of the solution S, it is accepted and we proceed to the next arc in the solution S without attempting the remaining options. At the end of the local search procedure, if the solution found S' improves the cost of the original solution S, then the new solution S' is the one used in the remaining of the algorithm.

4.5.7. Example

In order to better clarify this procedure let us give an example of a typical iteration i of the algorithm, while considering a fully connected graph with four demand nodes $\{1, 2, 3, 4\}$ and a root node t.

Let us assume that the HACO1 parameters[1] are given by: $\rho = 0.1$, $p_{best} = 0.5$, $G(S^g) = 150$, $\tau_{max} = 0.067$, and $\tau_{min} = 0.013$. The heuristic information matrix η_{ij} and the probability matrix P_{ij} are calculated according to $\eta_{ij} = \frac{1}{c_{ij}+b_{ij}}$ and to Eq. (4.18), respectively. Fig. 4.7 provides three pheromone matrices associated with this particular iteration i of the algorithm. Matrix (A) provides the pheromone values to be used at the construction of the solution of every ant (according to Sec. 4.5.1). Matrix (B) is the pheromone matrix obtained after the pheromone values update (as given in Section 4.5.3), and matrix (C) provides the pheromone values after all pheromone bounds violations of matrix (B) are corrected (see Section 4.5.4). The updating of pheromone matrices (B) and (C) is carried out towards the end of the iteration and will be explained later.

(A)

	1	2	3	4
t	0.013	0.021	0.019	**0.041**
1	-	0.013	0.035	0.022
2	0.042	-	0.052	0.013
3	0.037	0.021	-	0.049
4	**0.067**	**0.035**	**0.058**	-

(B)

	1	2	3	4
t	**0.012**	0.019	**0.017**	0.047
1	-	**0.012**	0.032	0.020
2	0.038	-	0.047	**0.012**
3	0.033	0.019	-	0.044
4	0.070	0.042	0.062	-

(C)

	1	2	3	4
t	0,019	0,019	0,019	0,047
1	-	0,019	0,032	0,020
2	0,038	-	0,047	0,019
3	0,033	0,019	-	0,044
4	0,070	0,042	0,062	-

Fig. 4.7: Pheromone matrices for an example with the current solution given by $S^i = \{(4, 2), (t, 4), (4, 1), (4, 3)\}$: (A) Initial pheromone matrix (arcs in solution S^i are shown in bold), (B) Updated pheromone matrix, and (C) Pheromone matrix with all values within the allowed pheromone bounds interval.

At the beginning of iteration i all ants are created and every ant constructs a solution. Then, a set W consisting of the best solution of iteration i, S^i, and four randomly chosen solutions is created and the local search procedure in Algorithm 6 is applied to each

[1] Some parameters are not provided in this example because their values are not required for the operations here exemplified.

solution in W[2]. We will use Fig. 4.8 to illustrate the evolution of a solution $S \in W$ with the application of local search.

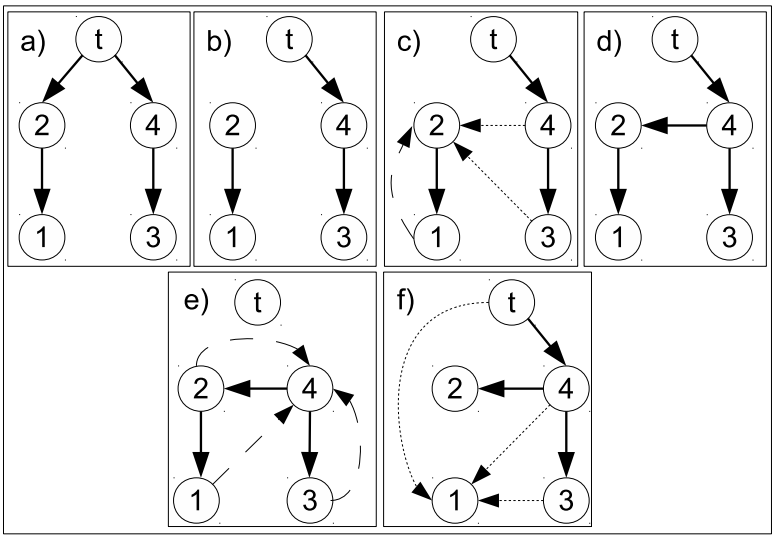

Fig. 4.8: Example of the Local Search procedure.

Let us suppose that we are inspecting the neighbourhood of one particular solution $S \in W$ given by $S = \{(t,2),(t,4),(2,1),(4,3)\}$, see Fig. 4.8 (a). Note that, solution S is already sorted in ascending order of arc pheromone, i.e. $\tau_{t2} \leq \tau_{t4} \leq \tau_{21} \leq \tau_{43}$ (see matrix (A) in Fig. 4.7).

We try to improve S by replacing each of the four arcs in S, one at a time, with arcs that decrease the total cost of the solution tree.

First, we remove arc $(t,2)$, as given in Fig. 4.8 (b).

Then, we identify the set of candidate arcs $(k,2) \notin S$ to substitute arc $(t,2)$, which is given by $P = \{(4,2),(3,2)\}$. Note that P is already sorted in descending order of arc pheromone, i.e. $\tau_{42} \geq \tau_{32}$. Fig. 4.8 (c) shows, in fine dashed lines, all arcs that could be used to reconnect node 2 to the solution tree S. Nonetheless, observe that arc $(1,2)$, in a dashed line, cannot be in the set of candidate arcs P because it would introduce a cycle into the solution, as well as a disconnected graph.

Following the algorithm, we replace arc $(t,2)$ with arc $(4,2)$ thus obtaining $S' = \{(4,2),(t,4),(2,1),(4,3)\}$.

Next, we calculate $G(S')$ and let us assume that $G(S') < G(S)$. Then, we accept this arc swap, Fig. 4.8 (d), and continue with the local search procedure considering this

[2] Please note that, in this small example although we make reference to five solutions in W, there could only have been a maximum of four solutions given the number of nodes in the problem.

new solution S' by making $S \leftarrow S'$. After this swap node 2 gets its demand from node 4 instead of from node t.

The local search will now try to replace arc $(t, 4)$. It is important to notice though that we never go backwards while trying to improve a solution, which means that once a swap has been made, the procedure will not try to improve the swaps that have already been performed. Let us go back to our example.

For arc $(t, 4)$ the new $P = \emptyset$, because none of the arcs $(k, 4) \notin S$ can provide a feasible solution (they would all introduce a cycle), see Fig. 4.8 (e). Therefore, the procedure keeps arc $(t, 4)$ in S, and continues the search with the next arc, arc $(2, 1)$, which will render $P = \{(t, 1), (4, 1), (3, 1)\}$, Fig. 4.8 (f).

The local search will continue until all arcs in the original solution S have been tested, and then steps into the next solution $S \in W$, until all five solutions have been improved (or attempted to).

Now, we return again to Algorithm 5. The algorithm identifies the best of the five solutions returned by the local search procedure, S^{LS}. Let us assume $S^{LS} = \{(4, 2), (t, 4), (4, 1), (4, 3)\}$ and $G(S^{LS}) = 100$.

We compare $G(S^{LS})$ with $G(S^i)$ and let us assume that $G(S^{LS}) < G(S^i)$. Then, we make $S^i \leftarrow S^{LS}$ and $G(S^i) \leftarrow G(S^{LS})$.

Since $G(S^i) < G(S^g)$, $100 < 150$, we update $S^g \leftarrow S^i$ and $G(S^g) \leftarrow G(S^i)$.

The new pheromone bounds are calculated: $\tau_{max} = 0.1$ and $\tau_{min} = 0.019$ (see Section 4.5.4).

Next, pheromone values are updated: Firstly, by evaporating 10% of the pheromone values of all arcs, $\tau_{ij} \leftarrow 0.9 \times \tau_{ij}$; Secondly, by adding $\Delta\tau_{ij} = \frac{Q}{G(S^i)} = \frac{2}{100} = 0.01$ to the pheromone τ_{ij} of each arc $(i, j) \in S^i$ (the arcs signalled in bold in matrix (A)). The resulting pheromone matrix (B) in given in Fig. 4.7.

The last step at iteration i is to check for violations on the upper and lower bounds of pheromone values. In matrix (B) in Fig. 4.7, all pheromone values smaller than τ_{min} or larger than τ_{max} are signalled in bold font. The signalled values only violate the lower bound and thus are replaced by τ_{min}, as can be seen in matrix (C) of Fig. 4.7.

The HACO1 algorithm steps into iteration $i + 1$, provided that the stopping criteria is not satisfied, and starts all over again now using pheromone matrix (C) as the initial pheromone matrix.

4.6. Computational Results

In this section, we report on the computational results obtained with the heuristic described above. In order to evaluate an heuristic algorithm, it is not always clear what must be tested. The work of Rardin and Uzsoy (2001) provides an excellent help in this area, and we have followed it to some extent.

We also present literature results for the same problems in order to compare the performance and effectiveness of our algorithms.

The algorithms proposed, the ACO and the ACO+LS which we have named HACO1, were implemented in Java and the computational experiments were carried out in a PC with an Intel Core 2 processor at 2.4 GHz and 4 GB of RAM.

Heuristics in general can be evaluated regarding three aspects of their performance:

- **Effectiveness** - the quality of the solution concerning the objective function(s) value(s).
- **Efficiency** - usually related to the computational time required to run the algorithm.
- **Robustness** - is related to the ability of the heuristic to reproduce the same solution or a very similar one, in different runs.

In order to measure effectiveness, the solutions obtained with the heuristic method can be compared with optimal values, obtained via some optimization software as CPLEX (2012) or LINGO (2012) or via an exact solution method, with known lower and/or upper bounds, or with benchmark results. The time used to run the algorithm is one of the most controversial measures used. Many authors argue that a more recent computer may always take less time to run the algorithm, defending that the number of evaluated solutions during the run of the algorithm is preferable. But, if we observe the available literature, we can see that this last approach is scarcely used. The analysis of the robustness can always be approximately achieved by computing the minimum, maximum, average and standard-deviation of the solutions. If the first three statistics are around the same values and the standard deviation is small, then the method is robust.

In this work, we compare the results obtained with our algorithm with the ones obtained via a Hybrid Genetic Algorithm (HGA) reported in (Fontes and Gonçalves, 2007), with the ones obtained via a Dynamic Programming algorithm (DP) reported in (Fontes et al, 2006b), and with the ones obtained with CPLEX 9.0. The computational time required to run HACO1 is compared to the one of the HGA and to the one of CPLEX, and

they were all run in the same computer. We make use of the following measures in order to evaluate the performance of our heuristic:

1. Time, in seconds, required to run the algorithm;
2. % Gap, which is calculated by comparing two solutions, and is given by:

$$\text{Gap}(\%) = \frac{HS - OptS}{OptS} \times 100,$$

where OptS stands for the best known solution, an optimum solution in some cases, and the HS stands for the best solution found with the heuristic in question. Regarding the gap, for each problem instance we present the average values obtained for the five runs of the algorithm.

4.6.1. Parameters Setting

In this section, we study the influence of the values for some of the most important parameters. The tests performed on the parameters use an ant colony optimization algorithm that does not consider local search, thus the algorithm is hereby identified as ACO. In order to make some tests, and following on the work of (Fontes and Gonçalves, 2007), we have used a randomly drawn set of 3 problems with different sizes and from different groups, and we have retained the three problem instances. We have taken problems with 10 nodes from group 10, problems with 25 nodes from group 7, and problems with 50 nodes from group 3. In total we have used nine problem instances. Therefore, we use, as the experimental set, the same set as the one that was used in (Fontes and Gonçalves, 2007).

To test the behaviour of the algorithm while varying the parameter values, in order to identify the best ones, we had to come up with some values to start with. The values used were mainly draw from literature because our first objective was to infer on the order of magnitude. Their values were: $\alpha = 1$, $\beta = 3$, $\rho = 0.1$, $Q = 2$, $\tau_0 = 1000000$. The tests were conducted in such a way as to fix a parameter value after identifying the best value for it and then proceeding to the next parameter under evaluation. At the end, all parameter values will be set and the algorithm will be ready to be tested in all the problems that were already described in Section 4.4.3.

4.6.1.1. Setting the Heuristic Information

The heuristic information, which is usually a fixed value from the beginning of the algorithm, is also called the *visibility* of arc (i, j) and originally, in the travelling salesman problem, it was seen as the information an ant has if it can use its eyes (Afshar, 2005). Therefore, the closest the cities were the more attractive they became. In our case, the distance is equivalent to the cost of an arc therefore, cheapest arcs must have a higher visibility value, whereas the others must have a lower one. The visibility function has been defined in several different ways, for example in (Lessing et al, 2004) a study is performed both regarding static and dynamic heuristic information.

In our case, we consider

$$\eta_{ij} = \frac{1}{b_{ij} + c_{ij}}, \tag{4.25}$$

for cost function Type I and III, and

$$\eta_{ij} = \frac{1}{b_{ij}}, \tag{4.26}$$

for cost function Type II. Recall that we use three polynomial cost functions, one linear concave and the other two quadratic functions. The most important coefficient is, in this case, the fixed-charge because it will add an extra cost, non-dependent on the flow, at each arc. The other parameter, the coefficient of the first-order term of the polynomial also influences the increase on the cost function. The coefficient of the second-order term of the cost function is disregarded because its main objective is to define the concavity of the cost function, and because the computational experiments performed have demonstrated that it has no significant influence on the performance of the algorithm.

4.6.1.2. Setting α and β Parameter Values

The values α and β, appearing in the probability function defined in Eq. (4.18), are two tunable parameters weighting the pheromone information and the heuristic information, respectively. It has become common knowledge that if, on the one hand, α has a very small value (close to 0) then the cheapest demand nodes are more likely to be chosen, and the algorithm becomes a sort of greedy algorithm. If, on the other hand, β is the parameter with a very small value then the algorithm will give priority to the pheromone information and faster convergence to poor solutions may arise. Therefore, an equilibrium

must be reached and the value for these two parameters must be carefully chosen or else the algorithm may have a low performance.

As the impact of these two parameters are so connected, we made an intensive study, within reasonable limits, extending the values reported in the literature, and using the combination of values for the two of them. In Figs. 4.9, 4.10, and 4.11 we present the results obtained for the gap, for each cost function, TypeI, II, and III, while varying both α and β values within {0,1,2,3,4,5}.

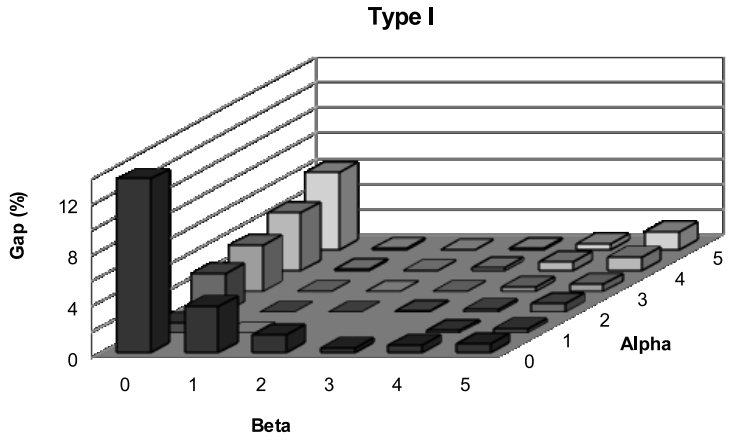

Fig. 4.9: Graphical representation of the average optimality gaps obtained while varying both α and β parameter values within {0,1,2,3,4,5}, and considering Type I cost function.

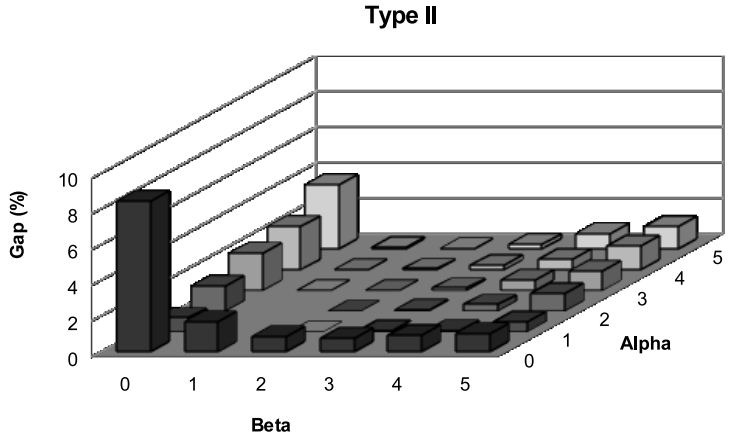

Fig. 4.10: Graphical representation of the average optimality gaps obtained while varying both α and β parameter values within {0,1,2,3,4,5}, and considering Type II cost function.

It becomes very clear, from observing Figs. 4.9, 4.10, and 4.11, that the worst performance of the algorithm occurs when the choice of the arc entering the solution is performed completely at random, that is when $\alpha = \beta = 0$. The bad performance is observed

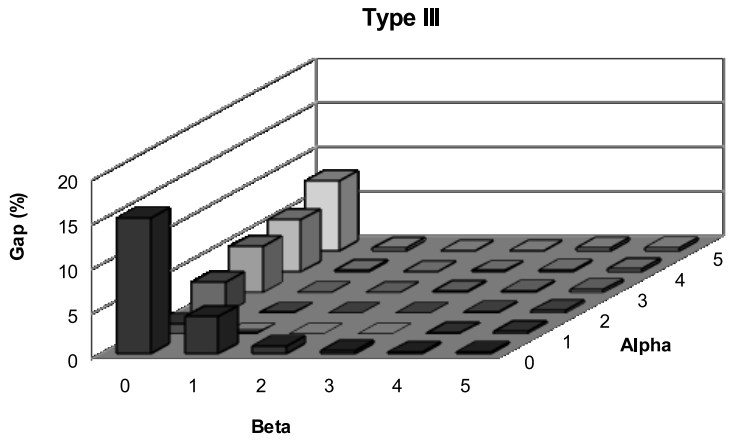

Fig. 4.11: Graphical representation of the average optimality gaps obtained while varying both α and β parameter values within {0,1,2,3,4,5}, and considering Type III cost function.

for each of the three cost functions considered, leading to a maximum average error of 15.31%. This behaviour is somehow expected because only eventually the first *guesses* of the algorithm are good solutions, and a lower gap is achieved. In the case $\alpha = 0$ the probability function ignores the pheromone values and only the heuristic value is considered to choose arcs into the solution. In this case, the algorithm becomes a greedy algorithm only relying on local information. In a problem with such a complexity as this, this is not sufficient to provide good solutions. Although initially the ants may make more use of the heuristic information to perform a fast exploration of the search space nearby solutions with lower cost components, it is not sufficient to produce a good solution afterwards. Furthermore, recall that the main core of an ant algorithm is the communication and cooperation between ants in order to find good solutions. When $\alpha = 0$ such communication is somehow prohibited and ants will tend do erratic walks never remembering where the last walk took them. In the case $\beta = 0$ the heuristic information is completely disregarded and the algorithm only relies on the pheromone values to make its choices. The performance in this case is even worse since the algorithm will depend upon the solutions of the first few iterations. Therefore, if in the first few iterations the algorithm is able to find a relative good solution, then at the end it may have a reasonable performance; Otherwise it will converge to a bad solution, as pheromones tend to increase in bad components. This observation allows us to conclude that the heuristic information is of capital importance for this problem and it must not be ignored.

The gap seems to concentrate at lower values in the middle of the interval considered increasing at the extremes. Therefore, the values associated to the best results are the ones that can be found slightly at the middle of the interval considered. In this case, the values corresponding to the best average gaps are $\alpha = 1$ and $\beta = 2$.

4.6.1.3. Setting the Pheromone Evaporation Rate Value

Evaporation plays an important role in an ACO algorithm. This operation simulates the natural process of evaporation preventing the algorithm from converging too quickly (all ants constructing the same tour) and getting trapped into local optima. The value of the evaporation rate indicates the relative importance of the pheromone values from one iteration to the following one.

In order to infer the best possible value for the pheromone rate regarding the solution of SSU concave MCNFPs, we have performed an intensive series of tests having obtained the results that can be seen in Figs. 4.12, 4.13, and 4.14. If ρ takes a large value, in this case close to 50%, then the pheromone trail will not have a lasting effect, potentiating the exploration of the solution space. Whereas small values increase the importance of the arcs a lot longer, potentiating the exploitation of the search space near good solutions. The results obtained show us that small values for ρ, this is, below 10%, translate into a not so good performance of the algorithm. The reason for this to happen is because the behaviour is like the one of a blind ant, which will only follow the most intensive pheromone trails, afraid of what it may encounter in "unknown" areas. If the evaporation rate becomes larger than 10%, then the algorithm is so focused in performing exploration that it ignores solutions in the neighbourhood that can be better. The best performance, with zero gaps, are obtained for an evaporation rate of 10% and 20%. The algorithm, although presenting very small gaps, cannot find an optimal solution for all problems.

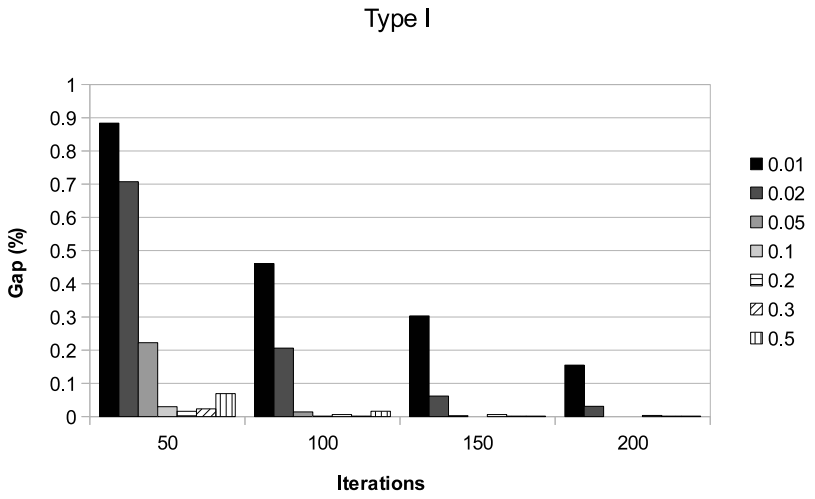

Fig. 4.12: Average optimality gaps obtained while varying the pheromone evaporation rate ρ within {0.01,0.02,0.05,0.1,0.2,0.3,0.5}, and considering Type I cost function.

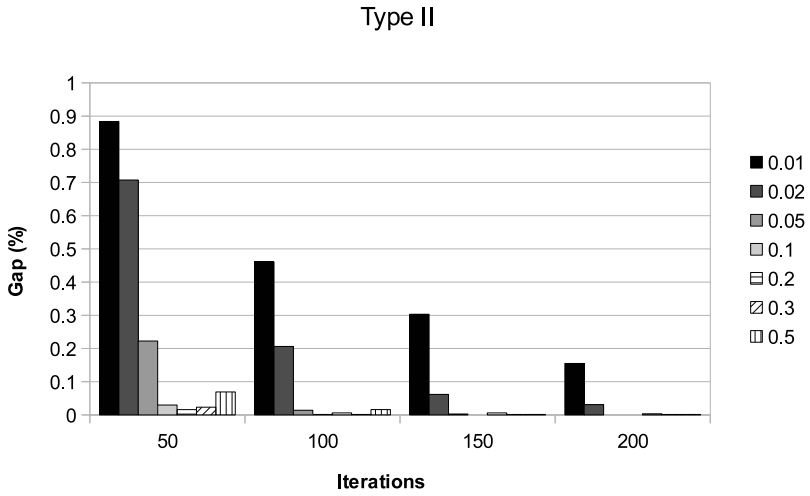

Fig. 4.13: Average optimality gaps obtained while varying the pheromone evaporation rate ρ within {0.01,0.02,0.05,0.1,0.2,0.3,0.5}, and considering Type II cost function.

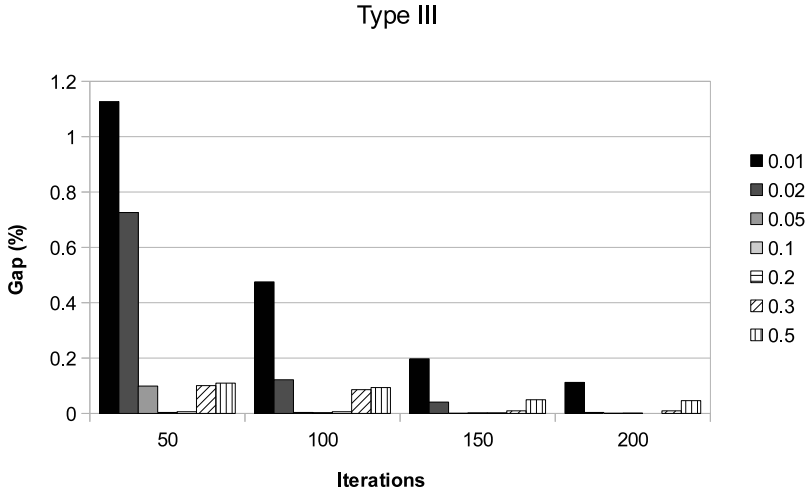

Fig. 4.14: Average optimality gaps obtained while varying the pheromone evaporation rate ρ within {0.01,0.02,0.05,0.1,0.2,0.3,0.5}, and considering Type III cost function.

4.6.1.4. Other Parameters

The stopping criterion may take several forms, such as a bound on the number of iterations, number of solutions evaluated, or even time. In our case, and due to the comparison we want to make with other methodologies in literature, we have set it to a limit on the number of iterations. Accordingly to the results obtained, with the test set, we have decided to limit the number of iterations to 200, as suggested by the results that were obtained while testing the parameters. Results obtained later confirmed this value.

Another parameter that can be tested is Q, the parameter controlling the proportion of pheromone to be deposited in each solution component. After setting the other parameters already mentioned in the previous sections, we came to the conclusion that no major difference was to be found with the variation of this parameter, nonetheless $Q = 2$ was one of the values consistently presenting good results.

Regarding the number of ants allowed to perform local search, experiments have been performed by both using a significantly larger number of ants and a smaller number of ants. While in the former case similar results have been achieved, although with larger computational time requirements, in the latter case the improvement was quite smaller and in many cases nonexistent.

4.6.1.5. Final Parameters

The results reported above were obtained only with the training set. We proceeded our experiences with the whole set of problems having used the final parameter values that can be seen in Table 4.1, which were the ones with the best average results for the training set.

Tab. 4.1: Final parameter values configuration for the Ant Colony Optimization procedure.

Parameter	Value
α	1
β	2
ρ	0.1
Q	2
p_{best}	0.5
τ_0	1000000
Number of ants	n
Number of iterations	200

4.6.2. Comparing Our Results with the Ones in Literature

In order to evaluate the efficiency of our algorithm, we compare our results with the optimum values, whenever possible. For the Type I cost function we have the optimum values obtained with the software CPLEX 9.0, for all problem instances. For cost functions Type II and Type III, and for problems with 10 up to 19 nodes the Dynamic Programming algorithm reported in (Fontes et al, 2006b) provides an optimal solution. For problems with 25 up to 50 nodes and function Type III we calculate the gap by comparing our results with upper bounds reported in (Fontes et al, 2003). We also compare the results obtained with our algorithm with results obtained with a Hybrid Genetic Algorithm (HGA) reported in (Fontes and Gonçalves, 2007). Although the results obtained with the HGA for Type II cost function were never published, the authors were kind enough as to make them available to us.

Let us start by analysing the impact of the local search procedure on the results regarding the solution quality (gap). In order to do so, we have calculated the ratio between the gaps obtained with HACO1 (ACO with LS) and with the ACO algorithm. A value below 100 denotes an improvement brought in by incorporating LS, otherwise we may conclude that no improvement was achieved. Table 4.2 summarizes the average ratios obtained by group and by problem size. Please recall that, for problems with 40 and 50 nodes only the first 5 groups are available. Gap results improved by HACO1 are indicated in bold.

There are two groups that are consistently improved with the introduction of local search, they are group 1 and 6, at least for Type I and III cost functions. Regarding cost function Type II group 4 is the one benefiting the most with the introduction of local search. This is very curious because both group 1 and 6 have a V/F ratio of 0.01 while group 4 has a V/F ratio of 2. Therefore, by observing the results obtained with ACO, we cannot defend neither the argumentation of Kennington and Unger (1976) stating that the difficulty for solving fixed-charge NFPs increases with V/F ratio, nor the argumentation of Palekar et al (1990) stating that the most difficult problems to be solved are the ones with intermediate values. Since local search has improved the performance of our algorithm, regarding the quality of the solutions obtained, we have hereafter abandoned the ACO algorithm and only report results for the HACO1.

Before continuing with the analysis of the results, let us provide some details about the implementation of the HGA developed in (Fontes and Gonçalves, 2007). In the HGA approach the authors use the following parameter settings: 10 times the number of nodes as the population size; a crossover probability (CProb) of 70%; the top (TOP) 15% of chromosomes are copied to the next generation; the bottom (BOT) 15% of chromosomes of the next generation are randomly generated; the remaining 70% are obtained by crossover;

Tab. 4.2: Average ratios between HACO1 and ACO obtained with the introduction of local search into the ACO algorithm, for each of the three cost functions considered, and classified by group and by problem size.

	Groups									
Number of nodes	1	2	3	4	5	6	7	8	9	10
Type I										
10	100	100	100	100	100	**99,95**	100	100	100	100
12	**99,94**	100	100	100	100	**99,96**	100	**99,98**	100	100
15	**99,47**	100	100	100	100	**99,98**	100	100	100	100
17	**99,90**	100	100	100	100	**99,999**	100	100	100	100
19	100	100	100	100	**99,97**	100	100	100	100	100
25	**99,86**	100	100	100	100	**99,91**	100	100	100	100
30	**99,96**	100	100	100	100	**99,36**	100	100	100	**99,98**
40	**99,26**	**99,98**	**99,998**	**99,98**	100	-	-	-	-	-
50	**99,72**	**99,99**	100	**99,998**	100	-	-	-	-	-
Type II										
10	100	100	100	100	100	100	100	100	100	100
12	100	100	100	100	100	100	100	100	100	100
15	100	100	100	**99,98**	100	100	100	100	100	100
17	100	100	100	100	100	100	100	100	100	100
19	**99,97**	100	**99,90**	100	100	100	100	**99,94**	100	**99,89**
25	100	100	**99,89**	**99,99**	100	100	100	100	**99,97**	**99,997**
30	100	100	**99,997**	**99,99**	100	100	100	100	100	**99,997**
40	100	**99,98**	100	**99,91**	100	-	-	-	-	-
50	100	100	**99,996**	**99,95**	**99,99**	-	-	-	-	-
Type III										
10	100	100	100	100	100	**99,94**	100	100	100	100
12	**99,85**	100	100	100	100	**99,64**	100	100	100	100
15	**98,71**	100	100	100	100	**99,73**	100	100	100	100
17	**99,82**	100	**99,99**	100	100	**99,91**	100	100	100	100
19	100	100	100	100	100	100	100	**99,95**	100	100
25	**99,85**	100	100	100	100	**99,94**	100	100	100	100
30	**99,91**	**99,70**	100	100	100	**99,31**	100	**99,97**	100	**99,99**
40	**99,03**	**99,91**	**99,999**	100	100	-	-	-	-	-
50	**99,66**	**99,95**	**99,998**	100	100	-	-	-	-	-

the fitness function is given by the cost; and finally the number of generations allowed is 100.

These values were chosen accordingly to the results obtained with a pilot study on the parameter settings where all possible combinations between the following values were considered: TOP $= (0.10, 0.15, 0.20)$, BOT $= (0.15, 0.20, 0.25, 0.30)$, CProb $= (0.70, 0.75, 0.80)$, population size $= (2, 5, 10, 15)$. The HGA algorithm also incorporates a local search procedure that uses nodes priorities to select the arcs to enter and to exit the solution. The local search heuristic is applied to all solutions in all generations, i.e. to $1000 \times N$ solutions.

We provide results for the HACO1 algorithm and compare them with previous results

obtained with the HGA already mentioned. Gaps are calculated, for both of them and for Type I cost function, using the optimum value obtained with CPLEX. In Table 4.3 we have the average times spent to run the algorithms and the gaps for both of them, as well as the time spent by CPLEX. Problems with 60 up to 120 nodes were only generated for this work, therefore we do not present results on the performance of the HGA. It should be noticed that all algorithms (HACO1, CPLEX, and HGA) were run on the same computer.

Tab. 4.3: Average computational results obtained for cost function Type I, grouped by problem size ranging from 10 up to 120 nodes.

	HGA		HACO1		CPLEX
Sizes	Gap	Time	Gap	Time	Time
10	0.005	0.82	0	0.08	0.31
12	0	1.23	0	0.12	0.19
15	0	2.11	0	0.22	0.24
17	0	3.15	0	0.32	0.27
19	0	4.00	0	0.42	0.41
25	0	9.51	0	0.85	0.58
30	0	14.61	0	1.49	0.69
40	0.005	31.67	0	3.77	1.27
50	0	59.22	0	7.71	2.03
60	-	-	0	15.96	3.08
80	-	-	0	46.41	21.37
100	-	-	0	115.63	82.46
120	-	-	0	226.20	1280.57

As we can see, for problems with up to 50 nodes, HACO1 is able to find an optimal solution in all results outperforming HGA that failed to reach that value in all 5 runs of one problem instance with size 10 and one problem instance with size 40.

Computational times (required for a full run of the algorithms) increase with problem size, as it is expected, see Figs. 4.15 and 4.16. Nonetheless, HACO1 times are considerably smaller than those of the HGA, HACO1 being up to 11 times faster. Furthermore, the increase in the computational times is much larger for HGA than for HACO1, which is most likely related to the number of solutions evaluated by each algorithm. The HGA time values reported were obtained by implementing the HGA in Visual Basic 6 and the computational experiments were performed on the same computer of the HACO1 ones. As both the HGA and the HACO1 algorithm use the number of nodes in the problem to calculate the number of solutions constructed in each iteration, we can compute the number of solutions evaluated by each of the algorithms. The HGA evaluates $10 \times n \times 100$ solutions and for each of these solutions the local search heuristic is applied to look for a

better neighbour solution. The HACO1 evaluates $n \times 200$ solutions and the local search heuristic is applied to only $5 \times n$ of such solutions, representing 80% less solutions evaluated, not accounting for the ones searched for by the local search procedure. This means that although with similar results, see the gaps, the HACO1 algorithm has the advantage of requiring much less computational effort due to the reduced number of solutions evaluated. Ants can locate and converge to the optimum value within a small number of walks.

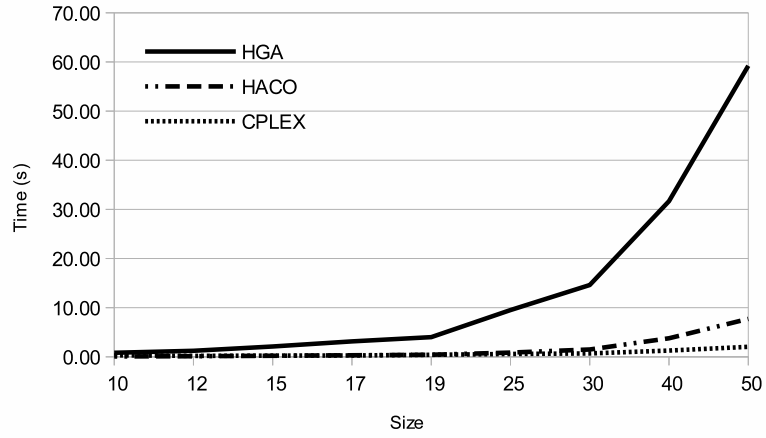

Fig. 4.15: Computational time results obtained for Type I cost functions, for problem sizes from 10 up to 50 nodes.

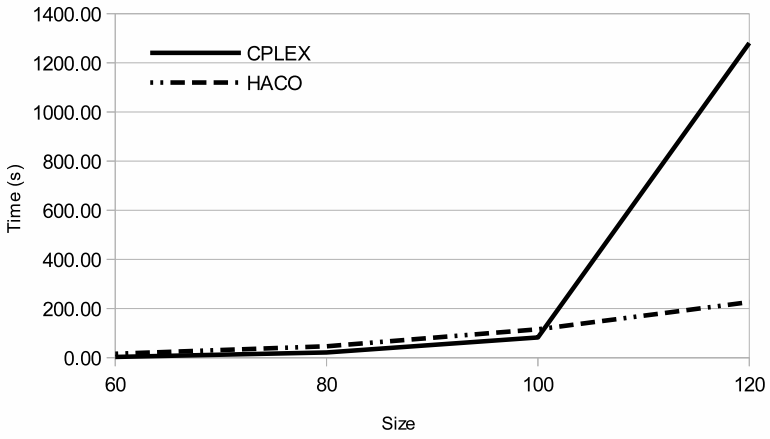

Fig. 4.16: Computational time results obtained for Type I cost functions, for problem size from 60 up to 120 nodes.

Ahuja et al (2002) provided a rule of thumb, as cited in (Gouveia et al, 2011), that says “the larger is the neighbourhood of each solution, the better is the quality of the

locally optimal solutions". However, latter on, Orlin and Sharma (2004) gave examples of neighbourhoods, with very differing sizes containing the same set of local optimums. Our results seem to agree with the conclusions of the latter authors.

The HACO1 computational time requirements are similar to the ones of CPLEX, for problems with up to 100 nodes. Regarding larger size problem instances the average time spent by CPLEX and by the HACO1 algorithm is quite different, see Fig. 4.16. Although for problem sizes with up to 100 nodes running times are of the same order of magnitude, when the problem size is larger the computational time spent by CPLEX shows a major increase. It should be noticed that when increasing the problem size from a 100 to 120 nodes, the computational time of HACO1 increases twice, while the time of CPLEX increases about 15 times.

Tab. 4.4: Average computational results obtained for cost function Type II, grouped by problem size ranging from 10 up to 50 nodes.

	HGA	HACO1	
Size	Time	HACO1/HGA	Time
10	0.84	100.00	0.08
12	1.32	100.00	0.12
15	2.24	100.00	0.21
17	3.27	100.00	0.32
19	4.10	99.99	0.41
25	8.99	100.00	0.84
30	15.36	100.00	1.43
40	33.46	100.00	3.63
50	60.66	100.00	7.44

In Table 4.4 we have the results obtained with cost functions of Type II for the same algorithms. The quality of the solutions is measured by a ratio computed as a percentage using the values obtained with the HGA. Both HGA and HACO1 have good performances regarding the problems for which an optimal value is known, and HACO1 was always able to find an optimal solution. Furthermore, HACO1 found a solution with the same cost as the HGA to all but one problem, a problem instance with 19 nodes, for which it was able to improve the solution of the HGA. Regarding the time spent to run the algorithm, HACO1 is up to 11 times faster than HGA, as can be seen in Fig. 4.17. It should be noticed that the increase of the computational time with problem size is again much smaller for the HACO1. This indicates that the difference in performance for the two algorithms is expected to grow with problem size.

Results for Type III cost functions are presented in Table 4.5, where BS stands for

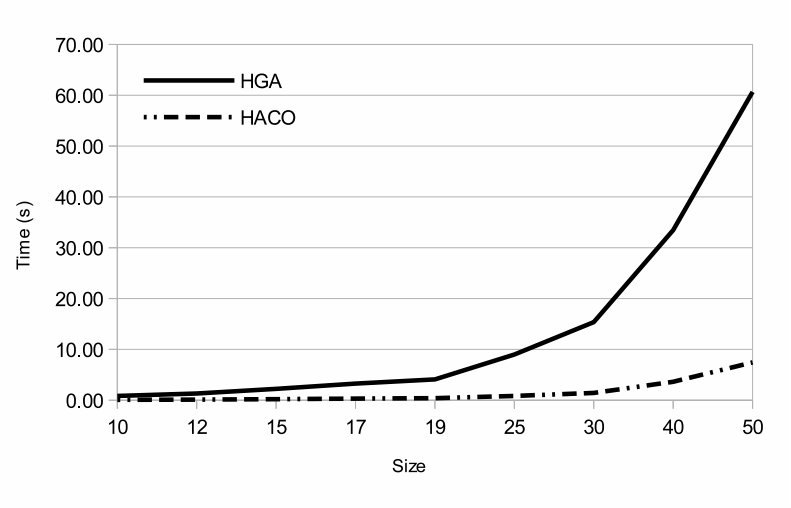

Fig. 4.17: Computational time results by varying problem size, for Type II cost functions.

the best known solution. For problems for which an optimal value is known HACO1 was always able to find it.

Tab. 4.5: Average computational results obtained for cost function Type III, grouped by problem size ranging from 10 up to 50 nodes.

	HGA	HACO1		
Size	Time	HACO1/HGA	Time	HACO1/BS
10	0.90	100.00	0.09	100
12	1.42	100.00	0.14	100
15	2.50	100.00	0.25	100
17	3.74	100.00	0.35	100
19	4.63	100.00	0.48	100
25	10.28	100.00	0.97	100.72
30	18.39	100.00	1.64	99.13
40	42.70	100.00	4.02	99.90
50	77.62	100.00	8.39	99.94

Again the computational time, see Fig. 4.18, is considerably smaller for HACO1 in relation to HGA. And again, increasing needs of computational times grow much faster for the HGA.

4.7. Summary

In this chapter, we have described the single source uncapacitated concave minimum cost network flow. A literature review on both exact and heuristic methods addressing

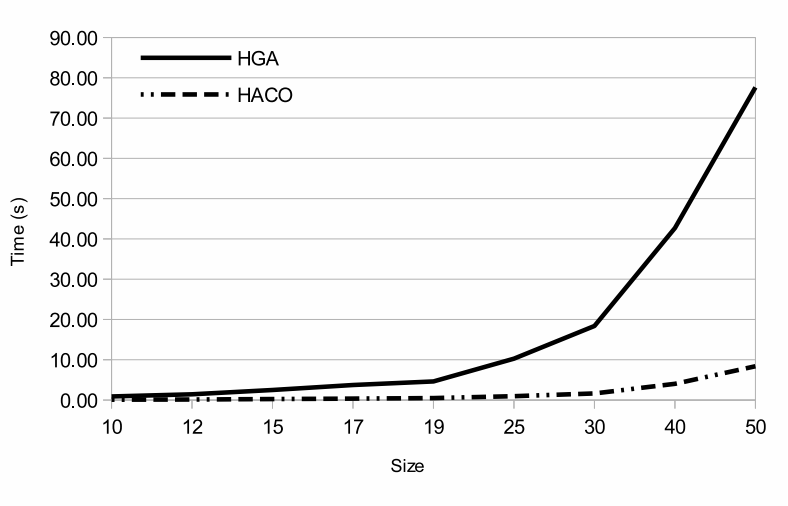

Fig. 4.18: Computational time results by varying problem size, for Type III cost functions.

this problem was initially made. A new method based on ant colony optimization, never before used to solve these problems, as far as we know of, was present and discussed. This algorithm is hybridized with a local search procedure, in order to perform exploitation. Finally, computational results have been reported along with a comparison between the HACO1 algorithm and a dynamic programming algorithm, a hybrid genetic algorithm, and CPLEX. A study on the behaviour of the ACO algorithm with different parameter values was also presented.

A preliminary version of the work presented in this chapter has been published as a full article in the proceedings of an international conference (Monteiro et al, 2011a) and the final results have been published in Journal of Heuristics (Monteiro et al, 2012b).

5

Hop-Constrained Minimum Cost Flow Spanning Trees

5.1. Introduction

The Minimum Spanning Tree (MST) problem is a very well-known combinatorial problem where the objective is to find a tree spanning all nodes in a network while minimizing the total costs incurred. The MST problem can be used to model several applications specially in the transportation and in the communications fields as is the case of (Frey et al, 2008) and (Hwang et al, 2007), that use it to model multicast networks. In its simplest version the MST problem can be solved in polynomial time.

An extension to the MST problem that limits the number of arcs allowed on each path from the root node to any other node is called the Hop-constrained Minimum Spanning Tree (HMST). The hop-constrained version of the MST is NP-hard (Gouveia, 1995).

In this chapter, we look into another generalization of the MST problem closely related to the HMST problem since in addition we consider that flow requirements at client nodes may have values other than the unit and be different across clients. Different flow requirements at client nodes is a characteristic of several service networks, such as telecommu-

nications, water, gas, and electrical power. Furthermore, the cost functions considered involve two components: a setup or fixed cost incurred by using the arc and a routing cost (nonlinearly) dependent on the flow being routed through the arc.

In the following sections, we concentrate our study in this new problem. Initially, we define the MST along with some problems closely related to the HMST. The HMST problem is also widely debated and mathematical formulations for this problem are given. Next, the hop-constrained minimum cost flow spanning tree problem is introduced and discussed and its mathematical formulation is presented. A literature review of the most recent works on the subject area is presented. Then, the ACO algorithm is provided, as well as, the local search procedure hybridized in it, followed by the discussion and conclusions of the results obtained.

5.2. Minimum Spanning Tree: Problem and Extensions

The minimum spanning tree problem is a very well-known combinatorial problem where the objective is to find a tree spanning all nodes in a network while minimizing the total costs incurred. This combinatorial problem is frequently used to model problems in the area of telecommunications, where there is a central device, for example a hub, that must be linked to a set of remote terminals, see Fig. 5.1.

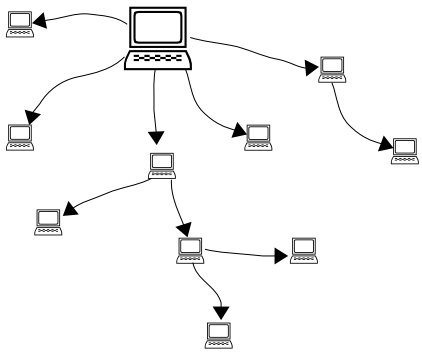

Fig. 5.1: A hub network.

The MST is a very important problem because it is widely used in practice. In (Frey et al, 2008) we can find an example of such an application to solve a problem of multicast routing. Multicast routing is a largely studied problem that involves the distribution of a package of information to a set of demand points. Live video streaming and internet TV are examples of such networks. The same information is to be accessed simultaneously and in real time by several client computers. It would be unreasonable for the source to replicate and send a package of information to every single client computer,

both due to high resource costs and to a quality degradation, in case of a large number of clients. Therefore, MSTs are usually associated with these problems in order to choose the most interesting points, from the network point-of-view, where a replication (copy) of the packages is to be made in order to send them further down in the network.

One of the most popular heuristic methods used to construct MSTs in weighted graphs is due to Prim, and is called Prim's algorithm (Prim, 1957). Given a set N of nodes and a set A of arcs the algorithm starts by randomly choosing a node to start from, let us say node a, and adds it to a set of used nodes $N_u = \{a\}$. Then, the algorithm searches the set of arcs whose parent node is in N_u and whose child node is not in N_u, that is arcs $\{(i,j) : i \in N_u, j \notin N_u\}$, in order to identify the one with the lowest cost of them all. After such an arc (a,w) is identified, it is added to the set of used arcs $A_u = \{(a,w)\}$ and node w is added to the set of used nodes $N_u = \{a,w\}$. The algorithm continues until all nodes are in N_u, which is the same as saying until a tree spanning all nodes in N has been constructed. Dijkstra (1959) developed another algorithm, the well-known Dijkstra's algorithm, that is used to solve shortest path problems. Dijkstra's algorithm is close to Prim's algorithm because in order to find the shortest path between two nodes in a graph, whenever a decision has to be taken regarding the arc to enter the solution, the cheapest arc from the set of feasible arcs is always the chosen one.

Both algorithms are greedy algorithms, in the sense that they both make the optimal local choice in every step of the construction of the tree, i.e, they always choose the cheapest arc to enter the solution tree.

Prim's and Dijkstra's algorithms can be used to find an optimal solution for some problems. In the case of more complex problems, they are examples of good starting points for other heuristics as they can generate fairly good local optima *per se*. Some of such heuristic algorithms are reviewed in Section 5.3.

The MST is a problem with various extensions, each one impelled by the necessity to model a practical problem. Bellow we describe three particular cases of such extensions, which are the ones closely related to the problem we are addressing in this work. For a recent and exhaustive survey on MSTs and solution methods we suggest the work by Mares (2008).

5.2.1. Diameter Constrained Minimum Spanning Tree Problem

The diameter constrained minimum spanning tree problem, or d-constrained Minimum Spanning Tree (d-MST) problem, is the one of finding the least cost tree spanning all nodes in a network provided that the path linking every pair of nodes has d or less arcs.

These problems are considered in undirected networks.

This problem can be used to model situations where a minimization of the number of messages communicated among processors per critical section is required, in order to minimize interference on the transmissions along the network (Abdalla and Deo, 2002).

Examples of solutions for this problem are provided in Figs. 5.2 and 5.3. Let us suppose that we are considering a complete network with 7 nodes, and that both figures represent minimum cost solutions for this problem. If a diameter of 3 is considered, this means that besides being the least cost solution, every path between any two nodes in the solution must not have more than 3 arcs. Figure 5.2 presents an unfeasible solution for the problem because the path between nodes 1 and 7 has four arcs. Paths linking all the other pairs of nodes are within the limit.

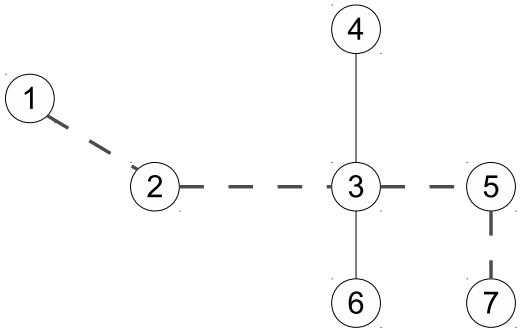

Fig. 5.2: An unfeasible solution for a 3-constrained Minimum Spanning Tree problem.

If we wanted to make this solution feasible, a possibility would be to remove arc $\{5, 7\}$ and add arc $\{2, 7\}$, as can be seen in Fig. 5.3. In this solution for the 3-constrained MST every pair of nodes is within 3 arcs of reach.

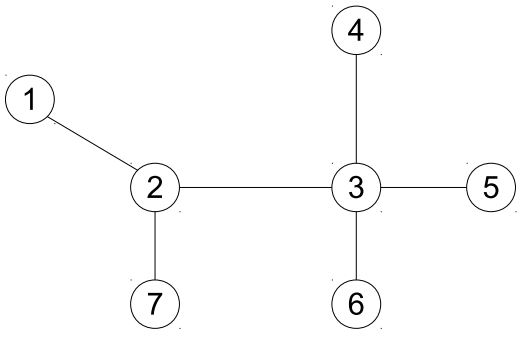

Fig. 5.3: A feasible solution for a 3-constrained Minimum Spanning Tree problem.

It was observed by Handler (1978) that feasible solutions for this problem presented a well defined and different structure dependent on whether the value of the diameter D is an odd or an even number. If, on the one hand, the value for the diameter is even a central

node t is clearly identified such that any other node from the solution tree is at most at $D/2$ arcs away from t. If, on the other hand, D is odd, then a central arc $[t, w]$ (in the case of undirected graphs) exists such that any other node from the solution tree is at most at $(D-1)/2$ arcs away from its closest node t or w.

Let us now introduce a formal definition and mathematical formulation for the problem, as given in (dos Santos et al, 2004), that has into account the observation of Handler (1978). We will only reproduce the case of an even D. For the odd case and alternative formulations please refer to (dos Santos et al, 2004) and the references therein. The formulation makes use of a directed graph $G' = (N, A)$, where N is the set of nodes and A is the set of existing directed arcs. Graph G' is obtained from the original undirected graph $G = (N, E)$ by replacing every undirected arc $[i, j] \in E$ with two directed arcs (i, j) and $(j, i) \in A$, with $i < j$, with costs $c_{ij} = c_{ji}$. As we are dealing with the even D case, let us consider $L = D/2$.

The formulation introduces an artificial central node t into the graph such that we have a new graph $G'' = (N', A')$ with $N' = N \cup \{t\}$ and $A' = A \cup \{(t, 1), \ldots, (t, |N|)\}$. Two decision variables are associated with this problem: x_{ij} taking the value 1 if arc $(i, j) \in A'$ is in the solution tree and the value 0 otherwise, and u_i giving the number of arcs in the path from t to $i \in N'$. The mathematical model for the d-MST problem when D is even can then be formulated as follows:

Model 3 A mathematical model for the d-MST problem when D is even.

$$\text{min:} \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{5.1}$$

$$\text{s.t.:} \quad \sum_{j\in N} x_{tj} = 1, \tag{5.2}$$

$$\sum_{i\in N'} x_{ij} = 1, \quad \forall_{j\in N}, \tag{5.3}$$

$$u_i - u_j + (L+1)x_{ij} \leq L, \quad \forall_{(i,j)\in A'} \tag{5.4}$$

$$0 \leq u_i \leq L+1, \quad \forall_{i\in N'} \tag{5.5}$$

$$x_{ij} \in \{0, 1\}, \quad \forall_{(i,j)\in A'}. \tag{5.6}$$

The objective is to minimize the objective function, as given in (5.1). Constraint (5.2) makes sure that the artificial central node t is in the solution and that it is linked to a single node in N, while constraints (5.3) impose that each node has only one incident arc. Constraints (5.4) and (5.5) make sure every path from the central node t to every other node $i \in N$ has at most $L+1$ arcs, where the extra arc is due to the artificial central node

that has been introduced in the graph. Undirected arcs $[i,j] \in E$ such that $x_{ij} = 1$ or $x_{ji} = 1$ in a feasible solution for Model 3 define a spanning tree T of G with diameter less than or equal to D.

5.2.2. Capacitated Minimal Spanning Tree Problem

The goal of a Capacitated Minimal Spanning Tree (CMST) problem is to find a minimum cost tree spanning all nodes provided that each subtree pending from the root node does not span more than k nodes. This problem is specially related with the design of centralized telecommunication networks, so that the transmission capacity of each link directly connecting a terminal to the central hub can be optimized. As transmissions do not happen all the time, by linking several terminals to one line (arc) its capacity can then be shared by the traffic of several terminals (Gouveia and Martins, 1999).

This problem is NP-complete for $2 < k \leq n/2$, (Papadimitriou, 1978). A possible formulation for the CMST problem can be obtained by regarding the problem as a capacitated network flow problem. Model 4 presents such a formulation, as given in (Gavish, 1983).

Model 4 A mathematical model for the CMST problem.

$$\text{min:} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{5.7}$$

$$\text{s.t.:} \quad \sum_{i\in N} x_{ij} = 1, \quad \forall_{j\in N\setminus\{t\}}, \tag{5.8}$$

$$\sum_{i\in N} y_{ij} - \sum_{i\in N\setminus\{t\}} y_{ji} = 1, \quad \forall_{j\in N\setminus\{t\}} \tag{5.9}$$

$$x_{ij} \leq y_{ij} \leq (K-1)x_{ij}, \quad \forall_{i,j\in N\setminus\{t\}} \tag{5.10}$$

$$x_{tj} \leq y_{tj} \leq Kx_{tj}, \quad \forall_{j\in N\setminus\{t\}}, \tag{5.11}$$

$$x_{ij} \in \{0,1\}, \quad \forall_{(i,j)\in A} \tag{5.12}$$

$$y_{ij} \geq 0, \quad \forall_{(i,j)\in A}. \tag{5.13}$$

In this model, decision variable x_{ij} indicates whether or not arc (i,j) is in the solution tree and y_{ij} represents the flow passing through arc (i,j). The objective function to be minimized is given in (5.7). Constraints (5.8) guarantee that every node is in the solution and constraints (5.9) ensure that the flow requirements of the demand nodes are satisfied. The capacity in the arcs is guaranteed by constraints (5.10) and (5.11) that, together with constraints (5.9) prevent cycles from entering the solution. Note that the flow in arc

(i, j) gives the number of nodes that are disconnected from the root node t if arc (i, j) is disconnected from the tree. Consequently, no subtree with more than K nodes can be connected to the root node. The nature of the variables is stated by constraints (5.12) and (5.13).

Figure 5.4 presents a solution tree for a problem with 12 demand nodes and a root node. If the capacity parameter $k = 6$ it is easy to see that the tree represents an unfeasible solution because the subtree on the right spans more than 6 nodes, whereas the solution presented in Fig. 5.5 provides a feasible solution for the same capacity parameter value.

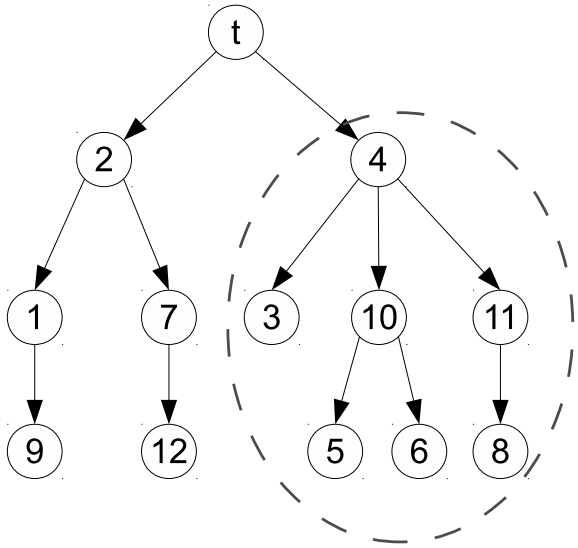

Fig. 5.4: An unfeasible solution for a Capacitated Minimum Spanning Tree problem.

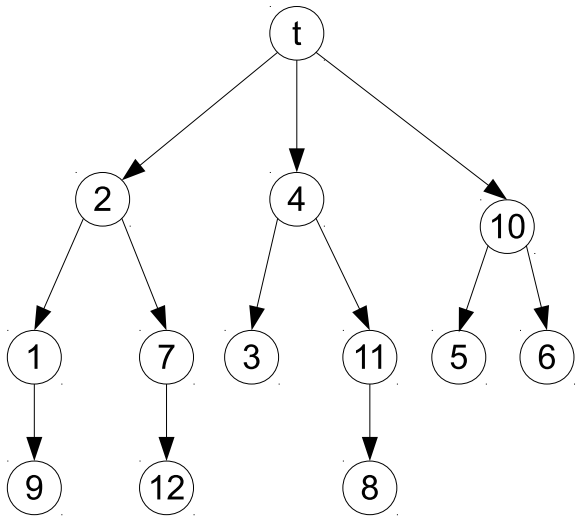

Fig. 5.5: A feasible solution for a Capacitated Minimum Spanning Tree problem.

For CMST problems, large values of k are usually associated with deep solution trees, whereas small values of k usually imply the solution tree to take a wider form, with many

subtrees pending from the root node.

5.2.3. The Hop-Constrained Minimum Spanning Tree Problem

The hop-constrained minimum spanning tree problem is an extension of the MST. In it, besides finding a tree spanning all nodes at minimum cost, the number of arcs in each path from the source node to every other node must not exceed a given integer H value, which is called the hop parameter. The addition of a maximum number of arcs in each path is usually related to reliability and availability issues.

In a problem where a central hub is sending packages of information to terminal computers, availability is usually associated with the probability of perfect functioning of all links between the central hub and each one of the terminals. Whereas reliability is the probability that the transmission of the package will not be interrupted by any external factor resulting in the fail of a link, (Woolston and Albin, 1988). Therefore, we can easily conclude that the increase on the number of arcs between the central hub and the terminal computer (hops) will result in a decrease on the reliability and on the availability of the network. It has even been shown by Woolston and Albin (1988) that the overall gain in terms of service quality compensates for the increase on the total costs when imposing a maximum limit on the number of hops on the network.

In Fig. 5.6 we provide a graphical example of a failure in a network with 15 terminals. To the left we have a network with a maximum of 6 hops. In the case of a failure in the link signalled, in a dashed line, there would be nine affected terminals, to which no information would be delivered. To the right we have another configuration for the network, spanning the same number of nodes, but with a maximum of 4 hops observed. In this case, if there is a fail in the same link the number of terminals affected is much lower, in this case, it would be 3 terminals.

In addition, these constraints, called hop-constraints, can also be associated to lower delay times in a multi-drop lines network, where packages of information may have to queue before reaching their destination. The total delay time is dependent on the number of arcs between the origin and the destination node (Akgün, 2011), therefore it will increase with the number of hops in the network.

The HMST problem is NP-hard because it contains a special case, when $H = 2$, which is a NP-hard version of the Simple Uncapacitated Facility Location (SUFL) problem with the client sites and the facility potential sites matching. Therefore, as the SUFL problem is known to be NP-hard the 2 hop-constraint MST problem is also NP-hard (Gouveia, 1995; Dahl, 1998).

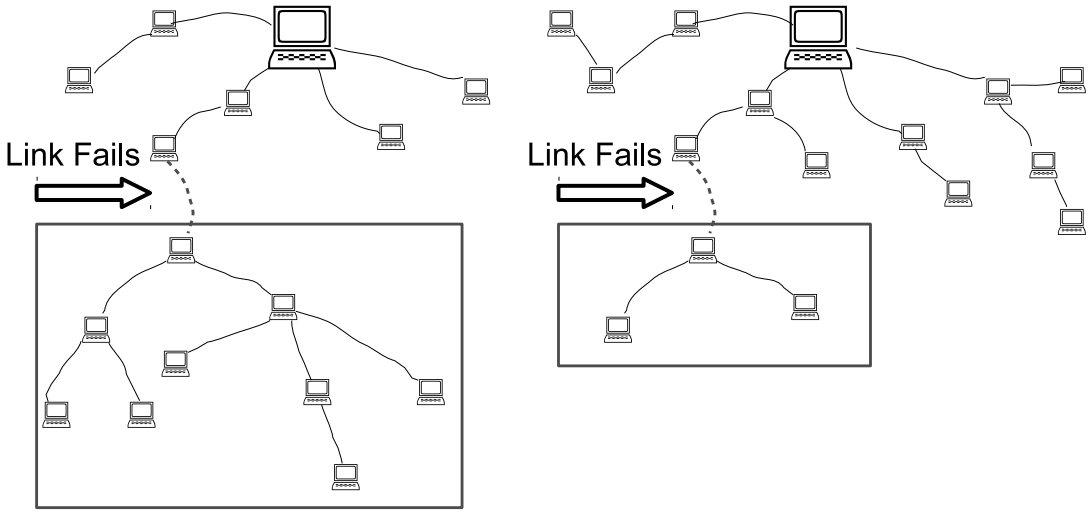

Fig. 5.6: A network with link failure.

5.2.4. Formulations for the HMST Problem

The mathematical model formulation for the HMST problem has been evolving in an attempt to improve the performance of the solution methods, while generating lower bounds for the HMST problem. In this section, we provide two examples of possible formulations for the HMST problem. Nonetheless, we are aware that the first formulation is not much used in literature, whereas the second model, based on a multicommodity network flow formulation, has been the means of many improvements.

Let us start by defining the HMST problem in a graph and providing the notation hereby used. Consider a directed network $G = (N, A)$, where N is a set of $n + 1$ nodes and A is a set of m available arcs (i, j). Furthermore, each arc is associated to a cost c_{ij}. A HMST is a problem dealing with the minimization of the total costs incurred with a network spanning all nodes. The commodity flows from a single source node t to the n demand nodes $i \in N \setminus \{t\}$. In addition, the maximum number of arcs on a path from the source node to each demand node is constrained by a hop parameter, H.

A Miller-Tucker-Zemlin based formulation

The mathematical formulation for the HMST problem that is given in (Gouveia, 1995) is based on the Miller-Tucker-Zemlin subtour elimination constraints, originally defined for the TSP.

This formulation, besides using the usual arc variables x_{ij} also includes node variables u_i, defined as follows:

$$x_{ij} = \begin{cases} 1, & \text{if arc } (i,j) \text{ is in the solution tree,} \\ 0, & \text{otherwise,} \end{cases}$$

u_i = number of arcs between the root node t and a demand node i.

Model 5 A Miller-Tucker-Zemlin constraints based mathematical model for the HMST problem.

$$\text{min:} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{5.14}$$

$$\text{s.t.:} \quad \sum_{i\in N} x_{ij} = 1, \quad \forall_{j\in N\setminus\{t\}}, \tag{5.15}$$

$$nx_{ij} + u_i \leq u_j + (n-1), \quad \forall_{i,j\in N\setminus\{t\}}, \tag{5.16}$$

$$1 \leq u_i \leq H, \quad \forall_{i\in N\setminus\{t\}}, \tag{5.17}$$

$$x_{ij} \in \{0,1\}, \quad \forall_{(i,j)\in A}, \tag{5.18}$$

$$u_i \in \mathbb{N}, \quad \forall_{i\in N\setminus\{t\}}. \tag{5.19}$$

In Model 5, (5.14) represents the cost function to be minimized and constraints (5.15) guarantee that every demand node is in the solution and has only one arc entering it. Constraints (5.16) are the Miller-Tucker-Zemlin subtour elimination constraints, from now on hereby referred to as MTZ constraints, given in (Miller et al, 1960). Originally, these constraints were used to eliminate cycles from the TSP problem and are also incorporated in this formulation of the HMST to deal with the possible circuits in the solution. Thus, whenever arc (i,j) is in the solution tree, i.e. $x_{ij} = 1$, Eq. (5.16) boils down to $u_i + 1 \leq u_j$, that is the same as stating that the path from root node t to demand node i must have, at least, one less arc than the path from root node t to demand node j. If constraints (5.16) are applied to a solution with a cycle then their successive application to every node in that cycle would lead to the condition $u_i < u_i$ which is a contradiction. If arc (i,j) is not in the solution, then $u_i \leq u_j + (n-1)$, which is true for all u_i and u_j. In order to guarantee that the number of arcs between root node t and demand node i does not surpass the maximum allowed number H, the model includes constraints (5.17) defined for every demand node i. These constraints state that at least one arc must separate t from every other demand node. Finally, constraints (5.18) and (5.19) define the binary nature of the decision variables x_{ij} and the positive integer nature of the decision variables u_i.

A multicommodity flow based formulation

Gouveia (1996) provides a general multicommodity flow formulation for the HMST prob-

lem as given bellow in Model 6. In this model, the variables are all based on arcs, whereas on the previous model decision variables u_i were defined for each node in the network. The motivation for this new formulation is due to the good quality reported for relaxations of multicommodity flow formulations.

Two decision variables are defined for this formulation:

$$x_{ij} = \begin{cases} 1, & \text{if arc } (i,j) \text{ is in the solution tree,} \\ 0, & \text{otherwise,} \end{cases}$$

$$y_{ijk} = \begin{cases} 1, & \text{if } (i,j) \text{ is in the path from root node } t \text{ to demand node } k,\ k \neq i, \\ 0, & \text{otherwise.} \end{cases}$$

Model 6 A multicommodity flow based formulation for the HMST problem.

$$\text{min:} \quad \sum_{(i,j)\in A} c_{ij}x_{ij}, \tag{5.20}$$

$$\text{s.t.:} \quad \sum_{i\in N} x_{ij} = 1, \quad \forall_{j\in N\setminus\{t\}}, \tag{5.21}$$

$$\sum_{i\in N} y_{ijk} - \sum_{i\in N\setminus\{t\}} y_{jik} = 0, \forall_{k,j\in N\setminus\{t\}, j\neq k}, \tag{5.22}$$

$$\sum_{i\in N} y_{ijj} = 1, \quad \forall_{j\in N\setminus\{t\}}, \tag{5.23}$$

$$\sum_{(i,j)\in A} y_{ijk} \leq H, \quad \forall_{k\in N\setminus\{t\}}, \tag{5.24}$$

$$y_{ijk} \leq x_{ij}, \quad \forall_{(i,j)\in A}, \forall_{k\in N\setminus\{t\}}, \tag{5.25}$$

$$x_{ij} \in \{0,1\}, \quad \forall_{(i,j)\in A}, \tag{5.26}$$

$$y_{ijk} \in \{0,1\}, \quad \forall_{(i,j)\in A}, \forall_{k\in N\setminus\{t\}}. \tag{5.27}$$

Equations (5.20) and (5.21) have the same meaning as Eqs. (5.14) and (5.15), respectively. Decision variables y_{ijk} use a third index as in multicommodity network flow models and constraints (5.22) refer to the usual flow conservation constraints. The single hop-constraint in the previous model is separated into two sets of constraints in Model 6. While constraints (5.23) guarantee that one hop is always to be observed in any path between the root node and any demand node j. Constraints (5.24) state that no more than H arcs are allowed in each path from the root node t to each demand node k. The coupling constraints (5.25) state that an arc (i,j) can only be in the path between root node t and demand node k if the arc is included in the solution. Constraints (5.26) and (5.27) state the binary nature of the decision variables.

Another model using four indexed decision variables can also be found in (Gouveia and Requejo, 2001), however we do not include it in this section as it is based on relax-

ations of Model 6.

We now provide a graphical example of an unfeasible solution, with respect to the hop-constraint, and of a feasible solution, while considering a complete digraph, as given in Figs. 5.7 and 5.8, respectively. In these examples we are looking for a minimum cost tree spanning 12 demand nodes and limiting the paths to a 4-hop length. To simplify, let us suppose that both trees are optimized with respect to the minimum cost. The tree given in Fig. 5.7, has two demand nodes, node 12 and node 3, whose path from the root node exceeds the 4 hops imposed, with both violating arcs signalled in a dashed line. Thus, this is an unfeasible solution regarding the hop-constraint.

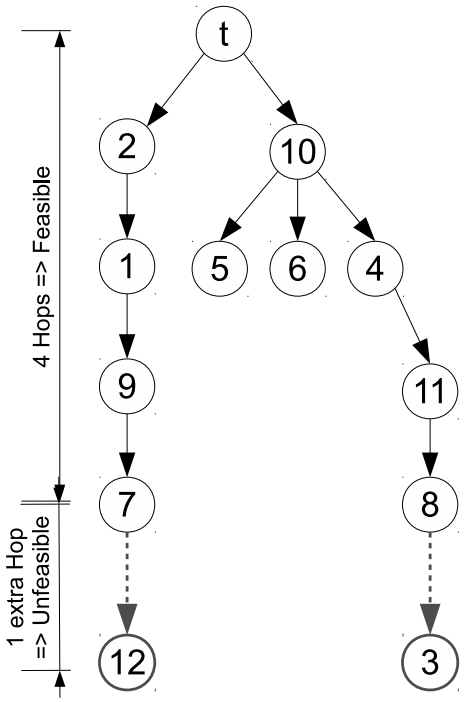

Fig. 5.7: An unfeasible solution for a Hop-constraint Minimum Spanning Tree problem.

In Fig. 5.8 we have a similar tree but the 4-hop constraint is no longer violated. Therefore, this solution is a feasible one. If we look closer at it, we can conclude that the only difference between the two trees is that arcs $(7, 12)$ and $(8, 3)$, which were violating the hop-constraint in the tree of Fig. 5.7, are not present in this new solution. Instead, new arcs $(5, 12)$ and $(4, 3)$, signalled in grey, are now introduced such that each demand node is within a 4-hop reach from the root node t.

5.2.5. The Hop-Constrained Minimum cost Spanning Tree Problem with Flows

The Hop-constrained Minimum cost Spanning Tree problem with Flows (HMFST) is an extension of the HMST problem. In addition to minimizing total costs incurred with a

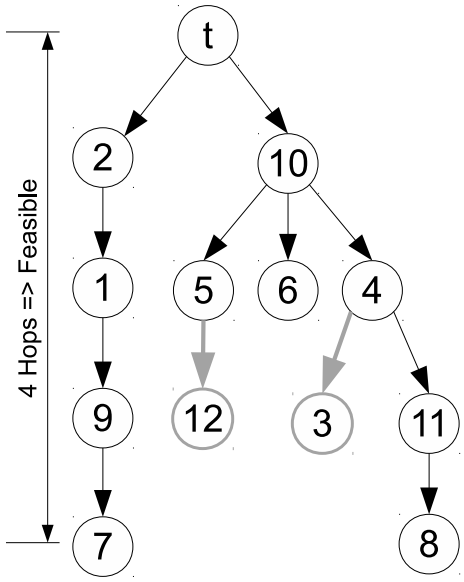

Fig. 5.8: A feasible solution for a Hop-constraint Minimum Spanning Tree problem.

tree spanning all its nodes, and to the constraint on the maximum number of arcs in the paths between the source node and each of the demand nodes, in HMFST problems we also have to find the flows to be routed through each arc. Therefore, the main difference between the HMST and the HMFST problem is that the latter problem allows nodes to have different flow demands.

The need for such a problem is obvious when we think of a transportation network that can be highly compromised if one or several of its links are interrupted. For example, if a problem in a section of a railway track blocks the passage of the train transporting some highly degradable commodity, high costs may be incurred by the owning company as time goes by. Therefore, the reliability and availability of the network is of a major importance, and limiting the number of arcs (routes) in the network is a way of assuring minimal damage.

5.2.5.1. Defining and Formulating the HMFST Problem

The HMFST problem considers the selection of a tree spanning all its nodes in a network in such a way that costs are minimized. In addition, the flows to be routed through each arc have to be found and the maximum number of arcs in each path between the source node and each of the demand nodes is limited. As the uncapacitated version of the problem is being considered, there are no upper bounds on the arcs capacity.

Formally, the problem can be defined as follows.

Consider a directed network $G = (N, A)$, where N is a set of $n + 1$ nodes and A is

a set of m available arcs (i, j). The number of available arcs is at most $n \cdot (n + 1)$ since there is only one source node. A HMFST is a problem dealing with the minimization of the total costs g_{ij} incurred with the network while satisfying the nodes demand d_j. The total demand of the network, D, is given by the summation of all node demands. The commodities flow from a single source node t to the n demand nodes $j \in N \setminus \{t\}$. In addition, the maximum number of arcs on a path from the source node to each demand node is constrained by a hop parameter, H. The mathematical programming model that is given next for the HMFST problem is an adaptation of the hop-indexed single-commodity flow formulation for the constrained MST problem by Gouveia and Martins (1999), which in turn is based on the one developed by Gavish (1983). An important feature has been added by Gouveia and Martins to the mathematical formulation by Gavish, which is the introduction of an extra index h identifying the position of an arc in the solution tree counting from the source node t, i.e, the number of hops to the arc. Therefore, h can only take values from 1 to H.

Considering the notation summarized bellow:

t - source node,

n - number of demand nodes,

d_j - demand of demand node $j \in N \setminus \{t\}$,

$$y_{ijh} = \begin{cases} 1, & \text{if arc } (i, j) \text{ is in position } h \text{ in the solution tree,} \\ 0, & \text{otherwise,} \end{cases}$$

x_{ijh} - flow on arc (i, j) which is in position h in the solution tree,

g_{ij} - cost of arc (i, j).

Model 7 provides a mixed-integer formulation for the HMFST problem.

Please note that in Model 7 we do not consider the following variables: variables y_{tjh} and x_{tjh} because any arc leaving the root node cannot have a hop value larger than 1; variables y_{ij1} and x_{ij1}, where $i, j \in \{1, \ldots, n\}$ because in this case we are dealing with arcs not directly connected to the root, therefore they cannot be in position (hop) 1; finally, variables y_{iij} and x_{iij} are also disregarded because we are not considering loops in the tree. The objective in this problem is to minimize total costs incurred with the tree spanning all its nodes, as given in (5.28). Constraints (5.29) guarantee that every node is in the solution. Constraints (5.30) are called the flow conservation constraints, and they state that the difference between the flow entering a node and the flow leaving a node must be the demand of the node. Furthermore, they also state that if the flow enters a node through an arc in position h, then the flow leaves that same node through an arc in position $h + 1$. Constraints (5.31) are called the coupling constraints and guarantee that flow is only sent through used arcs. These constraints together with constraints (5.30) guarantee that no

Model 7 A mixed-integer mathematical programming model for the HMFST problem.

$$\text{min:} \quad \sum_{i \in N} \sum_{j \in N \setminus \{t\}} \sum_{h=1}^{H} g_{ij}(x_{ijh}, y_{ijh}), \tag{5.28}$$

$$\text{s.t.:} \quad \sum_{i \in N} \sum_{h=1}^{H} y_{ijh} = 1, \quad \forall_{j \in N\{t\}}, \tag{5.29}$$

$$\sum_{i \in N} x_{ijh} - \sum_{i \in N \setminus \{t\}} x_{ji,h+1} = d_j \sum_{i \in N} y_{ijh}, \quad \forall_{j \in N \setminus \{t\}}, \ \forall_{h \in \{1,\ldots,H-1\}}, \tag{5.30}$$

$$y_{ijh} \leq x_{ijh} \leq D y_{ijh}, \quad \forall_{i \in N}, \ \forall_{j \in N \setminus \{t\}}, \ \forall_{h \in \{1,\ldots,H\}}, \tag{5.31}$$

$$x_{ijh} \geq 0, \quad \forall_{i \in N}, \ \forall_{j \in N \setminus \{t\}}, \ \forall_{h \in \{1,\ldots,H\}}, \tag{5.32}$$

$$y_{ijh} \in \{0, 1\}, \quad \forall_{i \in N}, \ \forall_{j \in N \setminus \{t\}}, \ \forall_{h \in \{1,\ldots,H\}}. \tag{5.33}$$

cycles are allowed in the solution. Constraints (5.32) and (5.33) state the nonnegative and binary nature of the model variables. It is assumed that the commodity produced by the source node t equals the sum of all the demands d_j, i.e.,

$$\sum_{j \in N \setminus \{t\}} d_j + d_t = 0, \tag{5.34}$$

where d_t is the demand of node t. As t is the source node, its demand is represented by a negative value, hence the form of Eq. (5.34).

The most important effect about this model is that, by formulating the HMFST as a network flow problem, we are able to take advantage of the knowledge already gathered for network flow problems. Such knowledge enables us to immediately identify the characteristics of local optimum solutions, of the search space, and almost all sorts of problem parameters affecting HMFST problems.

5.2.5.2. Cost Functions

Given the easiness of approximation of any cost function by the first few terms of a Taylor Series, four types of polynomial cost functions are considered in this work:

- **Type 1:**

$$f_{ij}(x_{ij}) = \begin{cases} b_{ij} \cdot x_{ij} + c_{ij}, & \text{if } x_{ij} > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5.35}$$

- **Type 2:**

$$f_{ij}(x_{ij}) = \begin{cases} 0, & \text{if } x_{ij} = 0, \\ b_{ij} \cdot x_{ij} + c_{ij}, & \text{if } x_{ij} \leq D/2, \\ b_{ij} \cdot x_{ij} + c_{ij} + b_{ij}, & \text{otherwise.} \end{cases} \tag{5.36}$$

- **Type 3:**

$$f_{ij}(x_{ij}) = \begin{cases} 0, & \text{if } x_{ij} = 0, \\ b_{ij} \cdot x_{ij} + c_{ij}, & \text{if } x_{ij} \leq D/2, \\ b_{ij} \cdot x_{ij} + c_{ij} - b_{ij}, & \text{otherwise.} \end{cases} \tag{5.37}$$

These first three cost functions F1, F2, and F3 consider linear routing costs b_{ij} per unit of flow routed through arc (i, j), as well as, fixed costs c_{ij}. Furthermore, there is an additional discontinuity point in F2 cost functions by adding b_{ij}, and in F3 cost functions by subtracting b_{ij}, when the flow passing through an arc is higher than half the total demand D. The fourth cost function is represented by complete second-order polynomials and is initially concave and then convex, thus also including an additional discontinuity point:

- **Type 4:**

$$f_{ij}(x_{ij}) = \begin{cases} 0, & \text{if } x_{ij} = 0, \\ -a_{ij} \cdot x_{ij}^2 + b_{ij} \cdot x_{ij} + c_{ij}, & \text{if } x_{ij} \leq D/2, \\ a_{ij} \cdot x_{ij}^2 + b_{ij} \cdot x_{ij} + c_{ij}, & \text{otherwise.} \end{cases} \tag{5.38}$$

All cost functions consider a_{ij}, b_{ij}, and $c_{ij} \in \mathbb{Z}^+$. Please note that we have dropped the h index in the flow variables x_{ij} because the cost of an arc does not depend on its position in the tree.

The use of these cost functions, except for F1, follows the work of Fontes and Gonçalves (2012). Cost function F1 was herein introduced with the purpose of assessing the behaviour of the algorithm regarding the additional discontinuity point, when considered along with F2 and F3. Furthermore, the introduction of F4 cost function allows us to infer if the same behaviour is to be expected from our algorithm if a more complex cost function is also to be accounted for.

5.2.5.3. Test Problems

In order to test the algorithm that was developed we downloaded the Euclidean test set available from (Beasley, 2012). The set is divided in ten groups $\{g_1, g_2, \ldots, g_{10}\}$ with different ratios between variable and fixed costs, V/F. Each of these subsets has three problem instances. Furthermore, the number of nodes considered is 10, 12, 15, 17, 19, 25, 30, 40, and 50. For the problems with 40 and 50 nodes, there are only 5 groups defined. For further details on these problems please refer to (Fontes et al, 2003). Therefore, there is a total of 240 problem instances to be solved ten times for each of the four cost functions considered F1, F2, F3, and F4 and each of the four H values, where $H \in \{3, 5, 7, 10\}$.

It is important to report that from the 240 available problems, for $H = 3$ and for $H = 5$, only 165 and 233 problems, respectively, have feasible solutions. For $H = 3$ none of the problem instances with size 40 and 50 has a feasible solution, while for $H = 5$ two problem instances with size 40 and five problem instances with size 50 have no feasible solution. Thus, all things considered, we have solved a total of 3512 problem instances ten times each.

5.3. Solution Methods for the HFMST and Some Related Problems

The general MST problem has been the subject of many studies and has been solved by several techniques, both heuristic and exact. But, as it happens, literature studies addressing the HMFST problem are scarce. The hop-constrained MST problem, which is the closest problem, will then be used to introduce some of the solution methods that have been used. The literature review presented here has as its main focus recent works. Therefore, it does not give an historical perspective on the subject. It is also our purpose to provide an outlook on the different classes of algorithms that have been used for this problem rather than concentrating in one or two methods in particular.

We start this section by reviewing solution methods for the HMST problem. The HMST problem has been studied for some time now, however the development of lower bound schemes is the most popular method that has been used to approach it. Lower bounds are usually obtained based on the solution of different relaxation techniques applied to the mathematical programming models developed to represent HMST problems.

Gouveia (1995) provides models for the HMST problem involving node variables and using extensions of Miller-Tucker-Zemlin subtour elimination constraints, the latter ones used in order to prevent the solution to incorporate cycles (see Model 5 and its reformulation in Section 5.2.4). The author derives stronger models, based on Model 5, as follows: the MTZ_L is given by the Linear Programming (LP) relaxation of the binary

constraints (5.18); the $EMTZ_L$ model is obtained from this latter one by substituting constraints (5.16) and (5.17) by

$$Hx_{ij} + u_i \leq u_j + (H-1), \quad \forall_{i,j \in N \setminus \{t\}}. \tag{5.39}$$

By observing that

$$u_i - u_j \leq (H-1), \quad \forall_{i,j \in N \setminus \{t\}}, \tag{5.40}$$

the MTZ constraints are lifted in model $EMTZ_L$ by using (Desrocher and Laporte, 1991) liftings. Therefore, a stronger third model, called $L1EMTZ_L$, is obtained. In this model, the constraints (5.39) are replaced by

$$(H-2)x_{ji} + Hx_{ij} + u_i \leq u_j + (H-1), \quad \forall_{i,j \in N \setminus \{t\}}; H \geq 2, \tag{5.41}$$

since we cannot have $x_{ij} = x_{ji} = 1$. A fourth model, named $L2EMTZ_L$, and valid for $H \geq 3$, results also from the lifting of constraints (5.39) in model $EMTZ_L$, by substituting them by

$$\sum_{k \in N \setminus \{t\}; k \neq i} x_{kj} + (H-3)x_{ji} + Hx_{ij} + u_i \leq u_j + (H-1), \quad \forall_{i,j \in N \setminus \{t\}}; H \geq 3, \tag{5.42}$$

based on the use of additional information on the values of decision variables u_i and u_j whenever an arc (i,j) is not in the solution and arc (k,j) is. Finally, the fifth model, $L3EMTZ_L$, is obtained by including in the same model constraints (5.41) and (5.42). In addition, subtour elimination constraints, as given by Eq. (5.43), earlier developed in (Edmonds, 1968) are also considered.

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \quad \forall_{S \subseteq N \setminus t} \text{ and } |S| \geq 2. \tag{5.43}$$

These constraints are then relaxed in lagrangean relaxation fashion and the model is solved by a subgradient method. The solutions of the LR models provide a directed minimum spanning tree which are lower bounds to optimal solutions. These solutions do not guarantee the satisfaction of the hop-constraints. Upper bounds for the HMST problem are then obtained by applying a heuristic as follows. The heuristic uses the directed minimum spanning tree found by the subgradient method as an initial solution.

Then, all nodes which stand at a distance from the root node t of $H \times k+1$ are disconnected from their parent node and are directly connected to the root node. This is repeated while there are paths exceeding H. Problems with up to 40 nodes were solved but the results were not as expected, and the author concludes that a stronger formulation must still be achieved, in order to tighten the LP bounds.

Latter on, Gouveia (1996) formulates the HMST problem as a multicommodity flow model (see Model 6 in Section 5.2.4), and suggests lower bounding schemes to solve it also based on Lagrangean Relaxation (LR), improving upon his latter work. The author starts by developing a formulation for the undirected case of the HMST problem. Then, a formulation for the directed case is derived by replacing each undirected arc $[i, j]$ with two directed arcs (i, j) and (j, i) each of which having the cost c_{ij}, earlier associated with $[i, j]$, thus obtaining a symmetric cost matrix. The hop-constraints (see Eq. (5.24) in Model 6) are lifted deriving two new sharper models. The first one is due to the observation that if an arc (k, v) is in the solution tree then the number of arcs in the path between root node t and demand node k cannot be equal to H and is given by,

$$\sum_{(i,j)\in A} y_{ijk} \leq H - x_{kv}, \quad \forall_{k,v\in N\setminus\{t\},k\neq v}. \tag{5.44}$$

The second one, developed to decrease the number of variables in the model, uses only the subset of the previous constraints involving the lowest cost arc (k, v_k) leaving demand node k,

$$\sum_{(i,j)\in A} y_{ijk} \leq H - x_{k,v_k}, \quad \forall_{k,v_k\in N\setminus\{t\}}. \tag{5.45}$$

Since this new formulation still generates large linear programming models, an arc elimination test is used in order to reduce the size of the models. The test removes any arc (i, j) for which $c_{ij} \leq c_{tj}$, that is, when its cost is larger or equal to the cost of linking node j directly to the root node t. Every time an arc is removed from the problem, $n - 1$ constraints of the form (5.25) are eliminated from the directed model. Two LRs are also developed for the directed model (Model 6). The first LR associates nonnegative multipliers of the form λ_{ijk} to constraints (5.25), and them dualizes them. The second LR associates lagrangean multipliers δ_{jk} to constraints (5.22) and (5.23) and nonnegative multipliers w_k to constraints (5.24), and then dualizes them. A directed subtour elimination constraint is also incorporated into both relaxed models. Each of these latter models are separated into two subproblems, one regarding only x_{ij} variables and the other, equivalent to n hop constrained shortest path problems, involving only y_{ijk} variables. The heuristic defined in (Gouveia, 1995) is improved by incorporating more complex arc ex-

changes. A node i is eligible if it is at a distance from the root node of $H \times k + 1$ and if the longest path departing from node i (also called the depth of node i) has less than $H - 1$ arcs. Nodes satisfying both conditions are linked to any other node j provided that the sum of the position of node j and the depth of node i do not overcome H. Nodes satisfying only the first condition are linked to the root node, as previously done in (Gouveia, 1995). This heuristic is incorporated in the subgradient optimization procedure (see (Held et al, 1974) for more details) used to obtain the optimal lagrangean multipliers, as a way to transform a spanning tree solution to the relaxed models into a HMST. Problems with fully connected graphs with up to 40 nodes and $H = 3, 4, 5$ are solved by each of the models defined by the author. The results show that the Linear Programming (LP) relaxation bounds for these models are significantly better than those of (Gouveia, 1995) and that the dual lagrangean bound for the LR is strictly better than the bound for the LP.

Gouveia (1998) provides a new lower bounding method, based on the variable redefinition method by Martin (1987), for minimum spanning trees with hop-constraints. A multicommodity flow formulation (MCF) of the problem is given, as in (Gouveia, 1996), see Model 6, and its linear programming relaxation, corresponding to the replacement of the binary constraints (5.26) and (5.27) by the respective nonnegativity constraints $x_{ij} \geq 0$ and $y_{ijk} \geq 0$, is considered (MCF_L). The other model is constructed in several steps. Initially, a lagrangean relaxation of the MCF formulation is performed associating nonnegative multipliers of the form λ_{ijk} to the coupling constraints (5.25), and then dualizing them the usual lagrangean way (MCF_λ). This new relaxed problem can be divided into two subproblems. The first subproblem involves only the x_{ij} variables, and can be easily solved by inspection. The second one involves the y_{ijk} variables and can be further separated into $|N - 1|$ subproblems, one for each $k \in N \setminus \{t\}$, each one corresponding to a shortest path problem between source node t and node k with at most H hops. Following the definition of the DP proposed by Martin (1987), the shortest-path problems can be easily solved by the DP, and an extended hop-path formulation can be made for the original hop-constrained problem. The new formulation is given by considering the binary arc variables z_{ijq} which assume the value 1 if from state i, which is in position $q - 1$, we move to state j in position q. The combination of the extended hop-path formulation with the MCF formulation results in a new model ($HDMCF$) involving the usual x_{ij} variables and new variables z_{ijqk}, the latter ones with an extra index when compared with y_{ijk}, indicating whether an arc (i, j) is in in position q of the path from the source node t to the demand node k. Finally, the binary constraints of the model are relaxed and the new relaxed model ($HDMCF_L$) is solved. The results indicate that the bounds of the $HDMCF_L$ model are close to the optimal value. Computational times for solving the MCF_L are larger than those of $HDMCF_L$, in part due to a less compact model.

Gouveia and Martins (1999) introduce an extended compact formulation for the capacitated MST problem, by observing that the problem can be reduced to the $HMST$ problem due to the one unit of traffic produced by every demand node, that is considered by the authors. This new formulation introduces another index in the variables, the hop index, identifying the position of an arc in the solution. Furthermore, a new set of inequalities, called hop-ordering inequalities, are developed based on the formulation of (Gavish, 1983) for a capacitated MST problem, to limit the flow in each arc which is not connected to the root. Latter on, Gouveia and Martins (2000) improve upon these lower bounds by proposing several levels of aggregation. The new model aggregates into one single variable all the variables associated with a given arc beyond a certain position P. This allows for the creation of a hierarchy of hop-indexed models and a reduction on the number of variables in the models. Lower bounds are computed by three iterative methods based on the solution of a sequence of lagrangean relaxations of the hop-indexed models with varying P values: the first one uses reduction tests to eliminate variables from the model; the second one adds each one of the hop-ordering inequalities which are violated by the solution; and the third one includes generalised subtour elimination constraints to the model.

The relaxation proposed by Gouveia (1998) produces very tight bounds, nonetheless it is very time consuming. To overcome this problem Gouveia and Requejo (2001) develop a new Lagrangean Relaxation (LR) to be applied to the hop-indexed MultiCommodity Flow formulation ($HDMCF$) of the HMST problem (explained earlier in this section). To derive the new LR of the model, ($HDMCF_\lambda$), the flow conservation constraints are associated with lagrangean multipliers and dualized the usual lagrangean way. The problem is further simplified, for a given value of the lagrangean multipliers, by observing that the optimal values for the binary variables z_{ijqk} can be obtained by dropping the linking constraints $\sum_{q=1}^{H} z_{ijqk} \leq x_{ij}$ and the binary constraints $z_{tj1k} \in \{0,1\}$ and $z_{ijqk} \in \{0,1\}$ from $HDMCF_\lambda$. This modified relaxation may be separated into two simple inspection subproblems: one involving only the x_{ij} binary variables, and the other involving only the z_{kkqk} binary variables. After the optimal values for the x_{ij} and z_{ijqk} are obtained, an approximation of the optimal multipliers is also obtained, using a subgradient optimization method. The authors compare results obtained with three models: the $HDMCF_\lambda$, the $HDMCF_L$, and the MCF_λ, the two latter ones defined in (Gouveia, 1998). The arc elimination procedure proposed in (Gouveia, 1996) is applied before solving each instance of the $HDMCF_\lambda$ and $HDMCF_L$ models, reducing considerably the number of arcs for some of the problems. Problem instances with 20, 40, and 60 nodes in complete graphs are solved. The new lagrangean relaxation $HDMCF_\lambda$ is found to be superior to

the one developed in (Gouveia, 1998) both in terms of bounds and computational time.

Gouveia et al (2007) model the HMST problem as a Steiner tree in a layered directed graph. The identification of each layer is associated with the hop-constraint value h, where $h = 1, 2, \ldots, H$, and nodes respecting h will be copied into the corresponding layer h. Therefore, layer 0 has only the source node assigned to it, layer 1 will have a copy of all the nodes reachable from the source node within one arc, an so on. Each node in layers 1 and $H - 1$ will be linked to their corresponding node in layer H, and will have a 0 cost associated. The other arcs of the problem are represented by links between nodes in sequential layers until layer H is reached, creating a graph where a spanning tree in the original graph now corresponds to a Steiner tree. Lower and upper bounds are computed by using a dual ascent heuristic, closely related to the one provided in (Uchoa et al, 2001), and a primal heuristic SPH-Prim given in (Takahashi and Matsuyama, 1980), respectively, and finally a cutting plane algorithm developed by (Hao and Orlin, 2002) is applied. The overall approach is shown to be very efficient, solving problems with up to 160 nodes in reasonable times.

A more recent work on the computation of lower bounds for the HMST problem is proposed in (Akgün, 2011). The author develops three Miller-Tucker-Zemlin (MTZ) based formulations. The first model $HMST/SD$ incorporates an adaptation of (Sherali and Driscoll, 2002) constraints for the asymmetric TSP, where a linearization of the non-linear product terms is performed. Constraints (5.46), (5.47), and (5.48) thus obtained are added to the basic MTZ formulation of the HMST problem (see Model 5 in Section 5.2.4), as well as a new set of more dedicated coupling constraints (refer to (Akgün, 2011) for more details), resulting into the $HMST/SD$ model:

$$\sum_{\forall_{i \in N \setminus \{t\}}} y_{ij} + 1 = u_j, \forall_{j \in N \setminus \{t\}}, \tag{5.46}$$

$$x_{ij} \leq y_{ij}, \forall_{i,j \in N \setminus \{t\}}, \tag{5.47}$$

$$y_{ij} \geq 0, \forall_{(i,j) \in A}, \tag{5.48}$$

where decision variables y_{ij}, given by $y_{ij} = u_i x_{ij}$, represent the rank-order of arc (i, j) in the path it is used in. Constraints (5.46) relate the rank-order of arc (i, j) with the position of node j, constraints (5.47) are the usual coupling constraints, and constraints (5.48) refer to the nonnegative nature of the decision variable y_{ij}. Topological constraints are derived from observations on the structure of feasible solutions to HMST problems. The second model $HMST/SD1$ is given by incorporating constraints related to the information about the nature of the nodes, i.e. whether a node is a leaf or not, directly

into $HMST/SD$. The third model $HMST/SD2$ also incorporates an extra set of constraints that can generate feasible solutions with special structures regarding node and arc labelling. Models $HMST/SD1$ and $HMST/SD2$ dominate the MTZ-based models with the best linear programming bounds in the literature.

The interested reader is referred to the comprehensive survey by (Dahl et al, 2006), and to the references therein for a discussion on formulations, including alternative formulations for the HMST problem based on natural design variables and on an exponential sized set of constraints. Solution methods comprising lower bounds for the hop-constrained MST problem, including techniques such as lagrangian relaxation or column generation, are also discussed.

Regarding the development of good heuristic methods to solve the HMST problem, not much has been done, apart from the works we review bellow.

In (Fernandes et al, 2007) five heuristic procedures are developed to solve the HMST problem based on ideas proposed for the capacitated minimum spanning tree, taking advantage of the problem similarities. Firstly, a savings heuristic, which belongs to the class of constructive heuristics, is given. In this heuristic the initial solution tree is such that all nodes are linked to the source node. Then, arcs in the solution tree are swapped with arcs not present in the solution tree that represent the best savings. Secondly, a second order algorithm is developed. This algorithm uses different starting solutions generated by the savings heuristic. To generate different initial solutions, the savings heuristic is given a set of allowed arcs S_1, as well as a set of prohibited arcs S_2, differing from solution to solution. Then, in each iteration the savings heuristic is used to find a solution, based on S_1 and on S_2. The solution obtained and its respective cost are updated, if necessary. S_1 and S_2 are modified accordingly, and the process starts all over. An improvement was also proposed by the authors by transforming S_1 and S_2 into permanent included and excluded arcs, whenever a solution improves the incumbent solution. The heuristics tested differ in the modification rules used to create S_1 and S_2. The first couple of heuristics do not use set S_2. Heuristic I1 considers simple inhibitions, that is, it defines S_1 as the set of arcs of the previous solution to be excluded from the next solution, provided that they are not linked to the source node. Then, the savings heuristic is run considering the exclusion of one arc (in S_1) at a time. Heuristic I2 is similar to I1 but instead it considers prohibiting two arcs at a time. The second couple of heuristics do not consider S_1. Heuristic J incorporates in S_2, for each node, the cheapest arcs incident to the node and the arcs with the minimum cost which are closer to the source node. The savings heuristic is run forcing one arc at a time to enter the solution. Heuristic J4 uses as candidate arcs to be included in the next solution, that is S_1, the set consisting of the four cheapest arcs incident to

each node, provided that they are not linked to the source node and they are not yet in the solution. The fifth and last heuristic is called ILA and is a modified version of I2 where at each iteration a new solution is calculated for each arc to be prohibited. Then, all arcs of that solution are possible candidates for prohibition. Problems with 40 and 80 nodes, considering complete graphs, are generated both with the root node at the center (TC) and at the corner of a grid (TE). An elimination test given in (Gouveia, 1996) is used, which substantially reduces the size of the TC problems and also of some TEs. The heuristic results were compared and the ILA heuristic, although with higher time requirements, had the best performance. The authors were able to improve some results for upper bounds reported in literature.

In another work, Gouveia et al (2011) use Dynamic Programming (DP) and heuristics based on the exchange of arcs and on node-level exchanges to solve HMST problems. The level of a node is defined as the maximum number of arcs a node can have between itself and the source node and is the base used on a restricted DP formulation of the problem. Given a positive integer value d, where d can take any value up to the number of demand nodes n, the state-space restriction rule allows the move of at most d nodes between any consecutive levels. The objective of this rule is to eliminate some states of the original formulation to make the DP more tractable. A restriction on the state-transition rule is also included in order to reduce the computational effort for computing the cost of an optimal tree associated to the states with a depth of at most k. Parallel shift moves starting and ending at different pairs of overlapping levels are prohibited. Path moves are also not allowed, for example, sending nodes from say level 1 (L1) to level 2 (L2) and from L2 to level 3 (L3), where $L1 < L2 < L3$ or $L1 > L2 > L3$. Three neighbourhood constructions are defined, based on the restricted DP formulation: shift, swap, and shift/swap neighbourhoods, where the latter one is the set of all neighbour solutions obtained either with a swap or a shift move. A standard arc-exchange neighbourhood was also developed, as well as a method combining arc-exchange and swap and shift moves. These neighbourhoods were used to construct five distinct heuristics and the method combining arc-exchange, swap and shift moves was found to be the best one. The authors have also compared their results with the ones obtained with the repetitive heuristics that can be found in (Fernandes et al, 2007), and the heuristics with the new neighbourhoods have outperformed the latter ones.

Although very often used to solve many optimization problems, there has not been much work on heuristic methods to solve the HMST problem. Although, in order to solve realistic sized problems it has been indicated that heuristic methods are much more adequate (Dahl et al, 2006). Therefore here, we review some problems which are closely related to the hop-constrained MST such as, for example, the capacitated minimum span-

ning tree problem (where there is a constraint on the size of each subtree of the root) or the d-constraint Minimum Spanning Tree problem (where each node cannot have more than d incident arcs), just to mention but a few.

Ant colony optimization algorithms are very popular since they have been initially developed and have been applied to solve a broad set of problems, MST based problems also included. For example, Reimann and Laumanns (2006) use it as part of the heuristic they have developed to solve the CMST problem. The construction of the solution is based on a method to solve the Capacitated Vehicle Routing Problem (CVRP). In a VRP the objective is to find a route for each vehicle of a fleet at disposal, all beginning and ending at the same depot, such that the flow requirements of a set of demand nodes is satisfied and that the total cost is minimized. If the VRP is capacitated then the routes have to be found provided that the capacity of each vehicle is not violated. The authors argue that for both problems the demand nodes have to be divided into clusters, due to the capacity constraints. The main difference between a solution for the CMST and the CVRP is that for each cluster, a MST has to be found for the CMST problem, while for the CVRP it is a TSP route. The algorithm, which is based on the savings algorithm first proposed by Clarke and Wright (1964), starts by linking all the demand nodes directly to the source node t, each node forming a different cluster. The cost to serve each demand node i is made of two components, $c_{ti} + c_{it}$, the first component representing the cost to travel from the source node t to the demand node i, and the second component representing the cost for returning from demand node i to the source node t. Therefore, the total cost of this initial solution is calculated by summing the costs of all clusters. Whenever two clusters are merged, a saving is obtained. For example, the initial cost with the clusters associated with demand nodes i and j is given by $c_{ti} + c_{it} + c_{tj} + c_{jt}$. But, if we merge these two clusters by introducing the arc (i, j), the saving that is obtained is given by $c_{it} + c_{tj} - c_{ij}$, which is the difference between the costs of the disappearing arcs (i, t) and (t, j) and the cost of the newly introduced arc (i, j). Obviously, this saving may be positive, and then the merge results in a lower cost solution, or negative if the merge results in a higher cost solution. The next phase is to solve the associated CVRP with an ant based algorithm, where the probability function uses the savings information as the heuristic information. After a solution is found local search based on swap moves is applied while the solution is improved. Finally, Prim's algorithm (Prim, 1957) transforms the solution of the CVRP into an MST, that is, a solution for the CMST problem. The authors show that their computational results are competitive with the ones obtained with several heuristic algorithms in literature by obtaining the same, or slightly higher, gaps but with much lower running times. Also, they provide a study on the influence of problem characteristics and of ACO parameter values on the solution quality.

Another ant based algorithm is adopted by Bui and Zrncic (2006) to solve a special case of the MST that limits the number of incident arcs d to each node, and is known as the degree-constrained MST. For this problem, only variable costs c_{ij} are considered. The ants do not construct an entire solution but rather parts of the solution. Each ant is placed at a distinct node and is allowed to travel to another node, thus choosing an arc. These arcs are not used directly to construct a tree but instead they are used to update the pheromone levels of each arc. After evaporation, the quantity of pheromone deposited in each arc (i, j) is equal to the initial pheromone level assigned to it and is also proportional to the number of ants that traversed the arc. A candidate list of arcs is constructed based on their pheromone levels. A modified version of Kruskal's algorithm is used to select arcs from this list to construct a solution tree, while respecting the degree constraints. Initial pheromone levels are set such that lower cost arcs have higher pheromone concentration. The values are given by $InitPher_{ij} = (M - c_{ij}) + (M - m)/3$, where M and m are the maximum and minimum arc costs, respectively, and c_{ij} is the cost of arc (i, j). This way, it is guaranteed that the minimum initial pheromone value is no smaller than 4 times the maximum pheromone value. A maximum and a minimum value allowed for pheromone levels are defined, but using $maxPher = 1000((M-m)+(M-m)/3)$ and $minPher = (M-3)/3$, instead of the more commonly pheromone limits used, and that can be found in (Stützle and Hoos, 1997). The algorithm was used in 143 fully connected graphs with the number of nodes ranging from 15 to 1000, and with $d = 2, 3, 4, 5$. The authors were able to improve upon the great majority of the best optimality gaps reported in the literature (on average, the improvement obtained was around 51%), that were previously obtained with a genetic algorithm.

More recently, Neumann and Witt (2010) make use of a simple ACO algorithm, called 1-ANT, to solve the MST problem. They compare the results obtained, regarding the running time behaviour, using two algorithms differing on the solution construction technique, in problems where pseudo-Boolean functions are to be optimized. One such ant algorithm incorporates a modified version of Broder's construction algorithm (Broder, 1989) to create a solution. This modified version uses pheromone information to choose the next arc to be included into the solution (provided that the child node has not yet been visited) rather than choosing it uniformly at random. The other ant algorithm incorporates an incremental solution construction named Kruskal-based construction technique. In this case a new arc is included into the solution tree if it does not introduce a cycle into the network. A polynomial upper bound is proved for the algorithm using the modified Broder construction technique. However, the algorithm using the Kruskal-based construction technique results in much lower running times.

Genetic algorithms were first used to solve the MST problem in the mid 90's and they

are still one of the most popular heuristic methods, in all optimization areas. Zhou et al (1996) use a GA to solve degree-constrained MSTs. Initially, they make a comparison between three types of encodings for the chromosome, in order to choose the one that will be used in the GA they propose. The encodings are: arc encoding, associating an index to each arc thus constructing chromosomes with $n-1$ genes; node encoding, which was the selected one, is based on the Prüfer number encoding allowing the use of $n-2$ genes to encode a tree with n nodes; and finally an arc and node encoding which associates a bias value to each node and each arc. The crossover between two parent chromosomes is achieved by using the one-cut-point crossover, with a probability of 0.5, and mutation is performed by choosing an arbitrary gene and randomly altering its value, using 0.01 as the probability of changing the gene. The degree constraint is guaranteed by replacing each violating gene with another gene that is not in violation of the degree constraint. The selection of the chromosomes to integrate the next generation is done with a mixed strategy using the *roulette wheel* and $(\eta+\lambda)-$ selection. The authors apply their GA only to a literature problem with a 9-node complete graph.

In the Steiner Tree Problem (STP), given a set of nodes, one seeks to establish a network between a subset of nodes, known as basic nodes, provided that the total length (cost) of the network is minimal and that there is a path between every pair of basic nodes. The difference between STP and MST is that extra nodes may be used in the STP if they shorten the total length between nodes. These extra nodes are called the Steiner nodes. If the STP is considered in a graph with a set N of nodes, then its objective, given a subset S of nodes in N, is to find a minimum cost subgraph such that there is a directed path between a given root node $t \in S$ and every basic node $i \in S \setminus \{t\}$. In addition, nodes from the set $N-S$ may also be added to the network if they provide a total cost decrease. Voß (1999) extends the mathematical formulation of the HMST given in (Gouveia, 1995) for the STP with hop constraints, and formulates two models. One model is used to compute optimum values and the other model is used to derive lower bounds. Also, a heuristic algorithm is developed to solve the STP with hop-constraints. The algorithm uses a cheapest insertion heuristic, based on principles of Prim's algorithm, and tabu search. The author starts by developing two cheapest insertion heuristics to construct an initial feasible solution, both of them starting the construction of a solution with a tree consisting of a single root node. The first heuristic finds and inserts into the solution the shortest length arcs between nodes in the solution and nodes not in the solution, provided that hop-constraints are not violated and that no cycles are added to the solution. This procedure is repeated until no more basic nodes are out of the solution and a tree is found, or until no more nodes can be added to the solution without transforming it into an unfeasible solution. Given that this method may lead to feasibility issues in sparse

networks, the author develops a second method to find an initial feasible solution that differs from the first one by allowing the introduction of cycles. After the construction of the solution, if any cycle is identified it is eliminated by inserting into the tree an extra node from the set of Steiner nodes as long as it does not violate the hop-constraint. This initial feasible solution is further improved, at each iteration, by using a tabu search that incorporates a Reverse Elimination Method (REM) to manage the tabu list, first proposed in (Glover, 1990). Initially, a so-called running list is used to store all the attributes of the moves that have been performed during the search. A new tabu list is created at each iteration with the attributes of the moves performed (for example, the arcs that have entered or exited a solution). The backtracing REM procedure changes a solution by tracing the actual running list, from the last attribute entering the list to the first one, in order to identify attributes whose change of status would lead to revisiting an already visited solution in the next iteration or move. While backtracing the running list, another list called Residual Cancellation Sequence (RCS) is updated. Whenever the complement of an attribute on the running list is not already in the RCS, this attribute is added to the RCS, otherwise the complement of the attribute is drawn from the RCS. The tabu list for the next iteration will be made of the complement of the RCS. The problems solved have $60, 80$, and 100 nodes and are generated based on the ones used in (Gouveia, 1998), where the number of available arcs is given by $2.5 \times (n + 1)$, where n is the number of demand nodes. The quality of the lower bounds is found to be much influenced by the value of the hop-constraint and the results obtained with the tabu search are very robust which is an advantage because an optimum value was not always found due to hardware limitations.

Blum and Blesa (2005) address the arc-weighted k-cardinality tree problem, by proposing three metaheuristics, namely tabu search, evolutionary computation, and ant colony optimization. The arc-weighted k-cardinality tree problem is a generalization of the MST problem and aims at finding the least cost subtree with exactly k arcs. This problem is also known as the k-Minimum Spanning Tree (k-MST) problem. The tabu search approach makes use of two tabu lists, the inList which stores arcs recently added to the solution, and the outList which stores arcs that were recently removed. The length of the tabu lists, also known as tenure, is dynamic in the sense that it changes during the execution of the algorithm, increasing in more difficult search areas and decreasing whenever a new best solution is found. The tenure also depends on the cardinality of the problem, i.e. number of arcs in the tree, being smaller for small and large cardinalities, and being larger for intermediate cardinalities since they are usually more complex. Neighbour solutions are generated with the use of swap moves. Whenever a first neighbour solution improving the current best solution is found, the search of the neighbourhood is stopped. However, if no improving solution is found the current solution is deleted and a new initial solu-

tion is generated (in a greedy manner) and the tabu lists are initialized. The evolutionary approach uses a population size dependent on the cardinality of the problem and on the number of arcs, that varies between 50 and 200 individuals. Small cardinality problems are allowed larger populations so that crossover may take place. The initial population is a set of randomly generated k-cardinality trees. For every k-cardinality tree one other tree, from the set of trees having at least one arc in common, is chosen by means of roulette-wheel-selection. These two trees act as parents in the crossover phase. Two crossover operators are applied to the parents: (i) union-crossover, that creates an offspring from the low weight arcs appearing in only one of the two parents, and (ii) intersection-crossover, that incorporates into the offspring low weighted arcs that both parents have in common and completes the offspring with the best arcs from the rest. Then, the best tree found among the two offspring and the first parent is chosen to incorporate the next generation. Local search is applied to every individual produced by the crossover procedure, by swapping one arc at a time. Afterwards, a run of the tabu search, described above, is applied to the best solution of the current population. The concept of ageing is introduced into the evolutionary algorithm in order to remove solutions that have passed on from generation to generation a certain number of times. After the tabu search run is applied, the age of each individual of the current population is incremented by one, and a sort of tabu list is constructed containing individuals whose age exceeds 10 generations. These individuals are removed from the population and new ones are randomly generated in order to substitute them. Finally, a max-min ACO algorithm is also developed. In this approach, the colony size is restricted to be between 15 and 50 ants. In order to build a solution, each ant starts by randomly selecting an arc to add to the solution tree. Then, $k-1$ arcs are added to the tree, one at a time, based on the pheromone values associated with the arcs and according to the transition rule defined in Eq. (3.53) (see Section 3.3.4.5). The same local search procedure used in the two previous approaches is applied to every tree constructed by the ants. Afterwards, the tabu search procedure is run in order to further improve the best solution of the iteration. Pheromones are updated as given in Eq. (3.46) (see Section 3.3.4.4). A convergence factor, based on the probability of generating all over again the best solution of the iteration, is used to reset the pheromone matrix and try to explore other search areas. The three algorithms were applied to a set of benchmark problems, available at (Beasley, 2012), proving the competitiveness of these algorithms when compared to the results in the literature. The authors have also generated a new set of benchmark arc-weighted k-cardinality tree problems, containing grid graphs, regular graphs, dense graphs and sparse graphs, having concluded that the performance of each algorithm is determined by the class of graphs being solved.

Another recent work on the k-MST problem is the one of Katagiri et al (2012) where

the authors propose a method by hybridizing tabu search and ant colony optimization. The TS procedure is used to perform an intensive search nearby good solutions, whereas the ACO procedure, an extension of the one developed by Blum and Blesa (2005), is used in order to expand the search towards other areas of the search space, that is to say in order to diversify the solutions. Prim's algorithm is used to find an initial feasible solution, i.e. a subtree with exactly k arcs, for the problem. The TS procedure performs local search in the neighbourhood $N(S)$ of a good solution S, where $N(S)$ is defined as the set of all solutions S' differing from S in only one node i. Therefore, in order to obtain a neighbour solution S', a node $i \notin S$ must be chosen to be added to solution S and a node $s \in S$ must be chosen to be dropped from the solution. The authors define the set of all nodes $i \notin S$ as the set of candidate nodes to enter the solution. For each node i in the candidate set the following procedure is defined. Initially, the average cost per arc, regarding all existing arcs linking node i to S, is computed and both the first and the second arcs with the lowest ratio are added to S, thus introducing a cycle into the solution. Then, the arc with the largest cost in the cycle is identified and is deleted from S. The node to be deleted from S is identified with an analogous process but now the algorithm chooses the node maximizing the average weight of the arcs outgoing from it. Then, the node and all arcs connecting to it are removed. This deletion may generate a tree with let us say P disconnected tree parts. These parts are joined by adding the $P - 1$ lowest cost arcs connecting the parts. These steps are attempted until no further improvement is achieved, thus retrieving a neighbour solution S'. The arcs that can be added to or deleted from a solutions S are controlled by two tabu lists, called inList and outList, keeping track of the removed and added arcs, respectively, and by a tenure parameter θ defining the period of time during which the arcs are to remain in the tabu lists. Whenever the TS finds a local optimum, pheromone is deposited in its arcs. Latter on, in the diversification phase of the hybrid algorithm, the ACO procedure is used in order to expand the search area of the solution. These two phases are performed until the maximum running time allowed is reached. Numerical experiments were performed in benchmark problems instances and were compared with the ones obtained with the algorithms developed in (Blum and Blesa, 2005). This new approach was able to improve upon some of the previous best known results.

Works considering HMST problems with flows other than the unit are scarce. We end this review with the only two works, that are known to the moment, on the HFMST problem: one focused on an exact method, dynamic programming, and the other developing an evolutionary algorithm. Fontes (2010) uses dynamic programming to solve an extension of the MST problem with hop-constraints which also considers flows. The dynamic programming formulation is recursive which means that with the use of a backward-

forward procedure the state space graph is successively expanded. Stages are defined by the number of demand nodes under consideration in set S. States are defined by three state variables, the set of demand nodes S to be considered, the node x acting as a source node to demand nodes in S, and the hop parameter value h, thus a state is represented as (S, x, h). The algorithm starts from the last stage and works backwards, through not yet computed states, until a computed state is reached. Then, after computing the state value the algorithm moves forward, through already computed states, until it reaches a state not yet computed. The process is repeated until the final state is reached and no better solution can be found. Each time the algorithm moves from one stage to another stage it includes a node into the tree. Thus, whenever the algorithm is at any state of stage i, i nodes are already included in the solution. The problem instances solved have 10, 12, 15, 17 and 19 nodes, hop parameter ranging from 3 up to 10, and three distinct nonlinear cost functions with discontinuities, depending on a percentage of the total demand, other than at the origin. In total, the author solves 4050 problem instances to optimality being able to demonstrate that the computational performance is independent of cost function type.

The same problem is approached in (Fontes and Gonçalves, 2012) where a Multi-Population hybrid biased random key Genetic Algorithm (MPGA), with three populations evolving separately, is used. These populations, which are randomly generated, are let to evolve independently and then, after every 15 generations the two best chromosomes are included in all the other populations. The encoding of the solution tree is made recurring to random keys. Therefore, a chromosome is made of $3n$ genes, where n represents the number of nodes in the tree. The first $2n$ genes of the chromosome are used by a decoder procedure called Tree-Constructor to construct a solution tree. (The decoder uses the first n genes to provide an order in which the nodes are considered to enter the tree, and the other n nodes are used to select an antecessor for the corresponding node). The remaining n genes are used later by a local search procedure. Each gene assigns a random priority to the node it represents. To construct the solution tree, the algorithm starts with the highest priority node not yet supplied and searches within the set of the remaining nodes, by priority order, a feasible supplier to it, i.e. a node not creating a cycle. The hop-constraint is handled a posteriori, by penalizing infeasible solutions with more than h arcs in the path from the source node. The penalty of a solution is proportional to the number of extra hops in each violating path. Local search is performed by replacing a node in the tree with another node not in the tree that is still an extreme flow. The order by which the nodes are considered for the improvement of the solution is given by the last n genes of the chromosome. In order to evolve from a generation to another, a set consisting of the 25% top best solutions are directly copied onto the next generation. Then, a biased uniform crossover is performed, between a parent chosen from the top best solutions

and a parent chosen from the set of all solutions, creating a new chromosome where each gene has a higher (biased) probability to be taken from the best parent. Finally, the mutation operator used, which is called the immigration operator, is not used in the usual sense, because the chromosomes are not mutated but instead are randomly created as in the first population. The newly generated chromosomes substitute the bottom (worst) 15% chromosomes of the next generation. The results were obtained for 150 problem instances, for each value of $h = 3, 5, 7, 10$, and were compared with the exact solutions obtained by the aforementioned DP procedure, proving that the heuristic is very efficient and effective.

In the following section we will present and explain, step by step, the ant colony based algorithm that was developed to solve the HMFST problem.

5.4. Solving Hop-Constrained Minimum Cost Flow Spanning Tree Problems

In this section, we address the hop-constrained minimum cost flow spanning tree problem with nonlinear costs. We propose a hybrid heuristic to solve it, since this problem is of a combinatorial nature and also the total costs are nonlinear. Hybrid algorithms usually try to manage two conflicting aspects of searching behaviour: exploration, the algorithm's ability to search broadly through the search space; and exploitation, the algorithm's ability to search locally around good solutions that have been found previously. In our case, we use an ant colony optimization algorithm to mainly deal with the exploration, and a local search algorithm to cope with the exploitation of the search space. It is known that an ACO algorithm incorporates several mechanisms, for example the evaporation rate ρ, that can be adjusted at any time, in order to encourage the colony of ants to perform a thorough search in the neighbourhood. But, these mechanisms are not always sufficient due to the *"hardness"* of the problems. As far as we are aware of, no other work using ACO algorithms neither to solve hop-constrained MST nor HMFST, has been reported in the literature.

In the following sections we review the ant colony optimization methodology and the hybrid ACO that was developed to solve the HMFST problem with nonlinear costs.

ACO algorithms are characterized by a set of decisions that have to be made regarding the construction of the solution and some of the used parameters. And, as was already mentioned before, the first and most important decision to be made while developing an ACO algorithm is the representation of the solution to the problem being solved, since a poor representation can lead to not so good solutions and high computational times.

5.4.1. Representation of the Solution

The solution for the HMFST problem is an arborescence, i.e. a connected graph without cycles. Therefore, between the central node, also known as root or source node, and every demand node, there can only exist a single directed path. The solution constructed by each ant must then consist on as many arcs as the number of demand nodes and include all demand nodes. Furthermore, a solution is considered feasible if the path from the source node t to each demand node has at most H arcs.

5.4.2. Solution Construction

In the method that is used to construct solutions for this problem all ants begin their solution construction at the source node. Initially, an ant selects an existing arc linking the source node t and one of the demand nodes $j \in N \setminus \{t\}$. Then, the ant selects another arc, from the set of available arcs linking the source or one of the demand nodes already in the partial solution to another demand node not yet in the solution such that no more than H arcs are included in any path from the root node, and adds it to the solution tree. For example, consider a fully connected network with four demand nodes $\{a, b, c, d\}$ and a source node t. Let us suppose that the first arc added to the solution is arc (t, a). The next step to be taken is to choose from the remaining demand nodes $\{b, c, d\}$ the one entering the solution tree provided that it is either linked to the source node t or to the demand node a, already in the solution. The possibilities are thus $\{(t, b), (t, c), (t, d), (a, b), (a, c), (a, d)\}$. One of these arcs is chosen not at random but rather based on the pheromone quantity present in it. These two steps are repeatedly performed until there remains no demand node outside the solution tree.

The steps described above, do not guarantee the feasibility of the solution regarding the hop-constraints. To overcome this problem each time an arc (k, j) is added to the solution tree the length l_j of the path linking the source node t to the demand node j is computed. If $l_j = H$, the maximum length (number of hops) allowed is reached in that path and the arcs considering node j as a parent are excluded from the set of viable arcs. New arcs are added until all the demand nodes are in the solution, or until no viable arcs are available. If an ant is able to construct a solution with n arcs then the solution is feasible. Otherwise, the solution is discarded. In this case, we could have chosen to fix unfeasible solutions or allow the ant to look for another solution or even allow unfeasible solutions to be used. However, given that the number of discarded solutions was not significant we decided to disregard unfeasible solutions.

As said before, an arc entering the solution is not chosen completely at random. In-

stead, it is chosen by using a probability function incorporating information about the visibility of arcs and on the pheromone quantity associated to them, as defined bellow:

$$P_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{(i,j)\in A} [\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}, \tag{5.49}$$

where τ_{ij} is the pheromone present in arc (i, j) at the current iteration; η_{ij} is the visibility of arc (i, j) and is usually represented as the inverse of the cost of the arc; $\alpha, \beta > 0$ are parameters weighting the relative importance of the pheromone value and of the visibility information, respectively. The visibility of an arc is also known as *heuristic information* and is only calculated once at the beginning of the algorithm, since it depends only on the problem data. If we make an analogy between cost and distance, the visibility of an arc (i, j) is higher if node j is "close" to node i and thus we can *see* it while standing at node i, and is lower if it is difficult to *see* node j from node i.

5.4.3. Updating and Bounding Pheromone Values

After all ants have constructed their solutions, the algorithm steps into the pheromone updating phase. In this phase the best solution of the current iteration S^i is identified and the algorithm updates the pheromones. We only allow the ant that has constructed S^i to reinforce the pheromone quantity in the arcs of its solution. The update of pheromones initially simulates the natural process of evaporation by reducing the pheromone values in every existing arc $(i, j) \in A$. This is represented by the first component of Eq. (5.50), where $\rho \in]0, 1]$ represents the pheromone evaporation rate and τ_{ij} the pheromone quantity in arc (i, j). If the evaporation rate parameter ρ takes a value near to 1, then the pheromone trail will not have a lasting influence throughout the following iterations, whereas a small value will increase the importance of the arcs a lot longer.

$$\tau_{ij} = (1 - \rho) \times \tau_{ij} + \Delta\tau_{ij}. \tag{5.50}$$

The second component, $\Delta\tau_{ij}$, represents the pheromone quantity to be deposited in arc (i, j), and is given by:

$$\Delta\tau_{ij} = \begin{cases} \dfrac{Q}{F(S^i)} & \text{if } (i,j) \text{ belongs to solution } S^i, \\ 0 & \text{otherwise,} \end{cases} \tag{5.51}$$

where Q is a positive proportionality parameter and $F(S^i)$ is the cost of the best solution

found at the current iteration.

In the initialization phase of the algorithm, an equal amount of pheromone τ_0 is deposited in every arc of the problem, thus guaranteeing that every arc has the same chance of being chosen, as stated in Eq. (5.52).

$$\tau_{ij} = \tau_0, \ \ \forall (i,j) \in A. \tag{5.52}$$

At each iteration, after the pheromone update is performed a check is done to find out if its value is within the interval $[\tau_{min}, \tau_{max}]$, following the work of (Stützle and Hoos, 1997). The τ_{max} value depends on the cost of the best solution found so far F^* and on the pheromone evaporation rate ρ, and the τ_{min} value depends on the upper bound for the pheromone value τ_{max} and on a parameter p_{best}, the probability of constructing the best solution, as given in Eq. (5.53).

$$[\tau_{min}, \tau_{max}] = \left[\frac{\tau_{max} \cdot (1 - \sqrt[n]{p_{best}})}{(\frac{n}{2} - 1) \cdot \sqrt[n]{p_{best}}}, \frac{1}{\rho \cdot F^*} \right]. \tag{5.53}$$

Since both τ_{min} and τ_{max} only depend on the cost of the best solution found so far, they merely have to be updated each time the best solution is improved. When checking if the pheromone values are within the limits defined, the following corrective actions are taken: if on the one hand, some pheromone value is bellow τ_{min}, it is set to τ_{min}; on the other hand, if some pheromone value is above τ_{max}, it is set to τ_{max}. By limiting the pheromone value within these bounds, the choice of the initial pheromone value τ_0 becomes less important as pheromones converge within a few iterations to reasonable values within the desired interval.

Although setting pheromone bounds is a good way to prevent getting trapped into local optima, the algorithm needs an extra mechanism to deal with stagnation and cycling of solutions. To solve this problem we keep track of the number of iterations an incumbent solution has not been changed. Whenever that number reaches 200 iterations, the pheromone values are reinitialized, that is, the pheromone values are set to τ_0 for all arcs. This gives the algorithm a chance to search for a better solution in another region of the search space, before the fixed number of allowed iterations is reached.

The pheromone updating scheme used here is the same as the one in the previous chapter. This fact is not surprising since pheromone update, i.e. the rewards given to good solutions and penalties given to all the other not so good solutions, is one of the main characteristics of ACO that allows it to be differentiated from other population based heuristics. Another reason is that the min-max ant algorithm has been found to be one of

the best ant colony optimization algorithms developed. Therefore, many ACO heuristics developed, to solve all kinds of optimization problems, share this common structure.

5.4.4. The Algorithm

Now that we have described every specific characteristic of the algorithm let us present the flowchart for the HACO2 heuristic, which is given in Fig. 5.9, along with a brief description.

The algorithm starts by depositing an initial pheromone quantity τ_0, on every arc and by initializing all necessary parameters. Then, every ant constructs its solution following the procedure explained in Section 5.4.2.

Solutions are sorted in ascending order of cost and the best solution of the iteration S^i is afterwards identified. A set W is created with the best five solutions found at the current iteration. Local search is performed on all solutions $S \in W$ and a set W' with the improved solutions, if any exist, is returned. S^i is updated by identifying the best solution between the previous S^i and the solutions returned in W'.

If the cost of S^i is lower than the cost of the best solution found so far S^g, the incumbent solution S^g and the best cost $F(S^g)$ are updated.

The next step is to update the pheromone values. Firstly, pheromones are evaporated in every single arc. Then, a pheromone value proportional to the cost of the best solution found at the current iteration is added to all the arcs in the solution. Afterwards, the algorithm tests all pheromone values for violations of the pheromone bounds. If the pheromone accumulated on an arc exceeds τ_{max} then its value is set to τ_{max}, and if it falls below τ_{min}, then its value is set to τ_{min}.

Continuing with the flowchart, if any of the stopping criteria is satisfied, then the algorithm stops and returns the best solution found, as well as its cost, as the final result; otherwise it continues to the next iteration. Two stopping conditions are used. The algorithm stops running if a pre-defined maximum number of iterations is reached, or it stops if three consecutive restarts, of the pheromone matrix, have been performed. Recall that the pheromone matrix is reinitialized every time that a number of iterations (200 in our case) has been performed without improving the best solution.

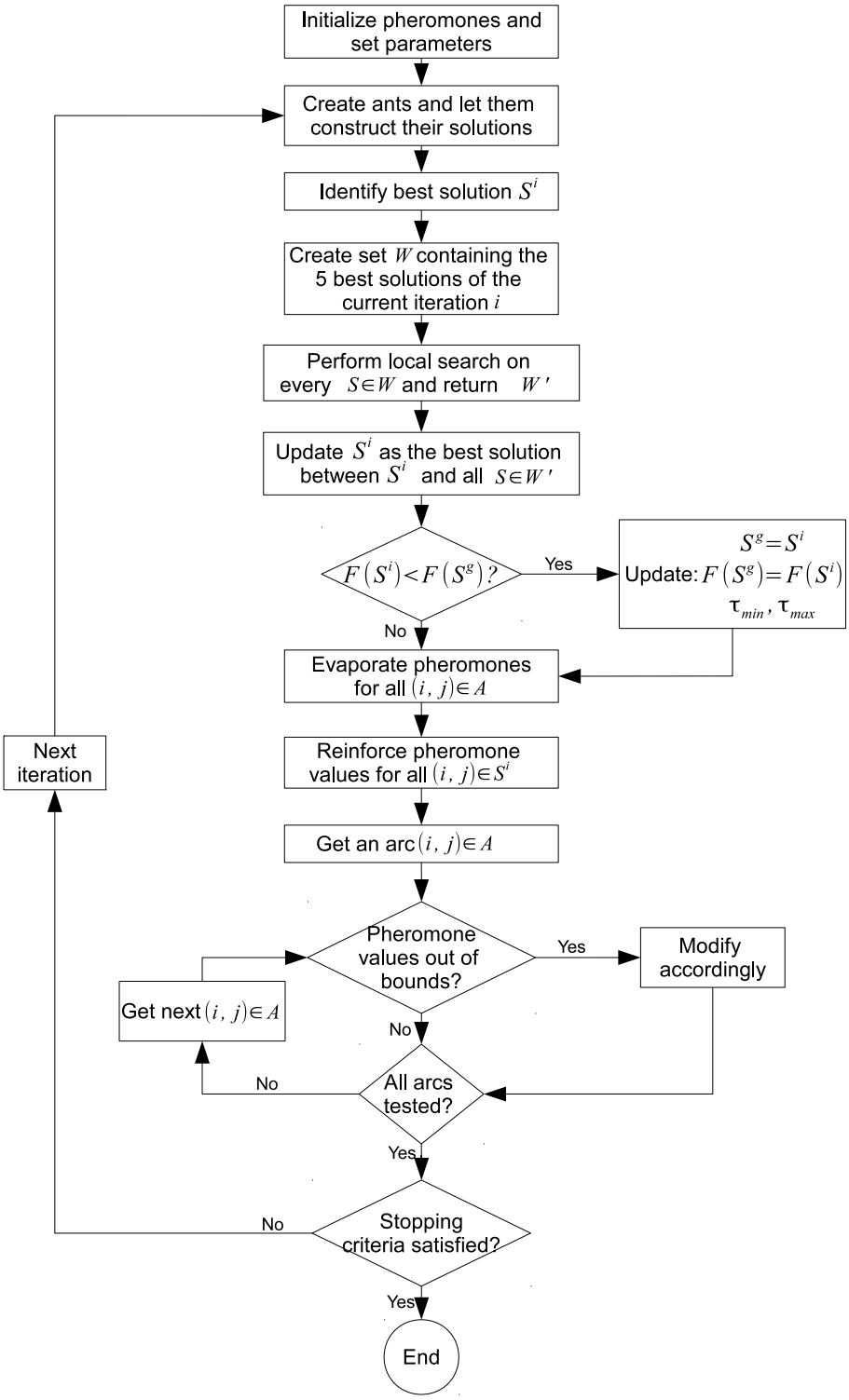

Fig. 5.9: Flowchart for the proposed ACO algorithm (HACO2).

5.4.5. Local Search

A local search procedure was developed and, as said before, is applied right after the ants have constructed a feasible solution and identified S^i. The feasible solutions provided by

the ants are sorted in ascending order of total costs, that is to say, sorted from the best to the worst. Then, we create a set W, containing the first five solutions, to have local search performed on.

The local search, when performed in the vicinity of a particular solution S, allows the algorithm to search for a neighbour solution that might have a lower cost. For this problem, a solution S^N is a neighbour solution of solution S if S^N is obtained from S by swapping an arc $(i, j) \in S$ with another arc $(l, j) \notin S$ that does not create a cycle and still ensures no node is more than H arcs away from the source node. Therefore, after the swap takes place node j gets its demand and the demand of all nodes it supplies, if any, from node l instead of from node i.

The local search algorithm, that is applied to the aforementioned five solutions, is given in Fig. 5.10.

The arcs in the selected solution S are sorted in ascending order of the value of their pheromone, in order to try to substitute in the first place the "worst" arcs. For each of them we try to find an alternative arc that improves the cost of the solution. In order to do so, we find all the arcs (k, j) that can replace the current one while maintaining feasibility of the solution, i.e. not forming a cycle and maintaining the constraint H for all paths. We attempt to replace the original arc (i, j), by starting with the ones with a higher pheromone value. If one of the replacements improves the cost of the solution we proceed to the next arc (i, j) in the solution without attempting the remaining options. The new solution to be used in the remaining of the algorithm is either S or S', whichever has the lowest cost.

Let us consider a small example to clarify how the local search procedure is applied. Consider a problem defined on a fully connected graph with six demand nodes $\{a, b, c, d, e, f\}$, a root node t, and where $H = 2$. We make use of Fig. 5.11 in order to illustrate the evolution of a solution S with the application of local search. Assume that we have already identified the set of five solutions selected to perform local search on and that we are inspecting the neighbourhood of one particular solution S, where $S = \{(f, c), (t, f), (f, e), (a, b), (t, d), (t, a)\}$, see Fig. 5.11 (a). In order to simplify, assume also that the arcs in S are already sorted in ascending order of arc pheromone, i.e. $\tau_{fc} \leq \tau_{tf} \leq \tau_{fe} \leq \tau_{ab} \leq \tau_{td} \leq \tau_{ta}$.

We try to improve solution S by replacing each of the six arcs in S, one at a time, with better arcs, i.e. with arcs that decrease the total cost of S.

First, we remove arc (f, c) from S, and then we identify the set P of candidate arcs $(l, c) \notin S$ to substitute arc (f, c), since this is the arc with the smallest pheromone value.

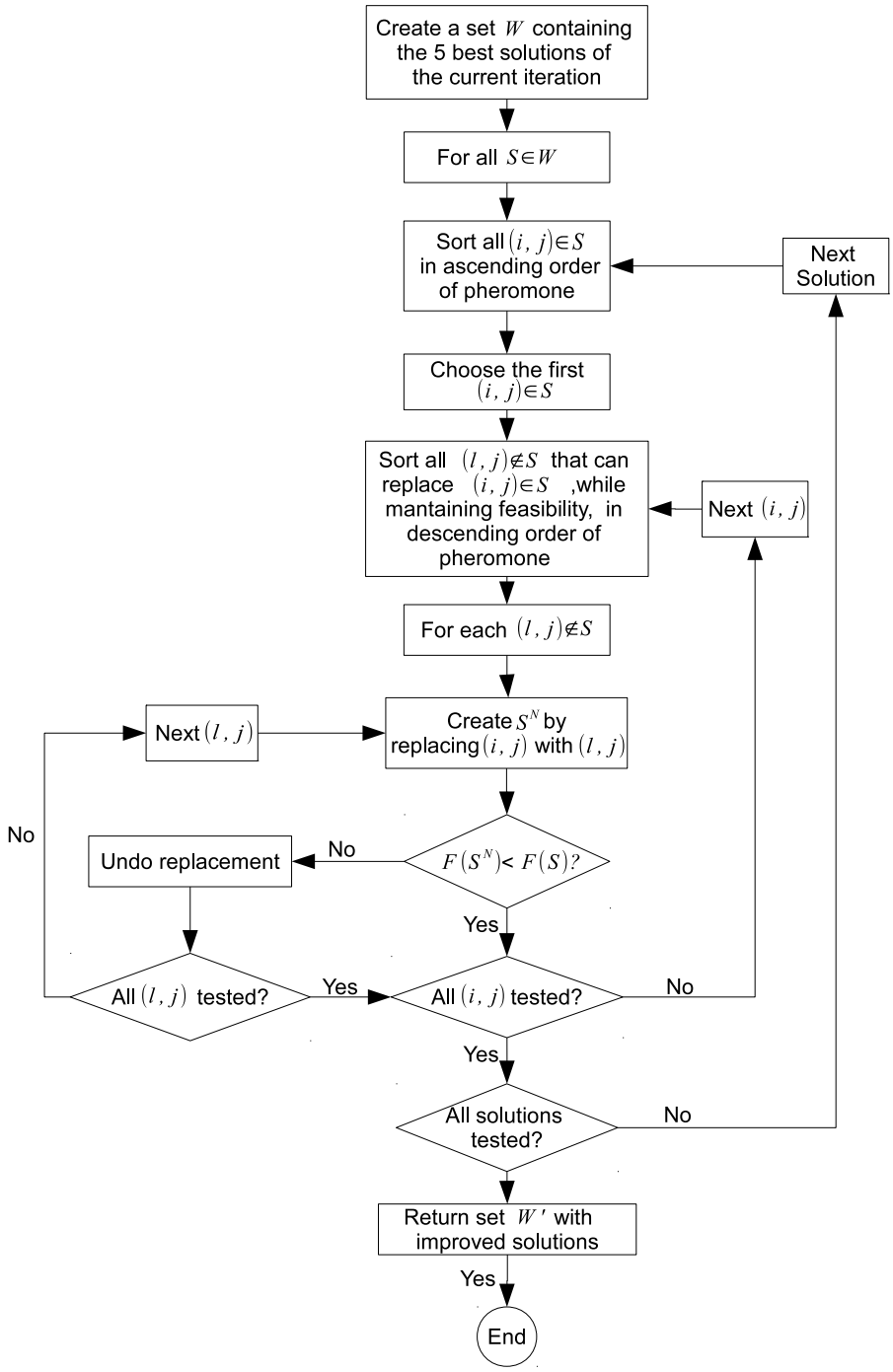

Fig. 5.10: Pseudo-code of the Local Search procedure that was incorporated into the ACO algorithm developed.

The candidate arcs set is given by $P = \{(d, c), (t, c), (a, c)\}$. Suppose that P is already sorted in descending order of arc pheromone, i.e. $\tau_{dc} \geq \tau_{tc} \geq \tau_{ac}$.

In Fig. 5.11 (b) all fine dashed lines represent arcs that can be used to reconnect node c to the solution tree S, whereas all dashed lines represent arcs that cannot be considered as candidate arcs because they would lead to unfeasible solutions. Note that arcs (b, c)

and (e, c), in dashed lines, cannot be in the set of candidate arcs P because they would violate the hop-constraint $H = 2$.

Following the algorithm, we replace arc (f, c) with arc (d, c) thus obtaining a neighbour solution $S^N = \{(d, c), (t, f), (f, e), (a, b), (t, d), (t, a)\}$. Next, we calculate $F(S^N)$ and compare it with $F(S)$. Let us assume that $F(S^N) < F(S)$. Then, we accept this arc swap and continue with the local search procedure considering this new solution S^N, by making $S \leftarrow S^N$, see Fig. 5.11 (c). After this swap takes place node c gets its demand from node d instead of from node f.

The local search will now try to replace the next arc in line in S, which is arc (t, f). It is important to notice though that we never go backwards while trying to improve a solution, which means that once a swap has been made, the procedure will not try to improve the swaps that have already been performed. Let us go back to our example.

For arc (t, f) the new $P = \emptyset$, because none of the arcs $(l, f) \notin S$ can provide a feasible solution (they would all introduce either a cycle or violate the hop-constraint), see Fig. 5.11 (d). Therefore, arc (t, f) is kept in S, and the procedure continues the search with the next arc, arc (f, e), which will render $P = \{(a, e), (d, e), (t, e)\}$, as can be seen in Fig. 5.11 (e).

The local search procedure continues the search for better solutions until all arcs in the original solution S have been tested. This is repeated for the remaining solutions in W, until all five solutions have been inspected in order to be improved.

5.4.6. Sets of Tested Parameters

There are a few decisions regarding the values to be taken by the parameters described in the previous sections. The development of our algorithm was achieved in several phases and we had to set some values for our first experiences. In an initial phase, we tested the parameter values given in the second column of Table 5.1. The ones with the best results are summarized in the third column. After developing the last phase of the HACO2 algorithm, we tested the parameter values once more. The results indicated that the ones chosen in the first tests were still the ones achieving the best results. Some comments must be made regarding the choice of the parameters.

The observation of Eq. (5.49) allows for the conclusion that α and β are related and that the choice of their values must be made carefully. Therefore, the tests we have made in order to choose them, consider all combinations between the values of these parameters, and the best combination was the one used thereafter. As for the evaporation rate,

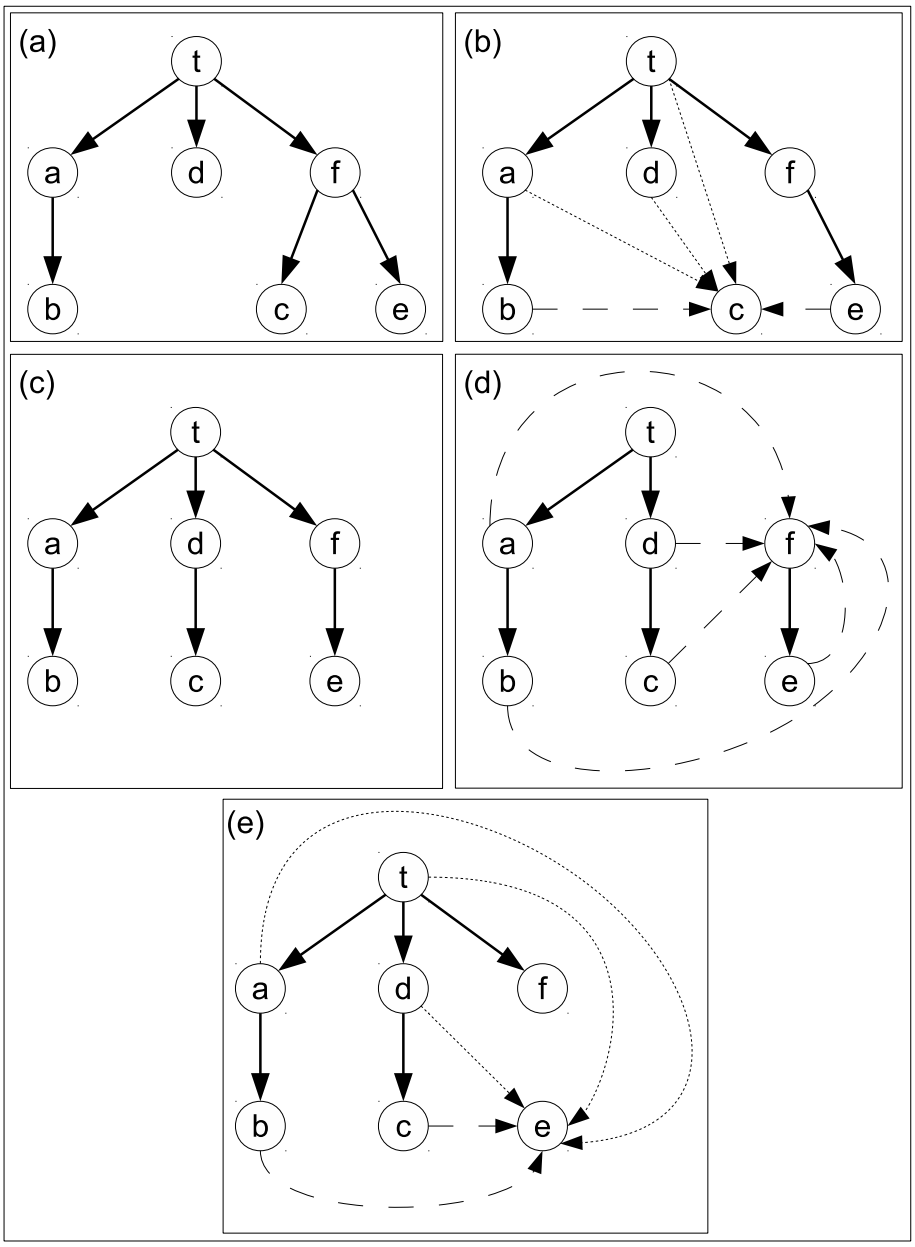

Fig. 5.11: Graphical representation of the example given for the local search procedure.

Tab. 5.1: Parameter values for the developed procedure.

Parameter	Tested Values	Final Values
α	1, 3, 5	1
β	1, 2, 3, 5	2
ρ	0.05, 0.1, 0.2, 0.5	0.1
Q	1, 2, 5	2
p_{best}	0.5, 0.05, 0.01	0.5
τ_0	1, 1000000	1000000
η_{ij}	$1/c_{ij}$, $1/b_{ij}$, $1/(c_{ij}+b_{ij})$	$1/(c_{ij}+b_{ij})$
no. of ants	n, 2n	2n
no. of iterations	500, 1000, 2000	2000

parameter ρ, it was found that 10% was the best value, which points towards privileging the exploitation of the search space nearby the best solution of each iteration.

Although we have opted to leave out of this chapter the results obtained with the variation on the ACO parameters, presenting only the final values that were used, a study similar to the one presented in Section 4.6.1 of Chapter 4, has also been performed for the HMFST problem.

5.5. Computational Results

In this section, we present the computational results that were obtained with the HACO2 algorithm that was developed along with results obtained with the commercial software CPLEX and also some literature results for the same problems, in order to compare the efficiency end effectiveness of our algorithm. The analysis of the robustness, the ability of the heuristic to reproduce the same solution or a very similar one in different runs, is approximately achieved by solving 10 times each problem instance and then by computing the minimum, maximum, average and standard-deviation of the solutions obtained. If the first three statistics are around the same values and the standard deviation is small, then the method may be considered robust.

The algorithm described in this paper was implemented in Java and the computational experiments were carried out on a PC with a Pentium D at 3.20GHz and 1GB of RAM. The CPLEX was run on the same PC.

To evaluate an heuristic, we characterise its solutions regarding:

1. Time, in seconds, required to perform a full run of the algorithm;

2. Optimality gap in %, which is given by:

$$\text{Gap}(\%) = \frac{HS - OptS}{OptS} \times 100,$$

where OptS stands for the optimum solution, obtained by CPLEX for cost functions F1, F2 and F3 when available, and the best known otherwise; and the HS stands for the best solution found with the heuristic in question. The HMFST problem was solved with CPLEX for F1, F2, and F3 cost functions. CPLEX does not solve problems with functions of type F4. Nonetheless, (Fontes, 2010) provided times and optimal solutions obtained with a dynamic programming algorithm for cost functions of type F4 and problems with up to 19 nodes. For larger problems we have used the results obtained by Fontes and Gonçalves (2012), available in www.fep.up.pt/docentes/fontes (under project PTDC/EGE-GES/099741/2008), by calculating the gap between the costs obtained by our HACO2 and the very best ones obtained with the MPGA, since no optimal results are available to the moment.

In order to infer about the stability of our method, we present a set of statistics regarding the gap. Tables 5.2 to 5.5 summarize the gap results obtained for each cost function herein considered. Note that Table 5.5 refers to problems with cost function F4 only with up to 19 nodes. Each table presents Minimum (Min), Average (Avg), 3rd Quarter (3Q), Maximum (Max), and Standard Deviation (SD) values obtained with the HACO2 algorithm, grouped by hop value, as well as Minimum (Min), Average (Avg), and Maximum (Max) gap results obtained with the aforementioned MPGA.

Before going any further please note that as cost function F1 was not considered in (Fontes and Gonçalves, 2012), Table 5.2 does not include results for the MPGA algorithm. Furthermore, by setting the hop value to 3 problems with 40 and with 50 nodes do not have any feasible solution. Recall that the problem instances do not consider a complete network.

Let us begin by analysing the gap. First of all, we observe that the minimum gap value obtained, regardless of cost function and hop value considered, is always zero. This result is very important because it means that in the 10 runs of the algorithm we are always able to find, at least once, the optimum solution. Furthermore, the value for the third quarter of the gap distribution is also zero, meaning that at least 75% of the solutions found by the HACO2 are optimal.

When we analyse the results from the hop value point of view, it is curious to notice that, although it is known that the difficulty in solving these problems increases with the

Tab. 5.2: HACO2 optimality gap (%) for F1 cost function with $H = 3, 5, 7$, and 10

	H=3					H=5				
N	Min	Avg	3Q	Max	Sd	Min	Avg	3Q	Max	Sd
10	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0.002	0	0.08	0.010
40	-	-	-	-	-	0	0	0	0	0
50	-	-	-	-	-	0	0.010	0	0.49	0.069
	H=7					H=10				
N	Min	Avg	3Q	Max	Sd	Min	Avg	3Q	Max	Sd
10	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0
40	0	0.001	0	0.02	0.003	0	0	0	0	0
50	0	0.002	0	0.05	0.010	0	0	0	0	0

decrease on the hop parameter value H, the HACO2 heuristic has a very good performance for $H = 3$, regardless of the cost function, and finds the optimum value for all runs of the algorithm. The same can not be said about the performance of the MPGA which presents the worst gap values precisely for $H = 3$, in particular, the largest gap value, 17%, was obtained for a problem with 19 nodes and considering cost function F2. The HACO2 and MPGA performance achieved for $H = 10$ is also of zero gap for cost functions F2, F3 and F4.

Problems with $H = 5$ and $H = 7$ proved to be the most difficult ones to solve by HACO2, and the largest gap value obtained is 0.51%, considering F2 and $H = 5$. Gaps larger than zero tend to happen with problems with at least 30 nodes, although one 17 node problem presented a nonzero gap.

Looking now at the results obtained by type of cost function, there seems to be no influence on the performance of our algorithm as the gap values are not that different. In general, whenever MPGA presents a positive average gap, HACO2 has improved it

Tab. 5.3: HACO2 and MPGA optimality gap (%) for F2 cost function with $H = 3, 5, 7$, and 10

		HACO2					MPGA		
	N	Min	Avg	3Q	Max	Sd	Min	Avg	Max
H=3	10	0	0	0	0	0	0	0.360	7.157
	12	0	0	0	0	0	0	0.104	1.900
	15	0	0	0	0	0	0	0.031	0.641
	17	0	0	0	0	0	0	0.120	3.743
	19	0	0	0	0	0	0	0.182	17.353
	25	0	0	0	0	0	0	0.283	6.141
	30	0	0	0	0	0	0	0	0
	40	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-
H=5	10	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0
	17	0	0.008	0	0.46	0.059	0	0	0
	19	0	0	0	0	0	0	0.062	4.919
	25	0	0	0	0	0	0	0.005	0.162
	30	0	0.002	0	0.08	0.013	0	0.109	4.083
	40	0	0	0	0	0	0	0.139	7.501
	50	0	0.005	0	0.51	0.051	0	0.076	0.991
H=7	10	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0.005	0.258
	17	0	0	0	0	0	0	0	0
	19	0	0	0	0	0	0	0	0
	25	0	0	0	0	0	0	0	0
	30	0	0	0	0	0	0	0	0
	40	0	0.0004	0	0.02	0.003	0	0.074	1.945
	50	0	0.001	0	0.05	0.007	0	0.047	1.426
H=10	10	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0
	17	0	0	0	0	0	0	0	0
	19	0	0	0	0	0	0	0	0
	25	0	0	0	0	0	0	0	0
	30	0	0	0	0	0	0	0	0
	40	0	0	0	0	0	0	0	0
	50	0	0	0	0	0	0	0	0

Tab. 5.4: HACO2 and MPGA optimality gap (%) for F3 cost functions with $H = 3, 5, 7$, and 10

		HACO2					MPGA		
	N	Min	Avg	3Q	Max	Sd	Min	Avg	Max
H=3	10	0	0	0	0	0	0	0.389	7.162
	12	0	0	0	0	0	0	0.110	1.914
	15	0	0	0	0	0	0	0.075	2.396
	17	0	0	0	0	0	0	0.120	3.743
	19	0	0	0	0	0	0	0.041	1.787
	25	0	0	0	0	0	0	0.251	6.993
	30	0	0	0	0	0	0	0	0
	40	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-
H=5	10	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0
	17	0	0.005	0	0.46	0.046	0	0	0
	19	0	0	0	0	0	0	0.095	4.922
	25	0	0	0	0	0	0	0.006	0.177
	30	0	0.002	0	0.08	0.010	0	0.071	4.083
	40	0	0	0	0	0	0	0.425	13.909
	50	0	0	0	0	0	0	0.084	0.992
H=7	10	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0.006	0.228
	17	0	0	0	0	0	0	0	0
	19	0	0	0	0	0	0	0	0
	25	0	0	0	0	0	0	0	0
	30	0	0	0	0	0	0	0	0
	40	0	0.001	0	0.02	0.004	0	0.048	1.945
	50	0	0.004	0	0.05	0.011	0	0.004	0.243
H=10	10	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0
	17	0	0	0	0	0	0	0	0
	19	0	0	0	0	0	0	0	0
	25	0	0	0	0	0	0	0	0
	30	0	0	0	0	0	0	0	0
	40	0	0	0	0	0	0	0	0
	50	0	0	0	0	0	0	0	0

Tab. 5.5: HACO2 and MPGA optimality gap (%) for F4 cost functions with $H = 3, 5, 7$, and 10

		HACO2					MPGA		
	N	Min	Avg	3Q	Max	Sd	Min	Avg	Max
H=3	10	0	0	0	0	0	0	0.251	7.160
	12	0	0	0	0	0	0	0.157	2.453
	15	0	0	0	0	0	0	0.038	2.352
	17	0	0	0	0	0	0	0.093	3.858
	19	0	0	0	0	0	0	0.122	9.108
H=5	10	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0
	17	0	0.01	0	0.48	0.067	0	0	0
	19	0	0	0	0	0	0	0.042	1.615
H=7	10	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0.003	0.131
	17	0	0	0	0	0	0	0	-
	19	0	0	0	0	0	0	0	-
H=10	10	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0
	17	0	0	0	0	0	0	0	0
	19	0	0	0	0	0	0	0	0

except for the particular case, already mentioned, of the problem with 17 nodes.

In Table 5.6, we report on the optimality gap results for the HACO2 and for the MPGA for problems with 25 up to 50 nodes and cost function F4. These problems have only been solved with these two heuristics therefore, the optimality gap is calculated by comparing the results obtained by each of the heuristics with the currently best solutions (lowest cost solution found by HACO2 or by MPGA). As it can be seen in Table 5.6, the tendency for finding the optimum value that has been observed for the former three functions is confirmed for HACO2 when $H = 3$, and for HACO2 and MPGA when $H = 10$. Nonetheless, there are two important advantages regarding the HACO2 algorithm. Firstly, HACO2 is always able to find a feasible solution whereas the MPGA is not. The MPGA was not able to find a feasible solution for 19 problem instances, see the results reported in (Fontes and Gonçalves, 2012). Secondly, the stability of HACO2 is confirmed by observing the maximum gap values obtained. Although each problem instance was solved 10 times by the HACO2 algorithm and only 5 times by the MPGA, the maximum gap values observed for

HACO2 are much lower than the ones observed for MPGA, except for the aforementioned problem with 17 nodes. Nevertheless, even in such case the maximum gap is below 0.5%.

Tab. 5.6: Optimality gap (%) obtained by HACO2 and MPGA for F4 cost functions with $H = 3, 5, 7$, and 10 when compared with the currently best known solutions (obtained either by HACO2 or by MPGA)

		HACO2					MPGA		
	N	Min	Avg	3Q	Max	Sd	Min	Avg	Max
H=3	25	0	0	0	0	0	0	0.287	6.961
	30	0	0	0	0	0	0	0.113	2.664
	40	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-
H=5	25	0	0	0	0	0	0	0.004	0.148
	30	0	0.012	0	0.329	0.055	0	0.152	4.087
	40	0	0	0	0	0	0	0.118	7.646
	50	0	0.023	0	0.579	0.115	0	0.077	0.579
H=7	25	0	0	0	0	0	0	0	0
	30	0	0	0	0	0	0	0	0
	40	0	0.001	0	0.078	0.009	0	0.084	1.984
	50	0	0.008	0	0.221	0.038	0	0.048	1.225
H=10	25	0	0	0	0	0	0	0	0
	30	0	0	0	0	0	0	0	0
	40	0	0	0	0	0	0	0	0
	50	0	0	0	0	0	0	0	0

Running time results are presented in Table 5.7 for CPLEX and HACO2, and in Figs. 5.12, 5.13, and 5.14 for MPGA and HACO2.

The time spent by CPLEX to solve the HFMST problem clearly increases both with size and with hop value. This behaviour was expected since the number of feasible solutions increases rapidly with problem size due to the combinatorial nature of the problem.

As we have already said, one of our objectives was to infer on the behaviour of the HACO2 algorithm as small changes were introduced in cost function F1, which in this work is represented by F2 and F3. It is curious to notice that even though CPLEX is much influenced with the form of the cost function, with average running time increasing, in general, from F1 to F2 and then to F3, the HACO2 performance is almost the same for the three functions. Furthermore, the HACO2 heuristic can be up to 7 times faster than CPLEX. Not even F4 cost function seems to influence HACO2 average running times.

Tab. 5.7: Computational time results, for CPLEX and HACO2, obtained for cost functions F1, F2, F3 and F4 and $H = 3, 5, 7$, and 10.

		Time (s)							
		CPLEX				HACO2			
Function	N	3	5	7	10	3	5	7	10
F1	10	0.7	1.3	1.4	2.2	0.3	0.5	0.5	0.5
	12	0.9	1.6	2.2	3.4	0.4	0.7	0.8	0.7
	15	1.6	2.6	3.9	5.7	0.7	1.2	1.4	1.2
	17	2.0	3.7	5.5	10.1	1.0	1.7	2.0	1.9
	19	2.8	4.7	8.6	11.1	1.4	2.1	2.6	3.0
	25	5.1	9.5	18.9	26.0	2.7	4.3	5.4	5.9
	30	7.8	16.2	30.3	54.5	4.7	8.4	8.9	10.1
	40	-	37.2	71.2	117.8	-	15.1	27.7	24.6
	50	-	98.6	138.5	252.2	-	35.6	68.8	44.8
F2	10	1.2	1.9	3.4	4.3	0.3	0.5	0.5	0.5
	12	1.9	2.9	5.6	6.8	0.5	0.7	0.9	0.8
	15	2.8	5.2	8.8	11.1	0.7	1.2	1.5	1.3
	17	3.8	7.8	11.8	16.9	1.0	2.0	2.1	1.9
	19	5.3	8.3	13.9	21.8	1.3	2.2	2.7	3.0
	25	9.7	17.6	28.9	44.1	2.8	4.7	5.4	5.8
	30	14.3	29.1	47.2	70.2	4.6	9.1	8.9	9.8
	40	-	65.2	113.1	172.4	-	15.0	26.3	24.3
	50	-	161.0	225.9	288.5	-	35.5	50.7	47.3
F3	10	2.2	2.0	2.9	3.3	0.3	0.5	0.6	0.5
	12	2.6	2.6	4.7	5.5	0.5	0.7	0.9	0.9
	15	3.4	4.3	7.2	8.5	0.8	1.2	1.5	1.5
	17	4.1	6.4	10.1	21.7	1.2	1.9	2.1	2.3
	19	5.6	7.4	15.7	24.1	1.4	2.2	2.7	2.9
	25	11.0	13.5	33.3	52.2	2.9	4.4	5.3	5.7
	30	16.5	29.4	55.7	70.9	4.6	8.7	8.6	9.5
	40	-	60.5	139.4	239.3	-	15.0	25.3	23.4
	50	-	155.5	275.3	367.4	-	32.4	68.0	48.0
F4	10	-	-	-	-	0.3	0.7	0.9	0.9
	12	-	-	-	-	0.8	1.3	1.6	1.8
	15	-	-	-	-	1.3	1.5	2.5	2.5
	17	-	-	-	-	1.0	1.8	2.2	2.3
	19	-	-	-	-	2.9	3.7	3.7	4.2
	25	-	-	-	-	2.6	4.1	4.9	4.8
	30	-	-	-	-	3.9	6.7	8.6	8.4
	40	-	-	-	-	-	14.2	19.6	20.3
	50	-	-	-	-	-	27.8	48.9	47.1

This means that one way or the other, with more realistic industrial problems, the HACO2 heuristic will be able to give a very good answer within a reasonable amount of time, whereas exact methods will not. Furthermore, our algorithm does not seem to be influenced by the type of cost function taken into account (whereas CPLEX seems to find F2 and F3 more challenging), which allows for concluding it to be very robust.

In addition, we present graphical representations of the average running times for both the HACO2 and the MPGA, by cost function, in Figs. 5.12, 5.13 and 5.14. The results show no big difference between the MPGA and the HACO2 time performance. However, it should be noticed though that the MPGA was run on a much faster PC, an Intel Core2 processor at 2.4 GHZ.

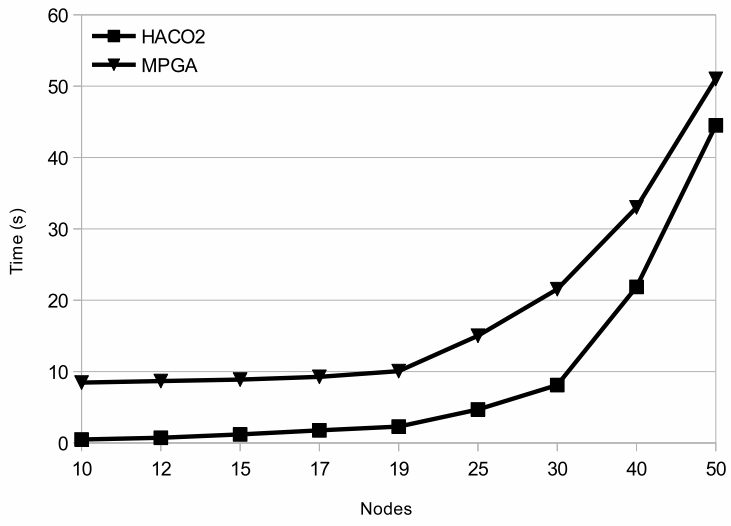

Fig. 5.12: Computational time results obtained with HACO2 and MPGA for F2 cost functions.

In order to infer about the evolution of CPLEX and HACO2 computational times, as well as about the HACO2 gap performance, we have also generated larger size problem instances with 60 and 80 nodes. For these sizes, we only consider five different groups, i.e. five different ratios between the variable and the fixed cost components. We have also considered a larger number of arcs in the networks that we have generated, since otherwise there would not be any feasible solutions for small hop values. The larger number of arcs in the networks for problems with 60 and with 80 nodes is the reason why they have feasible solutions even in cases where $H = 3$, as opposed to what happened with the set of problem instances that we have downloaded. Since these larger problems have not been solved with the MPGA, we only report on the results obtained by CPLEX and by HACO2.

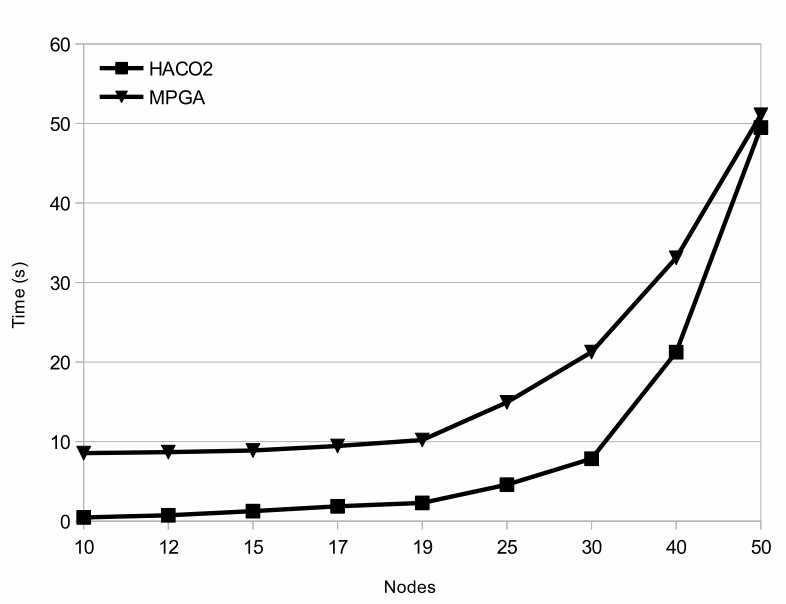

Fig. 5.13: Computational time results obtained with HACO2 and MPGA for F3 cost functions.

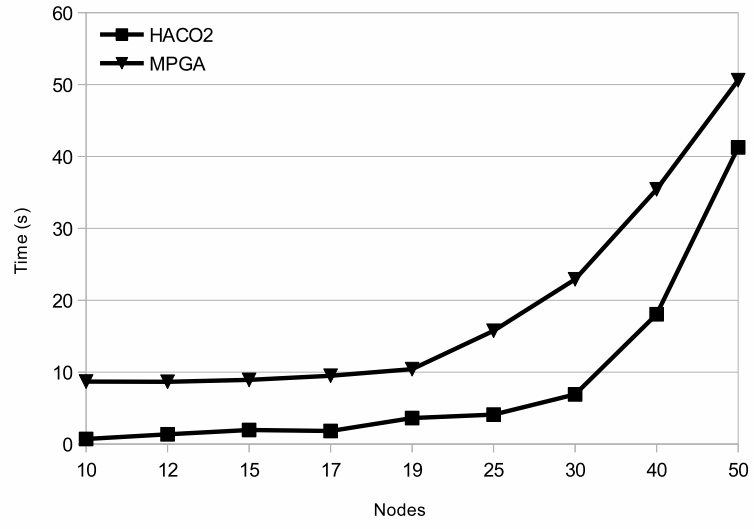

Fig. 5.14: Computational time results obtained with HACO2 and MPGA for F4 cost functions.

Tab. 5.8: Number of problem instances with 60 and with 80 nodes solved by CPLEX.

		No. of problems			
		CPLEX			
Function	N	3	5	7	10
F1	60	3	12	11	11
	80	3	9	9	9
F2	60	3	10	10	-
	80	3	8	-	-
F3	60	3	12	9	9
	80	6	9	9	-

Although HACO2 was able to solve all problem instances with 60 and with 80 nodes, it is important to refer that CPLEX was not due to memory failure. Since the behaviour of CPLEX was not uniform, regarding the number of problem instances solved without memory problems, we present in Table 5.8 the number of problem instances solved with CPLEX. Please recall that considering five groups and three problem instances in each of them, we have 15 problem instances to be solved per size and per hop value, each of which is to be solved 10 times. For both F2 and F3 cost functions CPLEX was not able to solve all the problem instances due to the form of the functions that increased the number of variables in the model (see Section 5.2.5.2), bringing once more problems with shortage of memory. In Table 5.9, we have the average optimality gap obtained by cost function and by hop value. As it can be seen, HACO2 maintains its lower average gaps and we were always able to find the optimum value within the 10 runs each problem instance was solved, for the problems CPLEX was able to provide the optimum value.

Regarding computational running times, see Table 5.10, besides the already observed increasing behaviour with problem size, for both CPLEX and HACO2, there is now stronger evidence of the influence of the hop values. It is notorious, by solving larger size problems, that running times increase with the hop parameter value, although for CPLEX the rate of influence is much larger. For instance, considering problems with 80 nodes and cost function F1, there is an increase of more than 75% of CPLEX running times from $H = 7$ to $H = 10$. Finally, these new problem instances confirm the conclusion already stated that CPLEX is influenced by the type of cost function whereas HACO2 is not.

Tab. 5.9: Average optimality gap (%) obtained with HACO2 for F1, F2, and F3 cost functions with $H = 3, 5, 7$, and 10 for problems with 60 and with 80 nodes.

		GAP			
		HACO2			
Function	N	3	5	7	10
F1	60	0	0	0	0
	80	0.001	0	0	0
F2	60	0	0	0	-
	80	0.001	0	-	-
F3	60	0.001	0	0	0
	80	0.001	0	0	-

Tab. 5.10: Computational time results, obtained with CPLEX and HACO2, for problems with 60 and with 80 nodes and considering cost functions F1, F2, and F3 and $H = 3, 5, 7$, and 10.

		Time (s)							
		CPLEX				HACO2			
Function	N	3	5	7	10	3	5	7	10
F1	60	65.9	246.8	297.6	517.5	16.6	31.5	36.7	40.8
	80	235.8	483.1	845.5	1511.8	56.1	89.7	105.8	116.7
F2	60	67.8	289.6	368.3	-	16.2	29.5	35.4	42.8
	80	197.7	541.1	-	-	52.1	88.5	109.1	117.0
F3	60	103.2	635.6	336.8	596.7	17.2	29.7	35.8	49.1
	80	607.3	549.4	943.3	-	53.4	87.2	131.7	122.0

5.6. Summary

In this Chapter the hop-constrained minimum cost flow spanning tree problem with non-linear costs is addressed by a hybrid algorithm based on ant colony optimization and on local search. The cost functions are of four types, and they comprise both fixed-charge and routing costs, which are very difficult to solve even for problems with a small number of nodes. We have solved problems with four different values of the hop-parameter and with a number of nodes ranging from 10 up to 50. We compared our results with the ones reported in the literature and our algorithm proved to be both very effective and very efficient. The solutions obtained were always better or as good as the ones provided by current literature, except for 13 problem instances out of the 2798 solved. It should be noticed that for the remaining 82 problem instances there are no feasible solutions. Fur-

thermore, our algorithm was always able to find a feasible solution, when there was one, whereas the MPGA was not. In fact, for 19 problem instances the MPGA failed to find any feasible solution, while for another 13 problem instances only in some runs a feasible solution was found. Although several cost functions with different complexity have been used, the algorithm performance has not been affected. Both solution quality and computational time remain of the same magnitude. Furthermore, the results obtained over 10 runs of the proposed algorithm are within 0.01% of each other proving the method to be very robust. Therefore, the proposed HACO2 heuristic proved to be a good alternative method to solve HMFST problems, having demonstrated a better performance over the existing heuristic (Fontes and Gonçalves, 2012) regarding gap values and over both this heuristic and CPLEX regarding time.

A preliminary version of the work presented in this chapter has been presented in an international conference (Monteiro et al, 2011b). A paper based on the final results of this chapter has already been submitted for publication in an international journal.

6

Conclusions and Future Work

6.1. Conclusions

In this work we have addressed two types of nonlinear costs network flow problems, the single source uncapacitated minimum concave cost network flow problem and the hop-constraint minimum cost flow spanning tree problem. Both these problems can be used to model several practical problems, specially in telecommunications, transportation, and in distribution networks, thus the interest in solving them.

Exact methods can be used to solve small instances of these problems. In here, we use CPLEX to solve the problems, but only for some of the cost functions considered. However, exact methods are not an alternative when the problem size increases, due to the exponentially increasing running times.

Two ACO based algorithms were developed to solve both problems in an attempt to provide a good alternative to exact methods. From the results obtained with both algorithms, we were able to draw the following conclusions:

ACO Parameters

The study we provide on the performance of the algorithms with the variation of the parameter values revealed that some are of vital importance for the good performance of the algorithm, while others can be set to almost any reasonable value within the problem context. We now summarize the results obtained for the parameters with a major influence on the gap results.

The heuristic information η_{ij} is aimed at controlling the relative importance of the local information that is to be used by ants while constructing their solutions, i.e., when ants are confronted with the choice of an arc to be inserted into the solution. For obvious reasons, this parameter is closely related with the cost functions considered in the problems. Therefore, the best results are obtained, when η_{ij} is set to the inverse of the sum between the parameter associated with the fixed-charge (whenever there is one to be considered) and the parameter associated with the component of the cost function which is proportional to the flow.

The parameters weighting the pheromone value and the heuristic information, α and β respectively, have a major role in the probability function. For these parameters, and within the context of the addressed problems, an equilibrium was found by using a larger weight, for the information of the pheromone value, than the one attributed to the heuristic information.

The evaporation rate ρ sets the size of the search space where ants are allowed to find solutions. In this case, the search space size increases with the increase of ρ. The empirical results we have obtained indicate that evaporation rates below 10% and above 20%, tend to have poor performances, because the ants are either too constrained or have too much freedom.

Cost Functions

The cost functions used in these problems were mainly motivated by the behaviour of the economy. It is rarely the case of a cost having a linear evolution. It is more likely that it is influenced by economies of scale, in some cases leading to an initial fast growth on costs and to a slow down on the growth as quantities increase. These cost functions are usually of a concave nature.

Three concave cost functions were considered for the SSU MCNFP, one is a fixed-charge function, and the other two are second order polynomials, one with and another without a fixed charge component. The results obtained do not sustain the hypotheses that

the behaviour of the HACO1 algorithm is affected by the form of the function, neither in terms of gap nor in terms of time. Rather, they show the method to be quite independent of the cost function type.

Regarding the HMFST problem, the nonlinear cost functions are of four types, and they comprise both fixed-charge and routing costs, which are very difficult to solve even for problems with a small number of nodes. We have introduced an increasing complexity on the cost functions, from F1 to F4, by introducing a discontinuity other than at the origin. Thus, except for cost function F1, which is a fixed-charge, all other functions are neither concave nor convex. The performance of the HACO2 algorithm was not affected by the cost function type, not even in terms of running time. The same cannot be said as to the running time performance of CPLEX, as it shows evidence of being affected by the cost functions, at least for cost function F3, suggesting that times will increase more rapidly for F3 than for F1 and F2 with the increase of problem size. Recall that CPLEX is unable to solve problems with cost function of type F4.

Solution Quality

We have solved problems ranging from 10 up to 50 nodes for the SSU MCNFP that are publicly available at the OR library. We compare our results with the ones in the literature and the solutions obtained were always as good or better than the ones obtained with another heuristic algorithm although the number of solutions evaluated by the HACO1 was much smaller. We have also generated and solved problem instances with $n = 60, 80, 100, 120$ nodes in order to investigate the evolution of the solution quality with larger size problems. The HACO1 algorithm maintained its good performance being always able to find the optimum value for all the problem instances considered.

The HMFST problem was solved for problems with 10 up to 80 nodes and with four different values for the hop-parameter $H = \{3, 5, 7, 10\}$. The solutions obtained were always better or as good as the ones obtained with another heuristic algorithm, except for 13 problem instances out of $240 \times 4 \times 3$ of which 2798 problem instances have a feasible solution. Therefore, the proposed HACO2 heuristic proved to be a good alternative method to solve HMFST problems, having demonstrated a better performance over the existing heuristic (Fontes and Gonçalves, 2012) regarding gap values. Furthermore, our algorithm was always able to find a feasible solution, when there was one, whereas the MPGA was not. In fact, for 19 problem instances the MPGA failed to find any feasible solution, while for 13 problem instances only in some runs a feasible solution was found, out of 2798 problem instances to be solved.

Generally, problems considering small hop values are more difficult to solve. The HACO2 that was developed did not show evidence of being greatly affected by this parameter at least for the lowest hop value used. The intermediate hop values considered did show a small degradation on the optimality gap, but only in a small number of cases.

Running Times

Regarding the SSU MCNFP, the HACO1 algorithm is able to decrease up to 11 times the HGA running time. Furthermore, although the computational time requirements of the HACO1 also increase with problem size, the rate of increase is much smaller than that of the HGA. This is very important and leads us to believe that for larger problems the HGA might not be an alternative to the HACO1. We have then proved HACO1 to be an alternative method to solve SSU concave MCNFP, with the advantage of representing a much lower computational effort, since it has to evaluate five times less solutions, when compared with HGA. When in comparison with CPLEX, although for smaller problems the computational times are much alike, when the problems to be solved have more than 100 demand nodes running times for CPLEX increase rapidly, whereas the increase of HACO1 running times is much slower. For example, for problems with 120 nodes the CPLEX requires approximately 21 minutes to solve the problems, whereas for the HACO1 4 minutes suffice.

For the HMFST problem we have improved upon the results of both the exact methods used to solve the problem, CPLEX and dynamic programming. Nonetheless, our times do increase with problem size, as is expected with combinatorial optimization problems. This increase in time is mainly due to the local search procedure and to a certain degree of stagnation of the solution leading to an increase on the number of iterations used by the algorithm. Although we have developed a mechanism to overcome the latter, the stagnation is still present. Nonetheless, we do improve upon running times of both the MPGA heuristic and CPLEX. Also, it is clear, specially with the results for the larger problems (60 and 80 nodes) that an increasing hop value leads to an increase on the computational times.

6.2. Future Work

This work provided us with very encouraging results obtained by solving these hard problems with the two HACO algorithms that were developed. We were able to show that, although the size of the problem instances was directly related to the time spent in solving

each problem, when comparing the cost of the solutions found by our algorithms with the optimum values, or upper bounds, when the formers were not available, the results were not biased by size. This is an indicator that ACO algorithms can be further applied to other network flow problems, that have not yet been solved by them, or whose results have a margin to be improved.

Regarding the problems that we have solved, there is still some space for improvement. For larger networks, which may include hundreds or thousands of demand nodes, the computational times may become excessive. Therefore, finding an optimum solution earlier would be very important. An idea to begin with would be to reduce the number of iterations to consider the algorithm stagnated and also to implement a tabu list of the most used arcs. Therefore, every time the algorithm has to make a restart of the pheromone matrix, the tabu list reduces the pheromone on the arcs accordingly to the number of times the arc has been used before. This way, the less used arcs would be the first ones to be tested, in order for the HACO algorithm to jump to a totally different search area, but not totally at random.

Given the good results obtained, we are encouraged to proceed our study on ACO algorithms and their applicability to other network flow problems. Some optimization problems have such a complex structure that no attempt to solve them with ACO has been reported. For instance, as far as we are aware of only a special case of concave cost transportation problems has been solved with an ACO based algorithm (see Section 3.3.3). Thus, it would be very interesting to approach transportation problems with ACO metaheuristics.

Bibliography

Abdalla A, Deo N (2002) Random-tree diameter and the diameter-constrained mst. International Journal of Computer Mathematics 79(6):651–663

Abramson D, Krishnamoorthy M, Dang H (1999) Simulated annealing cooling schedules for the school timetabling problem. Asia-Pacific Journal of Operational Research 16:1–22

Adlakha V, Kowalski K (2003) A simple heuristic for solving small fixed-charge transportation problems. Omega 31:205–211

Adlakha V, Kowalski K, Lev B (2010) A branching method for the fixed charge transportation problem. Omega 38:393–397

Afshar MH (2005) A new transition rule for ant colony optimization algorithms: application to pipe network optimization problems. Engineering Optimization 37(5):525–540

Aguado JS (2009) Fixed charge transportation problems: a new heuristic approach based on lagrangean relaxation and the solving of core problems. Annals of Operations Research 172:45–69

Ahuja RK, Magnanti TL, Orlin JB, Reddy M (1995) Applications of network optimization. In: Network Models, volume 7 of Handbooks in Operations Research and Management Science, pp 1–83

Ahuja RK, Ergun O, Orlin JB, Punnen AP (2002) A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics 123(1-3):75–102

Akgün I (2011) New formulations for the hop-constrained minimum spanning tree problem via Sherali and Driscoll's tightened Miller-Tucker-Zemlin constraints. Computers & Operations Research 38:277–286

Alaya I, Solnon C, Ghédira K (2004) Ant algorithm for the multi-dimensional knapsack problem. In: International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004, pp 63–72

Aleskerov F, Ersel H, Yolalan R (2003) Personnel allocation among bank branches using a two-stage multi-criterial approach. European Journal of Operational Research 148:116–125

Altiparmak F, Karaoglan I (2006) An adaptive tabu-simulated annealing for concave cost transportation problems. Journal of the Operational Research Society 59(3):1–11

Altiparmak F, Karaoglan I (2007) A genetic ant colony optimization approach for concave cost transportation problems. In: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, pp 1685–1692

Arya V, Garg N, Khanekar R, Meyerson A, Munagala K, Pandit V (2004) Local search heuristics for k-median and facility location problems. SIAM Journal of Computing 33(3):544–562

Badr A, Fahmy A (2004) A proof of convergence for ant algorithms. Information Sciences 160:267–279

Balinski ML (1961) Fixed cost transportation problem. Naval Research Logistics Quarterly 8:41–54

Barr F, Glover F, Klingman D (1981) A new optimization method for fixed charge transportation problems. Operations Research 29:448–463

Barros A, Dekker R, Scholten V (1998) A two-level network for recycling sand: A case study. European Journal of Operational Research 110(2):199–214

Baykasoglu A, Dereli T, Sabuncu I (2006) An ant colony algorithm for solving budget constrained and unconstrained dynamic facility layout problems. Omega 34:385–396

Beasley J (2012) Or-library. `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/netflowccinfo.html`, last accessed December-2012

Benson H (1995) Concave minimization: Theory, applications and algorithms. In: Horst R, Pardalos PM (eds) Handbook of Global Optimization, Kluwer Academic Publishers, pp 43–148

Benson H (2001) Concave programming. In: Floudas CA, Pardalos PM (eds) Encyclopedia of Optimization, vol 2, Kluwer Academic Publishers, pp 315–318

Bernardino EM, Bernardino AM, Sánchez-Pérez JM, Gómez-Pulido JA, Vega-Rodríguez MA (2009) A hybrid ant colony optimization algorithm for solving the terminal assignment problem. In: IJCCI 2009 - International Joint Conference on Computational Intelligence

Bertsekas DP (1998) Network Optimization: continuous and discrete models. Athena Scientific, Belmont, MA

Bin Y, Zhong-Zhen Y, Baozhen Y (2009) An improved ant colony optimization for vehicle routing problem. European Journal of Operational Research 196:171–176

Birattari M, Paquete L, Stützle T, Varrentrapp K (2001) Classification of metaheuristics and design of experiments for the analysis of components. Tech. Rep. AIDA-01-05, Intellektik Darmstadt University of Technology, Darmstadt, Germany

Blum C, Blesa MJ (2005) New metaheuristic approaches for the edge-weighted k-cardinality tree problem. Computers & Operations Research 32:1355–1377

Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys 35(3):268–308

Blum C, Roli A, Dorigo M (2001) HC-ACO: the hyper-cube framework for ant colony optimization. In: Proceedings of Metaheuristics International Conference MIC'2001, Porto, Portugal, July 2001, pp 399–403

Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm Intelligence - from Natural to Artificial Systems. Oxford University Press, New York

Borgatti SP (2005) Centrality and network flow. Social Networks 27(1):55–71

Bouhafs L, Hajjam A, Koukam A (2006) A combination of simulated annealing and ant colony system for the capacitated location-routing problem. In: KES (1), pp 409–416

Brannlund H, Bubenko JA, Sjelvgren D, Andersson N (1986) Optimal short term operation planning of a large hydrothermal power system based on a nonlinear network flow concept. Power Systems, IEEE Transactions on 1(4):75–81

Broder AZ (1989) Generating random spanning trees. In: 30th Annual Symposium on Foundations of Computer Science, 30 October-1 November 1989, Research Triangle Park, North Carolina, USA, IEEE, pp 442–447

Brotcorne L, Cirinei F, Marcotte P, Savard G (2012) A tabu search algorithm for the network pricing problem. Computers & Operations Research 39(11):2603 – 2611

Bui TN, Zrncic CM (2006) An ant-based algorithm for finding degree-constrained minimum spanning tree. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM, New York, USA, GECCO '06, pp 11–18

Bullnheimer B, Hartl RF, Strauß C (1997) A new rank based version of the ant system - a computational study. Central European Journal for Operations Research and Economics 7:25–38

Burkard R, Dell'Amico M, Martello S (2009) Assignment Problems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA

Burkard RE, Dollani H, Thach PT (2001) Linear approximations in a dynamic programming approach for the uncapacitated single-source minimum concave cost network flow problem in acyclic networks. Journal of Global Optimization 19:121–139

Calik H, Alumur SA, Kara BY, Karasan OE (2009) A tabu-search based heuristic for the hub covering problem over incomplete hub networks. Computers & Operations Research 36(12):3088 – 3096

Charikar M, Guha S (1999) Improved combinatorial algorithms for the facility location and k-median problems. In Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science pp 378–388

Chen CH, Ting CJ (2008) Combining lagrangian heuristic and ant colony system to solve the single source capacitated facility location problem. Transportation Research Part E: Logistics and Transportation Review 44(6):1099 – 1122, DOI 10.1016/j.tre.2007.09.001

Chen S, Coolbeth M, Dinh H, Kim YA, Wang B (2009) Data collection with multiple sinks in wireless sensor networks. In: Liu B, Bestavros A, Du DZ, Wang J (eds) Wireless Algorithms, Systems, and Applications, Lecture Notes in Computer Science, vol 5682, Springer Berlin / Heidelberg, pp 284–294

Chu P, JE B (1997) A genetic algorithm for the generalised assignment problem. Computers & Operations Research 24(1):17–23

Clarke G, Wright JW (1964) Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12:568–581

Cordon O, Herrera F, Stützle T (2002) A review on the ant colony optimization metaheuristic: Basis, models and new trends. Mathware & Soft Computing 9:141–175

Costamagna E, Fanni A, Giacintod G (1998) A tabu search algorithm for the optimisation of telecommunication networks. European Journal of Operational Researchs 106(2-3):357–372

CPLEX (2012) Ibm ilog cplex optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/, last accessed December-2012

Crawford B, Castro C (2006) Integrating lookahead and post processing procedures with aco for solving set partitioning and covering problems. In: ICAISC, pp 1082–1090

Dahl G (1998) The 2-hop spanning tree problem. Operations Research Letters 23:21–26

Dahl G, Gouveia L, Requejo C (2006) On formulations and methods for the hop-constrained minimum spanning tree problem. In: Resende MGC, Pardalos PM (eds) Handbook of Optimization in Telecommunications, Springer US, pp 493–515

Dang C, Sun Y, Wang Y, Yang Y (2011) A deterministic annealing algorithm for the minimum concave cost network flow problem. Neural Networks 24:699–708

Dell'Amico M, Lodi A, Maffioli F (1999) Solution of the cumulative assignment problem with a well-structured tabu search method. Journal of Heuristics 5:123–143

Deneubourg JL, Aron S, Goss S, Pasteels JM (1990) The self-organizing exploratory pattern of the argentine ant. Journal of Insect Behaviour 3:159–168

Derbel H, Jarboui B, Hanafi S, Chabchoub H (2010) An iterated local search for solving a location-routing problem. Electronic Notes in Discrete Mathematics 36:875–882

Desrocher M, Laporte G (1991) Improvements to the miller-tucker-zemlin subtour elimination constraints. Operations Research Letters 10:27–36

Diaz J, Fernandez E (2001) A tabu search heuristic for the generalized assignment problem. European Journal of Operational Research 132:22–38

Dijkstra EW (1959) A note on two problems in connexion with graphs. Numerische Mathematik 1:269–271

Dorigo M (1992) Optimization, learning and natural algorithms. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan

Dorigo M, Blum C (2005) Ant colony optimization theory: A survey. Theoretical Computer Science 344:243–278

Dorigo M, Di Caro G (1999) The ant colony optimization meta-heuristic, McGraw-Hill Ltd., UK, Maidenhead, UK, England, pp 11–32

Dorigo M, Gambardella LM (1997) Ant colony system: A cooperative learning approach to the travelling salesman problem. IEEE Transactions on Evolutionary Computation 1:53–66

Dorigo M, Stützle T (2004) Ant Colony Optimization. MIT Press, Cambridge, MA

Dorigo M, Maniezzo V, Colorni A (1991) Positive feedback as a search strategy. Tech. Rep. 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy

Dorigo M, Maniezzo V, Colorni A (1996) The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics - Part B 26(1):29–41

Dorigo M, Caro GD, Gambardella LM (1999) Ant algorithms for discrete optimization. Artificial Life 5(2):137–172

Driebeek NJ (1966) An algorithm for the solution of mixed integer programming problem. Management Science 12:576–587

Duhamel C, Lacomme P, Prins C, Prodhon C (2010) A GRASPxELS approach for the capacitated location-routing problem. Computers & Operations Research 37(11):1912–1923

Edmonds J (1968) Optimum branchings. In: Mathematics of the Decision Sciences, Lectures in Applied Mathematics, G. B. Dantzig and A. F. Veinott, Jr

Eggleston HG (1963) Convexity. Cambridge tracts in mathematics and mathematical physics

Eswaramurthy V, Tamilarasi A (2009) Hybridizing tabu search with ant colony optimization for solving job shop scheduling problems. The International Journal of Advanced Manufacturing Technology 40:1004–1015

Faria J, Silva C, Sousa J, Surico M, Kaymak U (2006) Distributed optimization using ant colony optimization in a concrete delivery supply chain. In: Yen GG, Lucas SM, Fogel G, Kendall G, Salomon R, Zhang BT, Coello CAC, Runarsson TP (eds) Proceedings of the 2006 IEEE Congress on Evolutionary Computation, IEEE Press, Vancouver, BC, Canada, pp 73–80

Feldman E, Lehrer FA, Ray TL (1966) Warehouse location under continuous economies of scale. Management Science 12(9):670–684

Fernandes M, Gouveia L, Voß S (2007) Determining hop-constrained spanning trees with repetitive heuristics. Journal of Telecommunications and Information Technology 4:16–22

Fontes DBMM (2010) Optimal hop-constrained trees for nonlinear cost flow networks. Infor 48

Fontes DBMM, Gonçalves JF (2007) Heuristic solutions for general concave minimum cost network flow problems. Networks 50:67–76

Fontes DBMM, Gonçalves JF (2012) A multi-population hybrid biased random key genetic algorithm for hop-constrained trees in nonlinear cost flow networks. Optimization Letters pp 1–22, DOI 10.1007/s11590-012-0505-5

Fontes DBMM, Hadjiconstantinou E, Christofides N (2003) Upper bounds for single-source uncapacitated concave minimum-cost network flow problems. Networks 41(4):221–228

Fontes DBMM, Hadjiconstantinou E, Christofides N (2006a) A branch-and-bound algorithm for concave network flow problems. Journal of Global Optimization 34:127–155

Fontes DBMM, Hadjiconstantinou E, Christofides N (2006b) A dynamic programming approach for single-source uncapacitated concave minimum cost network flow problems. European Journal of Operational Research 174:1205–1219

Fontes DBMM, Hadjiconstantinou E, Christofides N (2006c) Lower bounds from state space relaxations for network routing problems. J Global Optim 34:97–125

Frey H, Ingelrest F, Simplot-Ryl D (2008) Localized minimum spanning tree based multicast routing with energy-efficient guaranteed delivery in ad hoc and sensor networks. In: Proceedings of the 2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks, IEEE Computer Society, Washington, DC, USA, pp 1–8

Gallo G, Sandi C, Sodini C (1980) An algorithm for the min concave cost flow problem. Euro 4:249–255

Gambardella LM, Éric Taillard, Agazzi G (1999a) MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In: New Ideas in Optimization, McGraw-Hill, pp 63–76

Gambardella LM, Taillard ED, Dorigo M (1999b) Ant colonies for the quadratic assignment problem. Journal of the Operational Research Society 50:167–176

García-Martínez C, Cordón O, Herrera F (2007) A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. European Journal of Operational Research 180(1):116–148

Garey MR, Johnson DS (1979) Computers and Intractability: a guide to the theory of NP-Completeness. W. H. Freeman and Company

Gavish B (1983) Formulations and algorithms for the capacitated minimal directed tree problem. Journal of the ACM 30:118–132

Gendreau M, Iori M, Laporte G, Martello S (2008) A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. Networks 51(1):4–18

Geoffrion AM, Graves GW (2010) Multicommodity distribution system design by benders decomposition. In: Sodhi MS, Tang CS (eds) A Long View of Research and Practice in Operations Research and Management Science, International Series in Operations Research & Management Science, vol 148, Springer US, pp 35–61

Geunes J, Pardalos P (2005) Supply Chain Optimization. Springer, Berlin

Ghosh D (2003) Neighborhood search heuristics for the uncapacitated facility location problem. European Journal of Operational Research 150:150–162

Glover F (1986) Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13:533–549

Glover F (1990) Tabu search - part ii. ORSA Journal on Computing 2:4–32

Glover F, Amini M, Kochenberger G (2005) Parametric ghost image processes for fixed-charge problems: a study of transportation networks. Journal of Heuristics 11:307–336

Gonçalves JF, Resende MG, Mendes JJ (2011) A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. Journal of Heuristics 17:467–486

Gonçalves JF, Resende MGC (2012) A parallel multi-population biased random-key genetic algorithm for a container loading problem. Computers & Operations Research 39(2):179–190

Goss S, Aron S, Deneubourg J, Pasteels J (1989) Self-organized shortcuts in the argentine ant. Naturwissenschaften 76:579–581

Gouveia L (1995) Using the Miller-Tucker-Zemlin constraints to formulate a minimal spanning tree problem with hop constraints. Computers and Operations Research 22:959–970

Gouveia L (1996) Multicommodity flow models for spanning trees with hop constraints. European Journal of Operational Research 95:178–190

Gouveia L (1998) Using variable redefinition for computing lower bounds for minimum spanning and steiner trees with hop constraints. Informs Journal on Computing 10:180–188

Gouveia L, Martins P (1999) The capacitated minimal spanning tree problem: an experiment with a hop-indexed model. Annals of Operations Research 86:271–294

Gouveia L, Martins P (2000) A hierarchy of hop-indexed models for the capacitated minimum spanning tree problem. Networks 35:1–16

Gouveia L, Requejo C (2001) A new lagrangean relaxation approach for the hop-constrained minimum spanning tree problem. European Journal of Operational Research 132:539–552

Gouveia L, Simonetti L, Uchoa E (2007) Modelling the hop-constrained minimum spanning tree problem over a layered graph. In: In Proceedings of the International Network Optimization Conference, Spa, Belgium,

Gouveia L, Paias A, Sharma D (2011) Restricted dynamic programming based neighborhoods for the hop-constrained minimum spanning tree problem. Journal of Heuristics 17:23–37

Guisewite GM, Pardalos PM (1990) Minimum concave-cost network flow problems: Applications, complexity, and algorithms. Annals of Operations Research 25:75–99

Guisewite GM, Pardalos PM (1991a) Algorithms for the single-source uncapacitated minimum concave-cost network flow problem. Journal of Global Optimization 3:245–265

Guisewite GM, Pardalos PM (1991b) Global search algorithms for minimum concave-cost network flow problems. Journal of Global Optimization 1:309–330

Guisewite GM, Pardalos PM (1993) A polynomial time solvable concave network flow problem. Networks 23:143–147

Hajiaghaei-Keshteli M, Molla-Alizahed-Zavardehi S, Tavakkoli-Moghaddam R (2010) Addressing a nonlinear fixed-charge transportation problem using a spanning tree-based genetic algorithm. Computers & Industrial Engineering 59:259–271

Handler GY (1978) Minimax location of a facility in an undirected graph. Transportation Science 7:287–293

Hao J, Orlin JB (2002) A faster algorithm for finding the minimum cut of a graph. In: Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms, pp 165–174

Held M, Wolfe P, Crowder HP (1974) Validation of subgradient optimization. Mathematical Programming 6:62–88

Hochbaum D (1993) Polynomial and strongly polynomial algorithms for convex network optimization. In: Du DZ, Pardalos PM (eds) Network Optimization Problems: Algorithms Applications and Complexity, World Scientific, pp 63–92

Hochbaum DS (2005) Complexity and algorithms for convex network optimization and other nonlinear problems. 4OR 3(3):171–216

Hochbaum DS (2007) Complexity and algorithms for nonlinear optimization problems. Annals of Operations Research 153:257–296

Hochbaum DS, Segev A (1989) Analysis of a flow problem with fixed charges. Networks 19:291–312

Holland J (1975) Adaption in natural and artificial systems. MIT Press

Horst R, Thoai NV (1998) An integer concave minimization approach for the minimum concave cost capacitated flow problem on networks. OR Spectrum 20:47–53

Hwang IS, Cheng RY, Tseng WD (2007) A novel dynamic multiple ring-based local restoration for point-to-multipoint multicast traffic in wdm mesh networks. Photonic Network Communications 14:23–33

Jaramillo JH, Bhadury J, Batta R (2002) On the use of genetic algorithms to solve location problems. Computers and Operations Research 29:761–779

Jensen PA, Barnes JW (1980) Network Flow Programming. John Wiley & Sons, New York, NY, USA

Katagiri H, Hayashida T, Nishizaki I, Guo Q (2012) A hybrid algorithm based on tabu search and ant colony optimization for k-minimum spanning tree problems. Expert Systems with Applications 39:5681–5686

Kennington J, Unger V (1976) A new branch-and-bound algorithm for the fixed charge transportation problem. Management Science 22:1116–1126

Kim D, Pardalos P (2000) Dynamic slope scaling and trust interval techniques for solving concave piecewise linear network flow problems. Networks 35(3):216–222

Kim D, Pardalos PM (1999) A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. Networks 35:216–222

Kim D, Pan X, Pardalos PM (2006) An enhanced dynamic slope scaling procedure with tabu scheme for fixed charge network flow problems. Computational Economics 27:273–293

Kirkpatrik S, Gelatt C, Vecchi M (1983) Optimization by simulated annealing. Science 220:671–680

Kliewer N, Amberg B, Amberg B (2012) Multiple depot vehicle and crew scheduling with time windows for scheduled trips. Public Transport 3:213–244

Klinz B, Tuy H (1993) Minimum concave cost network flow problem with a single nonlinear arc cost. In: Network Optimization Problems, D. Z. Du and P. M. Pardalos

Klose A (2008) Algorithms for solving the single-sink fixed-charge transportation problem. Computers & Operations Research 35:2079–2092

Koopmans TC, Berkmann MJ (1957) Assignment problems and the location of economic activities. Econometrica 25:53–76

Krüger F, Middendorf M, Merkle D (1998) Studies on a parallel ant system for the bsp model, unpublished manuscript

Kuehn AA, Hamburger MJ (1963) A heuristic program for locating warehouses. Management Science 9:643–666

Ladner RE (1975) On the structure of polynomial time reducibility. J ACM 22:155–171

Lamar BW (1993) A method for solving network flow problems with general nonlinear arc costs. In: Pardalos DZDPM (ed) Network Optimization Problems: Algorithms Applications and Complexity, World Scientific, pp 147–167

Lessing L, Dumitrescu I, Stützle T (2004) A comparison between aco algorithms for the set covering problem. In: ANTS, pp 1–12

Lin SW, Yu VF, Chou SY (2009) Solving the truck and trailer routing problem based on a simulated annealing heuristic. Computers & Operations Research 36(5):1683 – 1692

LINGO (2012) Lindo systems inc. http://www.lindo.com, last accessed December-2012

Liu J (1999) The impact of neighbourhood size on the process of simulated annealing: Computational experiments on the flowshop scheduling problem. Computers & Industrial Engineering 37(1-2):285–288

Lozovanu D (1983) Properties of optimal solutions of a grid transport problem with concave function of the flows on the arcs. Engineering Cybernetics 20:34–38

Machado P, Tavares J, Pereira F, Costa E (2002) Vehicle routing problem: Doing it the evolutionary way. GECCO 2002:Proceedings of the Genetic and Evolutionary Computation Conference

Maniezzo V (1999) Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS journal on computing 11:358–369

Mares M (2008) The saga of minimum spanning trees. Computer Science Review 2:165–221

Martin R (1987) Generating alternative mixed-integer programming models using variable redefinition. Operations Research 35:820–831

Mehrabi AD, Mehrabi S, Mehrabi A (2009) A prunning based ant colony algorithm for minimum vertex cover problem. In: International Joint Conference on Computational Intelligence (IJCCI'09), pp 281–286

Melo MT, Nickel S, da Gama FS (2005) Dynamic multi-commodity capacitated facility location: a mathematical modeling framework for strategic supply chain planning. Computers & Operations Research 33:181–208

Michalewicz Z, Fogel DB (2002) How to solve it: modern heuristics. Springer-Verlag

Middendorf M, Reischle F, Schmeck H (2002) Multi colony ant algorithms. Journal of Heuristics 8:305–320

Miller C, Tucker A, Zemlin R (1960) Integer programming formulations and travelling salesman problems. Journal of the ACM 7:326–329

Mitchell M (1999) An Introduction to Genetic Algorithms. The MIT Press

Mladenovic N, Labbé M, Hansen P (2003) Solving the p-center problem with tabu search and variable neighborhood search. Networks 42(1):48–64

Monteiro MSR (2005) Bank-branch location and sizing under economies of scale. Master's thesis, Faculdade de Economia da Universidade do Porto

Monteiro MSR, Fontes DBMM (2006) Locating and sizing bank-branches by opening, closing or maintaining facilities. In Proceedings of the International Scientific Annual Conference Operations Research 2005 - OR2005, VerlagMonteiro2006

Monteiro MSR, Fontes DBMM, Fontes FACC (2011a) An ant colony optimization algorithm to solve the minimum cost network flow problem with concave cost functions. In: Krasnogor N, Lanzi PL (eds) GECCO, ACM, pp 139–146

Monteiro MSR, Fontes DBMM, Fontes FACC (2011b) An ant colony optimization approach for minimum spanning tree problems in nonlinear costs network flows, unpublished manuscript, presented at the International Conference on Operations Research (OR2011), August 30 to September 2, Zürich, Switzerland

Monteiro MSR, Fontes DBMM, Fontes FACC (2012a) Ant colony optimization: a literature survey, working paper, Nº 474, Universidade do Porto, Faculdade de Economia

Monteiro MSR, Fontes DBMM, Fontes FACC (2012b) Concave minimum cost network flow problems solved with a colony of ants. Journal of Heuristics DOI 10.1007/s10732-012-9214-6

Monteiro MSR, Fontes DBMM, Fontes FACC (2012c) Solving concave network flow problems, working paper, Nº 475, Universidade do Porto, Faculdade de Economia

Mullen R, Monekosso D, Barman S, Remagnino P (2009) A review of ant algorithms. Expert Systems with Applications 36:9608–9617

Musa R, Arnaout JP, Jung H (2010) Ant colony optimization algorithm to solve for the transportation problem of cross-docking network. Computers & Industrial Engineering 59(1):85 – 92

Nahapetyan A, Pardalos P (2007) A bilinear relaxation based algorithm for concave piecewise linear network flow problems. Journal of Industrial and Management Optimization 3:71–85

Nahapetyan A, Pardalos P (2008) Adaptive dynamic cost updating procedure for solving fixed charge network flow problems. Computational Optimization and Applications 39:37–50

Neumann F, Witt C (2010) Ant colony optimization and the minimum spanning tree problem. Theoretical Computer Science 411(25):2406 – 2413

Nguyen VP, Prins C, Prodhon C (2012) A multi-start iterated local search with tabu list and path relinking for the two-echelon location-routing problem. Engineering Applications of Artificial Intelligence 25(1):56–71

Orlin JB, Sharma D (2004) Extended neighborhood: Definition and characterization. Mathematical Programming 101:537–559

Ortega F, Wolsey LA (2003) A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. Networks 41:143–158

Osman I (1995) Heuristics for the generalised assignment problem: Simulated annealing and tabu search approaches. OR Spectrum 17:211–225

Palekar US, Karwan MH, Zionts S (1990) A branch-and-bound method for the fixed charge transportation problem. Management Science 36(9):1092–1105

Papadimitriou C (1978) The complexity of the capacitated tree problem. Networks 8:217–230

Pato MV, Moz M (2008) Solving a bi-objective nurse rerostering problem by using a utopic pareto genetic heuristic. Journal of Heuristics 14(4):359–374

Pham D, Ghanbarzadeh A, Koç E, Otri S, Rahim S, Zaidi M (2005) The bees algorithm - a novel tool for complex optimisation problems. Tech. rep., Manufactoring Engineering Centre, Cardiff University, UK

Pishvaee M, Kianfar K, Karimi B (2010) Reverse logistics network design using simulated annealing. The International Journal of Advanced Manufacturing Technology 47:269–281

Poorzahedy H, Abulghasemi F (2005) Application of ant system to network design problem. Transportation 32:251–273

Poorzahedy H, Rouhani OM (2007) Hybrid meta-heuristic algorithms for solving network design problem. European Journal of Operational Research 182:578–596

Pour HD, Nosraty M (2006) Solving the facility layout and location problem by ant-colony optimization-meta heuristic. International Journal of Production Research 44:5187–5196

Prim R (1957) Shortest connection networks and some generalisations. Bell Systems Technology Journal 36:1389–1401

Putha R, Quadrifoglio L, Zechman E (2012) Comparing ant colony optimization and genetic algorithm approaches for solving traffic signal coordination under oversaturation conditions. Computer-Aided Civil and Infrastructure Engineering 27:14–28

Rappos E, Hadjiconstantinou E (2004) An ant colony heuristic for the design of two-edge connected flow networks. In: ANTS Workshop, pp 270–277

Rardin RL, Uzsoy R (2001) Experimental evaluation of heuristic optimization algorithms: a tutorial. Journal of Heuristics 7:261–304

Rebennack S, Nahapetyan A, Pardalos P (2009) Bilinear modeling solution approach for fixed charge network flow problems. Optimization Letters 3:347–355

Reimann M, Laumanns M (2006) Savings based ant colony optimization for the capacitated minimum spanning tree problem. Computers & Operations Research 33:1794–1822

dos Santos AC, Lucena A, Ribeiro CC (2004) Solving diameter constrained minimum spanning tree problems in dense graphs. In: WEA, pp 458–467

Santos L, ao Coutinho-Rodrigues J, Current JR (2010) An improved ant colony optimization based algorithm for the capacitated arc routing problem. Transportation Research Part B: Methodological 44(2):246 – 266

Schaerf A (1999) Local search techniques for large high school timetabling problems. IEEE Transactions on Systems, Man and Cybernetics- part A: Systems and Humans 29(4):368–377

Shah-Hosseini H (2009) The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. International Journal of Bio-Inspired Computation 1:71–79

Shangyao Y, Chung-Gee L (1997) Airline scheduling for the temporary closure of airports. Transportation Science 31(1):72–82

Sherali HD, Driscoll PJ (2002) On tightening the relaxations of Miller-Tucker-Zemlin formulations for asymmetric traveling salesman problems. Operations Research 50:656–669

Shima T, Rasmussen SJ, Sparks AG, Passino KM (2006) Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms. Computers & Operations Research 33:3252–3269

Shyu SJ, Yin PY, Lin BMT (2004) An ant colony optimization algorithm for the minimum weight vertex cover problem. Annals of Operations Research 131:283–304

Shyu SJ, Lin BMT, Hsiao TS (2006) Ant colony optimization for the cell assignment problem in pcs networks. Computers & Operations Research 33(6):1713–1740

Silva JDL (2003) Metaheuristic and multiobjective approaches for space allocation. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, United Kingdom

Skiscim CC, Golden BL (1983) Optimization by simulated annealing: a preliminary computational study for the tsp. In Proceedings of the 1983 Winter Simulation Conference

Smith DK, Walters GA (2000) An evolutionary approach for finding optimal trees in undirected networks. European Journal of Operational Research 120(3):593 – 602, DOI DOI:10.1016/S0377-2217(98)00385-3

Soland RM (1974) Optimal facility location with concave costs. Operations Research 22:373–382

Solimanpur M, Vrat P, Shankar R (2004) Ant colony optimization algorithm to the inter-cell layout problem in cellular manufacturing. European Journal of Operational Research 157:592–606

Solimanpur M, Vrat P, Shankar R (2005) An ant algorithm for the single row layout problem in flexible manufacturing systems. Computers & Operations Research 32:583–598

Stützle T, Hoos H (1997) Max-min ant system and local search for the traveling salesman problem. In: IEEE International Conference On Evolutionary Computation (ICEC'97), IEEE Press, Piscataway,NJ, pp 309–314

Sun M, Aronson JE, McKeown PG, Drinka D (1998) A tabu search heuristic procedure for the fixed charge transportation problem. European Journal of Operational Research 106:441–456

Takahashi H, Matsuyama A (1980) An approximate solution for the steiner problem in graphs. Mathematica Japonica 24:573–577

Talbi EG (2002) A taxonomy of hybrid metaheuristics. Journal of Heuristics 8:541–564

Talbi EG, Roux O, Fonlupt C, Robillard D (2001) Parallel ant colonies for the quadratic assignment problem. Future Generation Computer Systems 17:441–449

Tavares J, Pereira F (2011) Evolving strategies for updating pheromone trails: A case study with the tsp. In: Schaefer R, Cotta C, Kolodziej J, Rudolph G (eds) Parallel Problem Solving from Nature - PPSN XI, Lecture Notes in Computer Science, vol 6239, Springer Berlin-Heidelberg, pp 523–532

Thiel J, Voß S (1994) Some experiences on solving multiconstraint zero-one knapsack problems with genetic algorithms. INFOR 32(4):226–242

Thompson J, Dowsland K (1996) Variants of simulated annealing for the examination timetabling problem. Annals of Operations Research 63:105–128

Tuy H (2000) The minimum concave-cost network flow problem with a fixed number of nonlinear arc costs: complexity and approximations. In: Pardalos PM (ed) Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems, Kluwer Academic Publishers, pp 383–402

Tuy H, Ghannadan S, Migdalas A, Värbrand P (1995a) The minimum concave cost flow problem with fixed numbers of nonlinear arc costs and sources. Journal of Global Optimization 6:135–151

Tuy H, Ghannadan S, Migdalas A, VÄrbrand P (1995b) Strongly polynomial algorithm for two special minimum concave cost network flow problems. Optimization 32(1):23–43

Uchoa E, de Aragão MP, Ribeiro CC (2001) Dual heuristics on the exact solution of large steiner problems. Electronic Notes in Discrete Mathematics 7:150–153

Varela RV, Puente CR, Gomez AJ, Vidal AM (2001) Solving jobshop scheduling problems by means of genetic algorithms. In: Chambers L (ed) The Practical Handbook of Genetic Algorithms Applications, CRC Press, Inc., pp 275–294

Vasquez M, Hao J (2001) A "logic-constrained" knapsack formulation and a tabu search algorithm for the daily photograph scheduling of an earth observation stellite. Computational Optimization and Applicationss 20(2):137–157

Venables H, Moscardini A (2006) An adaptive search heuristic for the capacitated fixed charge location problem. In: Dorigo M, Gambardella L, Birattari M, Martinoli A, Poli R, Stützle T (eds) Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science, vol 4150, Springer Berlin / Heidelberg, pp 348–355

Vignaux G, Michalewicz Z (1991) A genetic algorithm for the linear transportation problem. Systems, Man and Cybernetics, IEEE Transactions on 21(2):445 –452, DOI 10.1109/21.87092

Voß S (1999) The steiner tree problem with hop constraints. Annals of Operations Research 86:321–345

Wagner HM (1958) On a class of capacitated transportation problems. Management Science 5:304–318

Wang Q, Batta R, Bhadury J, Rump CM (2003) Budget constrained location problem with opening and closing of facilities. Computers & Operations Research 30:2047–2069

Wiitala SA (1987) Discrete Mathematics: A Unified Approach, 1st edn. McGraw-Hill, Inc., New York, NY, USA

Wilson RJ (1985) Introduction to Graph Theory, 3rd edn. Longman Scientific & Technical

Wolpert D, Macready W (1995) No free lunch theorems for search. Tech. Rep. SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM, USA

Wolpert D, Macready W (1997) No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1):67–82

Woolston K, Albin S (1988) The design of centralized networks with reliability and availability constraints. Computers & Operations Research 15:207–217

Yang Z, Yu B, Cheng C (2007) A parallel ant colony algorithm for bus network optimization. Computer-Aided Civil and Infrastructure Engineering 22:44–55

Yeh SH, Yeh HY, Soo VW (2012) A network flow approach to predict drug targets from microarray data, disease genes and interactome network - case study on prostate cancer. Journal of Clinical Bioinformatics 2

Yin PY, Wang JY (2006) Ant colony optimization for the nonlinear resource allocation problem. Applied Mathematics and Computation 174:1438–1453

Zangwill W (1968) Minimum concave cost flows in certain networks. Management Science 14:429–450

Zhang D, Liu Y, M'Hallah R, Leung SC (2010) A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems. European Journal of Operational Research 203(3):550 – 558

Zhou G, Gen M, Wu T (1996) A new approach to the degree-constrained minimum spanning tree problem using genetic algorithm. In: Systems, Man, and Cybernetics, 1996., IEEE International Conference on, vol 4, pp 2683–2688 vol.4