Journal of Heuristics manuscript No.
(will be inserted by the editor)

Concave Minimum Cost Network Flow Problems Solved with a Colony of Ants

Marta S.R. Monteiro · Dalila B.M.M. Fontes · Fernando A.C.C. Fontes

Received: date / Accepted: date

Abstract In this work we address the Single-Source Uncapacitated Minimum Cost Network Flow Problem with concave cost functions. This problem is NP-hard, therefore we propose a hybrid heuristic to solve it. Our goal is not only to apply an Ant Colony Optimization (ACO) algorithm to such a problem, but also to provide an insight on the behaviour of the parameters in the performance of the algorithm. The performance of the ACO algorithm is improved with the hybridization of a local search procedure. The core ACO procedure is used to mainly deal with the exploration of the search space, while the Local Search is incorporated to further cope with the exploitation of the best solutions found. The method we have developed has proven to be very efficient while solving both small and large size problem instances. The problems we have used to test the algorithm were previously solved by other authors using other population based heuristics. Our algorithm was able to improve upon some of their results in terms of solution quality, proving that the HACO algorithm is a very good alternative approach to solve these problems. In addition, our algorithm is substantially faster at achieving these improved solutions. Furthermore, the magnitude of the reduction of the computational requirements grows with problem size.

This work is funded by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, projects PTDC/EGE-GES/099741/2008 and PTDC/EEA-CRO/116014/2009.

Marta S. R. Monteiro
Faculdade de Economia and LIAAD-INESC Porto L.A., Universidade do Porto, Tel.: +351-22-0426240, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal, E-mail: martam@fep.up.pt

Dalila B. M. M. Fontes
Faculdade de Economia and LIAAD-INESC Porto L.A., Universidade do Porto, Tel.: +351-22-0426240, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal, E-mail: fontes@fep.up.pt

Fernando A. C. C. Fontes
Faculdade de Engenharia and ISR-Porto, Universidade do Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal, E-mail: faf@fe.up.pt

Keywords Ant Colony Optimization · Concave Costs · Hybrid · Local Search · Network Flow

Mathematics Subject Classification (2000) MSC 90C27 · MSC 90C59

1 Introduction

The Minimum Cost Network Flow Problem (MCNFP) 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. Generally speaking, in this problem the objective is to distribute some commodity from the sources to the demand nodes while minimizing the total cost incurred. Thus, many practical applications exist for the MCNFP and in many different fields, for instance in supply chains, logistics, production planning, communications, facility location, and transportation, just to mention but a few (Ahuja et al, 1995; Geunes and Pardalos, 2005).

MCNFPs with linear costs are solvable in polynomial time: they are considered easy to solve. In this work, we consider nonlinear costs comprising fixed costs as well as variable costs, associated with the flow in each arc. Regarding the variable costs, we consider both linear and nonlinear concave costs. As such, the total cost incurred when using an arc is always nonlinear and concave. Concave costs are, in many applications, more realistic than linear ones because of the association of prices with economies of scale. 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 with its full capacity or not. Therefore, the cost per unit transported will necessarily decrease with the increase on the number of transported items. Initial costs incurred with facilities and equipment are another example leading to concave costs. When concave costs are introduced in MCNFPs, the difficulty to solve them increases and they become NP-Hard (Guisewite and Pardalos, 1991). The complexity arises from the fact that in the minimization of a concave function (even over a convex feasible region) a local optimum is not necessarily a global optimum. The special case of the Single-Source Uncapacitated Minimum Cost Network Flow Problem (SSU MCNFP) with fixed-charge costs has also been proven to be NP-Hard by Hochbaum and Segev (1989). Kennington and Unger (1976), Barr et al (1981), and Palekar et al (1990) discuss the difficulty of MCNFPs regarding the case of fixed-charge cost functions. One of the main attractiveness of concave MCNFPs is that any Network Flow Problem (NFP) with general nonlinear costs can be transformed into a concave NFP in an expanded network (Lamar, 1993), and also capacitated and multiple sources can be transformed into uncapacitated and single-source NFPs (Wagner, 1958; Zangwill, 1968). Thus, SSU MCNFPs have a major utility.

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 NP-Hard Travelling Salesman

Problem, where the shortest circular route between a given set of nodes is to be defined, without visiting any node twice. 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 SSU MCNFP can be viewed as the problem of finding shortest paths (i.e. least cost paths) between a source node and every single demand node in a network having into account the flow to be routed between pairs of nodes. Therefore, we believe that ACO will also have a good performance while solving the SSU concave MCNFP, since this problem joins together these three characteristics: concave costs, shortest paths, and flow between pairs of nodes. Moreover, ACO algorithms are inspired on the natural behaviour of ants while searching for the best paths between the nest and food sources. The similarities between nature ants problem and the MCNFP are easy to find, which can be another motivation to expect ACO to perform well for this particular problem.

SSU MCNFPs have already been solved with Genetic Algorithms (GAs) with very good results (Fontes and Gonçalves, 2007), and GAs are known to have very good performance in solving combinatorial optimization 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 starting this work we expected ACO to have a competitive performance, in comparison with other heuristic methods already used to solve this problem. Our expectations were confirmed by the results.

In this work we propose an Ant Colony Optimization algorithm hybridized with a local search procedure to solve Single-Source Uncapacitated Minimum Cost Network Flow Problems with concave cost functions. The cost functions have a fixed and a variable component and this latter component might be linear or nonlinear. In order to further improve the best solution found at each iteration, a Local Search (LS) procedure, based on swap moves, is incorporated into the ACO. The results show the effectiveness and efficiency of our ACO algorithm for both small and large size problems, proving ACO to be an alternative solution method for MCNFPs with nonlinear concave costs. This work is an extension of a preliminary work presented at GECCO (Monteiro et al, 2011).

Our contribution is twofold. On the one hand, the application of an ACO based algorithm to solve MCNFPs is, to the best of our knowledge, here proposed for the first time. We also provide a study on the variation of the ACO parameters values and their influence on the solution quality. On the other hand, we are able to improve previous results that have been obtained with other population based heuristics, by substantially 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 to the optimum or the best solutions found so far. In fact, HACO is able to reduce

up to 10 times the computational time of the HGA and up to 5.7 times the computational time of the commercial optimization software CPLEX. Furthermore, the larger reductions are achieved for larger size problems.

The remainder of this paper is organized as follows. In Section 2, we give a formal description of the SSU concave MCNFP along with its mathematical formulation. A literature review on MCNFPs is provided in Section 3. In Section 4, we review some work on ACO. In Section 5 we develop our approach to solve the SSU concave MCNFP. The results obtained and the subsequent analysis are reported and discussed in Section 6. Finally, Section 7 provides some conclusions of what has been done and discusses future work.

2 Problem definition and mathematical formulation

In the SSU MCNFP the objective is to find a network defined on a given graph, as well as the flows to be routed through the chosen arcs. The commodity flows from a single source to a set of demand nodes, such that all demand nodes are satisfied and costs are minimized. As we are considering the uncapacitated version of the problem, there are no upper or lower 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.

The mathematical model for the SSU MCNFP can then be written as follows:

$$\text{min:} \quad \sum_{(i,j)\in A} g_{ij}(x_{ij}, y_{ij}) \tag{1}$$

$$\text{s.t.:} \quad \sum_{\{i|(i,j)\in A\}} x_{ij} - \sum_{\{k|(j,k)\in A\}} x_{jk} = d_j, \quad \forall_{j\in N\setminus\{t\}}, \tag{2}$$

$$x_{ij} \leq M y_{ij}, \quad \forall_{(i,j)\in A}, \tag{3}$$

$$x_{ij} \geq 0, \quad \forall_{(i,j)\in A}, \tag{4}$$

$$y_{ij} \in \{0, 1\}, \quad \forall_{(i,j)\in A}. \tag{5}$$

The objective in this problem is to minimize the total cost incurred with the network, as given in Eq. (1). Constraints (2) 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 (3) 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) and (5) 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, that is,

$$\sum_{j\in N} d_j = 0. \tag{6}$$

2.1 Cost Functions

In this work, three types of polynomial concave cost functions are considered:

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

$$g_{ij}(x_{ij}, y_{ij}) = \begin{cases} b_{ij} \cdot x_{ij} + c_{ij}, & \text{if } y_{ij} = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

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

$$g_{ij}(x_{ij}, y_{ij}) = \begin{cases} -a_{ij} \cdot x_{ij}^2 + b_{ij} \cdot x_{ij}, & \text{if } y_{ij} = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

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

$$g_{ij}(x_{ij}, y_{ij}) = \begin{cases} -a_{ij} \cdot x_{ij}^2 + b_{ij} \cdot x_{ij} + c_{ij}, & \text{if } y_{ij} = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

where a_{ij}, b_{ij}, and $c_{ij} \in \mathbb{Z}^+$. The use of polynomial cost functions is motivated by the easiness of approximation of any analytic function by the first few terms of a Taylor series.

3 Literature Review on MCNFPs

The works developed around concave MCNFPs can be differentiated regarding the type of concave cost function used.

Some works consider SSU MCNFPs with nonlinear concave routing costs that do not include a fixed component. Among exact methods to solve such concave MCNFPs we can find branch-and-bound (BB) and Dynamic Programming (DP). Both methods usually start with a main problem dividing it into smaller subproblems which, in turn, are further divided into smaller subproblems, and so on. These methods are commonly used to find upper bounds for the solution cost, thus reducing the number of extreme points to be searched for. In Gallo et al (1980) a BB algorithm is developed to solve SSU concave MCNFPs. The branching part of the algorithm is performed by adding arcs extending the current subtree. Then, lower bounds are obtained by using linear underestimation. Guisewite and Pardalos (1991) improve these lower bounds by projecting them on the cost of extending the current path. Horst and Thoai (1998) consider the capacitated version of concave MCNFPs. A branch-and-bound algorithm based on the work of Soland (1974) has been developed. This BB uses linear underestimation by convex envelopes, where rectangles are defined by the capacity flow constraints to partition the search space, to improve the lower bounds.

Other works solving concave MCNFPs consider costs with both a fixed-charge and a linear routing component. 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. The idea behind it is to find a linear factor that can represent the variable and fixed costs at the same time. Then, at each iteration the cost function is updated by using the information of the solution found in the previous iteration. Latter on, Kim and Pardalos (2000) extended the use of DSS by joining it with Trust Interval Techniques to solve concave piecewise linear NFPs. A recent work on MCNFPs is the one of Nahapetyan and Pardalos (2008), 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 and it is then solved with a dynamic slope scaling method, based on the one proposed by (Kim and Pardalos, 1999). Rebennack et al (2009) propose a continuous bilinear formulation for the fixed-charge network flow problem from which an exact algorithm is derived, based on these two previous works. Ortega and Wolsey (2003) also provide an exact algorithm using a branch-and-cut method by extending the cutting planes previously used to solve uncapacitated lot sizing problems.

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. Burkard et al (2001) consider concave cost functions on the arcs of uncapacitated acyclic single-source networks. 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. Upper bounds based on

local search are obtained in (Fontes et al, 2003) to solve MCNFPs, while lower bounds derived from state space relaxations are given in (Fontes et al, 2006b). These lower bounds are used in the bounding phase of a branch-and-bound procedure that is developed in (Fontes et al, 2006a) in order to solve SSU concave MCNFPs. Fontes et al (2006c) use DP to optimally solve MCNFPs. 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.

Exact methods can be very expensive in terms of computational effort, even for small sized problems. As heuristic methods can easily overcome this problem they have become very popular in recent years, although they may provide only a local optimum. 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. Randomly generated feasible trees are considered for the initial population and the mutation operator is defined so that extra arcs are added to the tree in such a way as to maintain its feasibility. The problems solved by this GA have concave flow costs given by the square root of the flow, and their sizes range from 15 to 25 nodes. However, no results are reported regarding the computational time spent by the algorithm nor regarding the quality of the solution. Fontes et al (2003) propose a local search procedure to solve MCNFPs that uses information about the structure of the network obtained with a lower bound solution derived from a state space relaxation. 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.

Fontes and Gonçalves (2007) use a genetic algorithm coupled with a local search procedure 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. The local search procedure operates through swap operations between arcs already in the solution tree and arcs not in the solution. Arcs (i, j) belonging to the solution tree are sorted and considered in descending order of nodes priority, given by the random keys. Then each arc (k, j) outside the solution tree, is considered in descending order of node k 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). More recently, Dang et al (2011) have 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. Two cost functions are considered, a linear cost function without a fixed-charge component, and a second-order concave polynomial function. 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. 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 are dealt with by using Lagrangean Multipliers. Computational results are

provided on networks with 5 up to 12 nodes.

4 Ant Colony Optimization

ACO algorithms were initially developed based on the natural behaviour of ants while searching for a path between their nest and some food source. Although their goal may not be to find the shortest path, in the end, and with their behaviour, that is exactly what they end up achieving, and that is what it makes them so attractive to optimization research investigators. The first experiences with real ants unravelling the reasons for ants to choose between paths were made by Goss et al (1989) and by Deneubourg et al (1990). They discovered that ants, while travelling several times between their nest and a food source, deposit in the path a chemical substance called pheromone[1]. If a path has a large concentration of pheromone, this is probably due to its shorter length that allows ants to travel faster, resulting in a larger number of travellers and thus of ants depositing pheromone throughout the path. When an ant is faced with the decision to choose between paths to follow to reach the food source, the ant will choose with higher probability the path with the largest pheromone concentration. It was the observation of this sort of communication developed by the ants that inspired Dorigo et al (1996) to develop the first ACO algorithm which was called Ant System (AS). The AS was originally used to solve the Travelling Salesman Problem (TSP), a well known NP-Hard problem. The ant system has two main phases, the construction of the solution and the pheromone update. Nevertheless, other decisions have to be made before the ants can start finding a solution, such as defining the structure of the solution, deciding on the number of ants to use and on the initial pheromone quantity to spread in each path.

From the results of the experiences that took place following the definition of the AS (for example the definition of the elitist AS, the Rank-based AS, or the Ant Colony System), the Ant System structure was improved, leading to one of the most important developments that followed which was the description of the Ant Colony Optimization metaheuristic by Dorigo and Di Caro (1999). The main difference from the basic structure of the AS algorithm is the introduction of an optional *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 nature. The daemon, having an overall knowledge of what is happening at each stage of the algorithm, can perform tasks that no individual ant can do. For example, it can control the feasibility of each solution by evaporating a percentage of the pheromone quantity in its arcs, as a way of penalizing such a solution, or decide to completely disregard the solution. Most commonly it is used to identify the

[1] This behaviour is shared by several animal species and even by some plants.

best solution ever and the best solution of the current iteration, which are the only ones that are usually allowed to deposit pheromone in their solution, as opposing to the AS where all ants deposited pheromone. This is done mainly to prevent premature convergence of the algorithm.

In an ant algorithm, ants move on the network by going from one node to another. The one where it moves to is probabilistically chosen based on the pheromone quantities deposited on the arcs outgoing from the current node. After the ants have constructed their respective solutions, the pheromone trails are updated. The update is done in two ways: on the one hand pheromone values are decreased through evaporation, that is its values are decreased by a constant decay; on the other hand, pheromone values are increased in the parts of the network which are present in the best solution(s). Such increase is proportional to the solution(s) quality. The process of solution construction and pheromone updating is repeated until some stopping criterion, initially defined, has been reached.

A major feature, commonly used by authors in ant algorithms that were developed ever since, is the introduction of a Local Search procedure following the construction of the solutions by the ants and just before pheromone update. This is an optional feature but it has been proved to be very important in the exploitation of the search space nearby good solutions, leading almost always to better performances.

Although initially applied to solve the TSP, ACO algorithms have become very popular and have been applied to solve a broad set of combinatorial optimization problems, mainly due to their versatility and easiness of adaptation.

Rappos and Hadjiconstantinou (2004) use ACO to solve two-edge connected network flow design problems making use of flow ants to construct the network and of reliability ants to deal with the reliability of the network. While solving degree-constrained Minimum Spanning Trees, Bui and Zrncic (2006) define maximum and minimum allowed pheromone values, based on the differences between the cost of the most expensive and of the least expensive arcs. Reimann and Laumanns (2006) use saving values as the heuristic information in an ACO algorithm developed to solve Capacitated Minimum Spanning Tree problems. Many other problems have also been solved with ACO algorithms. Shyu et al (2006) define two heuristic information matrices to solve the Cell Assignment Problem. These are associated with the choice of to which switch to move to when located at a certain cell and vice-versa. Two colonies are used by Chen and Ting (2008) to solve Single Source Capacitated Facility Location Problems, one to find the location of facilities and the other to assign customers to locations. Lessing et al (2004) studied the influence of the heuristic information, in the performance of ant algorithms when solving Set Covering Problems. The Terminal Assignment problem is solved in (Bernardino et al, 2009) by means of an ACO algorithm, with a Local Search procedure embedded in it, that uses the pheromone quantity laid in each path to modify the solutions obtained previously. Other areas of research have also been using ACO algorithms such as Image Processing (Meshoul and Batouche, 2002), Data Mining (Parpinelli et al, 2002), Protein Folding (Hu et al, 2008),

Power Electronic Circuit Design (Zhang et al, 2009), Grid Workflow Scheduling Problem (Chen and Zhang, 2009), just to mention but a few.

In the past few years authors have also developed hybrid algorithms between ACO algorithms and other Metaheuristic methods. Simulated Annealing (SA) and an Ant Colony System (ACS) are joined together in (Bouhafs et al, 2006) to solve Capacitated Location-Routing problems. The SA component of the algorithm is used to locate the distribution centres (DC) and assigning customers to each DC, while the best routes are defined by the ACS. Crawford and Castro (2006) solve both Set Covering and Set Partitioning benchmark problems with ACO algorithms, and with hybridizations between ACO and Constraint Programming techniques, Forward Checking, Full Lookahead, Arc Consistency, and Post Processing procedures. In (Altiparmak and Karaoglan, 2007) the authors hybridize ACO with a Genetic Algorithm to solve Transportation Problems with square root concave costs. They introduce a twofold mechanism to identify whenever the algorithm has stagnated. All the pheromones are set to a maximum value whenever more than 50% of the arcs in the network have reached the minimum value allowed for the pheromones. Also, whenever the global best solution has not been updated for 50 iterations 10% of the worst chromosomes of the population are replaced with randomly generated ones.

Many more works could be cited, but it would be beyond the objective of this work. For the interested reader, besides the works already mentioned here and the references therein, (Cordon et al, 2002; García-Martínez et al, 2007), and (Mullen et al, 2009) provide excellent surveys on ant colony algorithms and applications.

Although a preliminary version of this study was published and presented at GECCO (Monteiro et al, 2011), as far as the authors are aware of, the SSU concave MCNFP has not yet been solved by an ACO algorithm. Next, we will describe the ACO approach we have developed and implemented to solve the SSU concave MCNFP.

5 Hybrid Ant colony optimization approach for the SSU concave MCNFP

Ant colony algorithms have a set of characterising features that can be considered as their step stones or building blocks. These characteristics should be specified when describing an ant algorithm, so that it can be differentiated from other algorithms. These characteristics are:

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

In the following sections we describe the algorithm that we have developed, following this list.

5.1 Defining and constructing a 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 taken is the representation of the solution to the problem being solved, because a poor representation can lead to poor solutions.

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). SSU MCNFPs have a finite solution if and only if there exists a direct path going from the source node to every demand node and if there are no negative cost cycles; otherwise an unbounded negative cost solution would exist. 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). 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. So, each ant solution consists on a number of directed arcs that equals the number of demand nodes.

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 node t and one of the demand nodes $j \in N \setminus \{t\}$. Consider a network with four demand nodes $\{1, 2, 3, 4\}$ and assume that the first arc entering the solution is arc $(t, 2)$. Then, the ant selects another arc, from the set of available arcs linking the source node or one of the demand nodes already in the partial solution to another demand node not yet considered. In the example given, the set of arcs that could be chosen to enter the solution, assuming a fully connected network, would be $\{(t, 1), (t, 3), (t, 4), (2, 1), (2, 3), (2, 4)\}$. This last step is repeatedly performed until all demand nodes are in the solution. Therefore the feasibility of the solution is guaranteed. The arc entering the solution is chosen by using the probability function defined in Eq. (10).

$$P_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{(i,j)\in A} [\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}, \tag{10}$$

where τ_{ij} is the pheromone value associated to arc (i, j) at the current iteration; η_{ij} is the visibility (heuristic information) of arc (i, j), which is a value defined and calculated only once at the beginning of the algorithm, and in our case is given by $\eta_{ij} = \frac{1}{c_{ij}+b_{ij}}$; α and β are both positive parameters weighting the relative importance of the pheromone value and the heuristic information, respectively, in the choice of a new arc entering the solution.

5.2 Updating Pheromones

After all ants have finished the construction of their solutions for the current iteration, the best solution is identified, and the algorithm proceeds to the update of the pheromone values. The pheromone update is performed according to the update function given in Eq. (11).

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \Delta\tau_{ij}. \tag{11}$$

The pheromone values are reduced in every existing arc $(i, j) \in A$, which is an attempt at simulating the natural process of evaporation, and is represented by the first component of Eq. (11), where ρ represents the pheromone evaporation rate, with $\rho \in]0, 1]$. The value of the evaporation rate indicates the relative importance given to the pheromone values from one iteration to the following one. A small value of ρ will increase the importance of the arcs throughout a large number of iterations or period of time, whereas if ρ takes a value near to 1 the pheromone trail will not have a lasting effect. The second component of Eq. (11) represents the pheromone quantity to be deposited in arc (i, j), at the current iteration. The pheromone quantity $\Delta\tau_{ij}$ to be deposited in arc (i, j) depends on $g(S^i)$, i.e., on the cost of the best solution S^i found at the current iteration, and on a positive proportionality parameter Q, and is given by:

$$\Delta\tau_{ij} = \begin{cases} \frac{Q}{G(S^i)} & \text{if } (i, j) \text{ belongs to solution } S^i, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Note that, although the evaporation process is applied to all available arcs $(i, j) \in A$, only the n arcs belonging to the best solution found at the current iteration will have an increase on their pheromone values. This means that, although for small problem instances the pheromone evaporation rate may take a small part, its role increases with problem size, due to the increase of the feasible solution space to be explored. In this latter case, a small value for ρ allows the algorithm to retain old solutions in memory a lot longer. Thus, potentiating the use of good arcs in the following iterations in an attempt to search nearby good solutions. Given the form of Eq. (12), it is easy to conclude that the quantity of pheromone to be deposited in each arc depends on the cost of the solution.

5.2.1 Pheromone bounds

Stützle and Hoos (1997) have made an important contribution for the ant colony optimization research area as they were able to avoid the fast convergence of the pheromone trail, which usually induces stagnation of the algorithm. Typically, if one or a few solution components have more pheromone than the others, ants tend to include those components into their solutions, thus generating over and over again the same solution. Stützle and Hoos, by

proving the convergence of the pheromone trail, in certain conditions, developed lower and upper bounds for the pheromone trail. This approach has been found helpful in other works, such as in Rappos and Hadjiconstantinou (2004), Venables and Moscardini (2006), and Altiparmak and Karaoglan (2007). We now describe the method that is used in this work.

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, as can be seen in Eq. (13).

$$\tau_{ij} = \tau_0, \forall (i, j) \in A. \tag{13}$$

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. (14).

$$\tau_{max} = \frac{1}{\rho \cdot G^*}. \tag{14}$$

The τ_{min} value depends on the upper bound for the pheromone value τ_{max} and on the probability of constructing the best solution, hereby represented by parameter p_{best}, as given in Eq. (15).

$$\tau_{min} = \frac{\tau_{max} \cdot (1 - \sqrt[n]{p_{best}})}{(\frac{n}{2} - 1) \cdot \sqrt[n]{p_{best}}}. \tag{15}$$

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

5.3 Algorithm

In Algorithm 1 we provide an outline of the ACO algorithm that we have developed to solve the SSU MCNFP with concave costs. 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 5.1. 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 Sec 5.4, 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.

Algorithm 1 Pseudo-code of the proposed ACO algorithm

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 $W = \{S^i \cup 4 \text{ randomly chosen } S_a\}$
  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 $G(S^g)$

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, that is, it is set to τ_{max} in case it exceeds it, or it is set to τ_{min} if its value is lower than τ_{min}. The algorithm runs until the maximum number of iterations allowed $maxIter$ is reached.

5.4 Local Search

Although the first experimental results in the test set achieved good solutions, as it can be seen in Section 6.2, 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 1), 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 MCNF problem, 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 the time from S, in ascending order of 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 2.

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 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.

Algorithm 2 Pseudo-code for the Local Search procedure that was incorporated into the ACO algorithm developed

```
1: W = {S^i ∪ 4 randomly chosen S_a}
2: for all S ∈ W do
3:    Sort all (i,j) ∈ S in ascending order of τ_ij
4:    for all arcs (i,j) ∈ S do
5:       Identify P as the set of all arcs (k,j) ∉ S that can replace (i,j) ∈ S without
         forming a cycle
6:       Sort (k,j) ∈ P in descending order of τ_kj
7:       for each arc (k,j) ∈ P do
8:          S' = S\{(i,j)} ∪ {(k,j)}
9:          if G(S') < G(S) then
10:            S ← S'
11:            GOTO next (i,j) ∈ 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
```

5.5 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 HACO parameters[2] 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. 10, respectively. Fig. 1 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. 5.1). Matrix (B) is the pheromone matrix obtained after the pheromone values update (as given in Sec. 5.2), and matrix (C) provides the pheromone values after all pheromone bounds violations of matrix (B) are corrected (see Sec. 5.2.1). The updating of pheromone matrices (B) and (C) is carried out towards the end of the iteration and will be explained later.

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 2 is applied to each solution in W[3]. We will use Fig. 2 to illustrate the evolution of a solution $S \in W$ with the application of local search.

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. 2 (a). Note

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

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

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. 1).

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. 2 (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. 2 (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. 2 (d), and continue with the local search procedure considering this 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. 2 (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. 2 (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).

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

Now, we return again to Algorithm 1. The algorithm identifies the best of

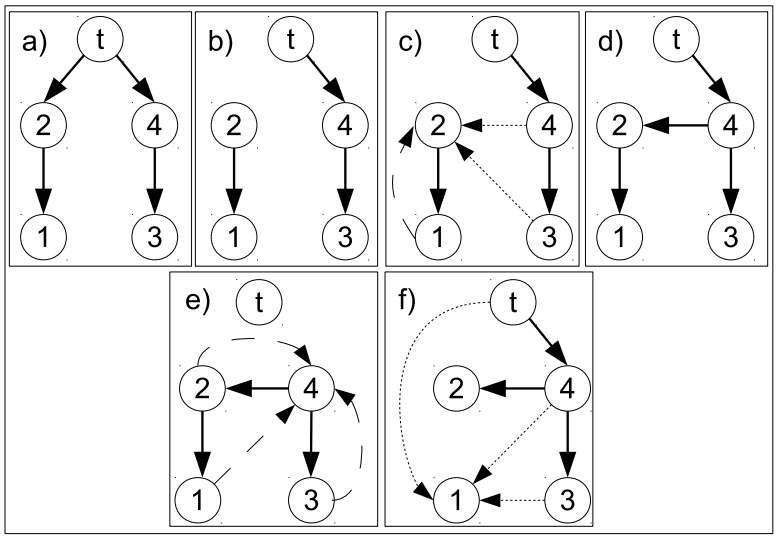

Fig. 2 Example of the Local Search procedure

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 5.2.1).

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. 1.

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. 1, 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. 1.

The HACO 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.

6 Computational Experiments

In this section we report on the computational results obtained with the heuristic described in the previous section. We also report literature results for the same problems in order to compare the performance and effectiveness of our algorithms. In order to evaluate an heuristic algorithm, it is not always clear, a priori, what must be tested. The work of Rardin and Uzsoy (2001) provides valuable guidelines that were adopted here.

The performance of the heuristics is evaluated by using two measures, Time (in seconds) required to perform a full run of the algorithm, and % Gap. The Gap is calculated by comparing two solutions, and is given by:

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

where OptS stands for the optimal solution, and the HS stands for the solution found with the heuristic in question.

The algorithms proposed in this paper, the ACO and the ACO+LS which we have named HACO, were implemented in Java and the computational experiments were carried out in a PC with an Intel Core 2 processor at 2.4 GHz with 4MB of RAM. CPLEX was run in the same computer. The computer used is the same of that of the HGA, with which we compare our results.

6.1 Test Problems

In order to test the algorithm that was developed to solve SSU concave MCNFPs we downloaded the Euclidean test set of problems available from (Beasley, 2010). 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 by Hochbaum and Segev (1989) that the values of such ratios are the main parameter in defining problem difficulty for fixed-charge MCNFPs. Regarding this ratio in the literature we find two distinct opinions. 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 Hochbaum and Segev (1989) and Palekar et al (1990) suggest 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 problems we have solved have V/F ratios ranging from 0.01 to 10.

Another important difficulty parameter is the number of arcs with concave arc costs. For example, SSU networks with a single nonlinear concave arc cost have been proven to be solvable in polynomial time (Guisewite and Pardalos, 1993) and (Klinz and Tuy, 1993). Furthermore, Tuy (2000) proves the strong polynomial-time solvability of SSU MCNFPs with a fixed number of concave cost arcs. It is important to stress out that every single arc in the problems

we solve has associated a concave cost and that Guisewite (1995) has demonstrated that finding a strict local optimum for the SSU concave MCNFP is NP-hard. Thus, we are considering problems with proven complexity. Furthermore, in the work by Horst and Thoai (1998), a concave cost is associated with a subset of existing arcs, while the rest of the arcs have linear costs, and the results empirically obtained, by using a branch-and-bound procedure, show evidence of an increasing difficulty to solve problems with the same size but with an increasing number of arcs with concave costs. In addition, 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.

For each of the 10 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 ten groups are available, while for the remaining problems only the first five groups are available (For more details on these problems please refer to Fontes et al (2003)). Therefore, there is a total of 240 problem instances to be solved. Since three types of cost functions are considered, see Section 2.1, the number of problem instances to be solved increases to 720. Each problem instance is solved five times and the average results obtained are reported and discussed in the following sections.

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, and for which we report on the CPLEX 9.0 solution regarding cost function of Type I. (Problems with cost functions of Type II and III cannot be solved by CPLEX) Furthermore, cost function of Type III in addition to the nonlinear concave cost also incorporates a fixed-charge component. 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)).

6.2 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 ACO algorithm that does not consider local search, thus the algorithm is hereby solely 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. Problems with 10 nodes from group 10, problems with 25 nodes from group 7, and problems with 50 nodes from group 3 have

been taken. In total we have used 9 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 the previous section.

6.2.1 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 $\eta_{ij} = \frac{1}{b_{ij}+c_{ij}}$ for cost function Types I and III, and $\eta_{ij} = \frac{1}{b_{ij}}$ 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 function is disregarded because its main objective is to define the concavity of the cost function.

6.2.2 Setting α and β parameter values

The values α and β, appearing in the probability function defined in Eq. (10), 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 closer to a 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.

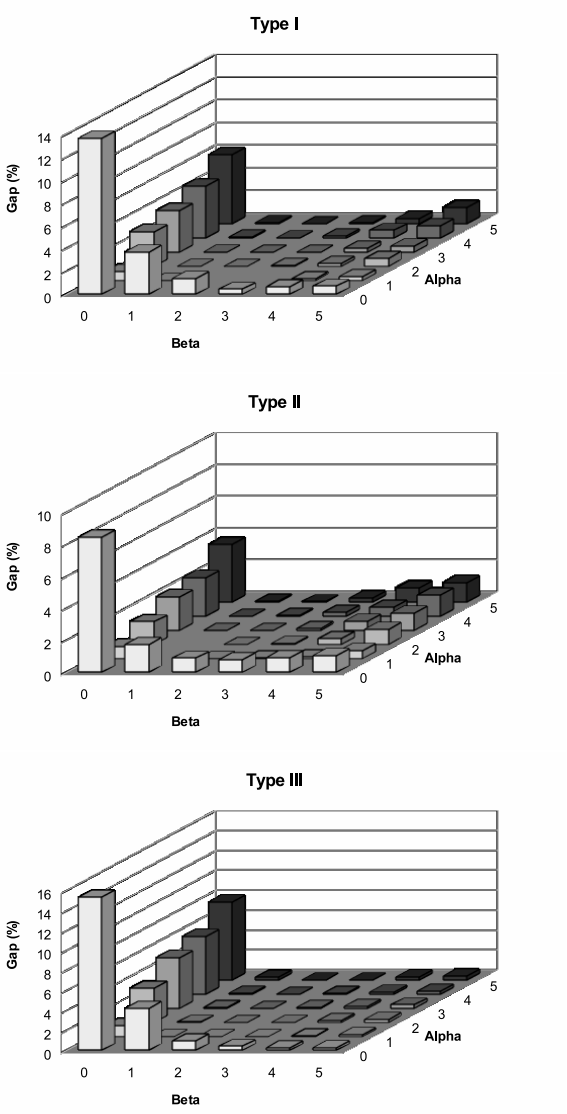

Fig. 3 Graphical representation of average gaps obtained while varying both α and β parameter values within {0,1,2,3,4,5}, and considering Type I, Type II, and Type III cost functions

As the impact of these two parameters are so connected, we have made an intensive study extending the values reported in the literature, on the combination of values for the two of them. In Fig. 3 we present the results obtained for the gap, for each cost function, while varying both α and β values within {0,1,2,3,4,5}.

It becomes clear, from the observation of Fig. 3, 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 for each of the three cost functions considered, leading to a maximum average error of 15.31%. This behaviour is somehow expected because only by chance the first *guesses* of the algorithm are good solutions, and a lower gap is achieved. In the case of $\alpha = 0$ the probability function ignores the pheromone values and only the heuristic value is considered to choose arcs into the solution. For 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 a good solution. 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. 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 for the conclusion that the heuristic information is of capital importance for this problem and it must not be ignored.

The gap seems to concentrate on 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, leading the algorithm to the optimum value. In this case, the values corresponding to the best gap averages are $\alpha = 1$ and $\beta = 2$.

6.2.3 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 a local optimum. 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 Fig. 4. 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 solutions 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 that small values for ρ, that 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

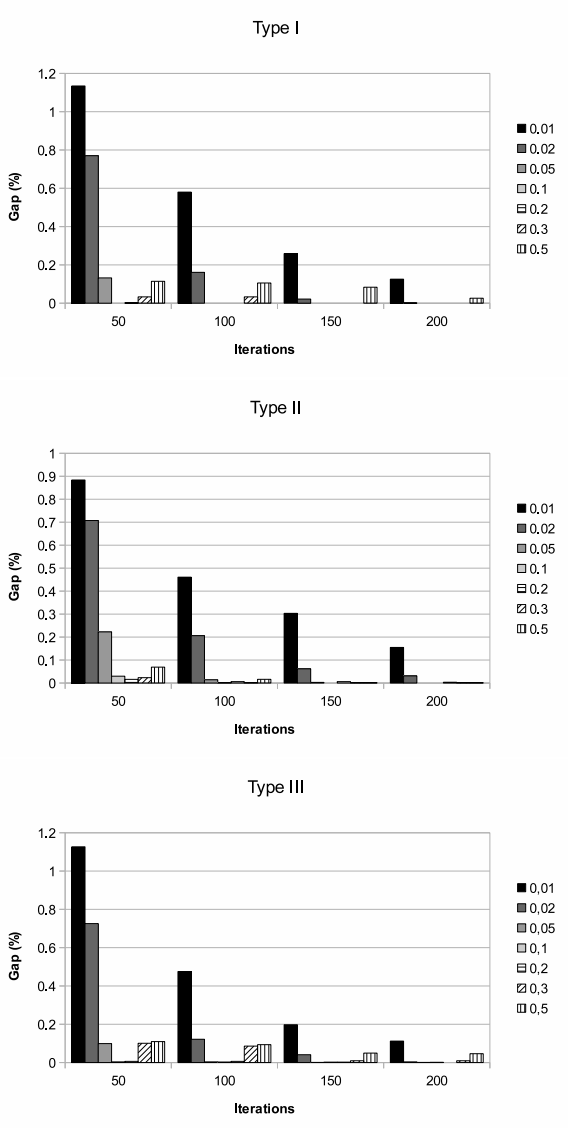

Fig. 4 Average gaps obtained while varying the pheromone evaporation rate ρ within {0.01,0.02,0.05,0.1,0.2,0.3,0.5}, and considering Type I, Type II, and Type III cost functions

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 evaporation rates of 10% and 20%. The algorithm, although presenting very small gaps, cannot find an optimal solution to all problems.

6.2.4 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. According 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 consistently giving 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.

6.2.5 Final Parameters

The results reported above were obtained only with the training set. We have proceeded with our experiences with the whole set of problems having used the final parameter values that can be seen in Table 1, which were the ones with the best average results for the training set.

Table 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
no. of ants	n
no. of iterations	200

6.3 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 cost function Type I 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 to 19 nodes the Dynamic Programming algorithm reported in (Fontes et al, 2006c) provides an optimal solution. For problems with 25 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 HACO (ACO with local search) and with the ACO algorithm. A value below 100 denotes an improvement brought in by incorporating local search, otherwise we may conclude that no improvement was achieved. Table 2 summarizes the average ratios obtained by group and by problem size[4].

There are two groups that are consistently improved with the introduction of local search, they are group 1 and 6, at least for Types 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 HACO.

Now let us compare the results obtained with HACO with the ones reported in literature. 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; 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 node priorities to select the arcs to enter and to

[4] Please recall that, for problems with 40 and 50 nodes only the first 5 groups are available.

Table 2 Average ratios between HACO and ACO, for each of the three cost functions considered, and classified by group and by problem size

Type I	Group									
No. of Nodes	1	2	3	4	5	6	7	8	9	10
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	**100,00**	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	Group									
No. of Nodes	1	2	3	4	5	6	7	8	9	10
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	**100,00**	**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	Group									
No. of Nodes	1	2	3	4	5	6	7	8	9	10
10	100	100	100	100	100	**99,9**	100	100	100	100
12	**99,9**	100	100	100	100	**99,6**	100	100	100	100
15	**98,7**	100	100	100	100	**99,7**	100	100	100	100
17	**99,8**	100	**99,99**	100	100	**99,9**	100	100	100	100
19	100	100	100	100	100	100	100	**99,95**	100	100
25	**99,9**	100	100	100	100	**99,9**	100	100	100	100
30	**99,9**	**99,7**	100	100	100	**99,3**	100	**99,97**	100	**99,99**
40	**99,0**	**99,9**	**99,999**	100	100					
50	**99,7**	**99,95**	**99,998**	100	100					

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 HACO 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 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 (HACO, CPLEX, and HGA) were run on the same computer.

As we can see, for problems with up to 50 nodes, HACO 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.

Table 3 Average computational results obtained for cost function Type I and grouped by problem size

	HGA		HACO		CPLEX
Size	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

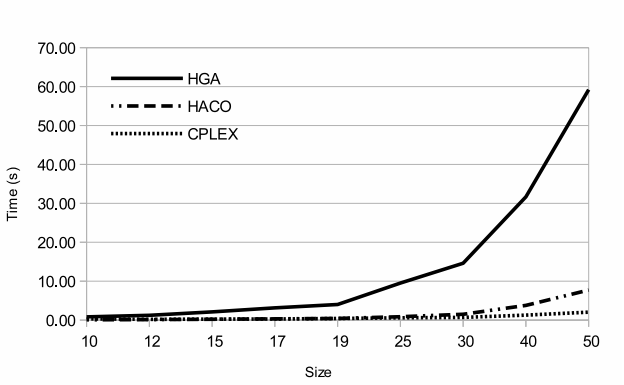

Fig. 5 Computational time results obtained for Type I cost functions, by problem sizes from 10 up to 50

Computational times (required for a full run of the algorithms) increase with problem size, as it is expected, see Figs. 5 and 6. Nonetheless, HACO times are considerably smaller than those of the HGA, HACO being up to 11 times faster. Furthermore, the rate of increase in the computational times is much larger for HGA than for HACO, 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 HACO ones. As both the HGA and the HACO 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.

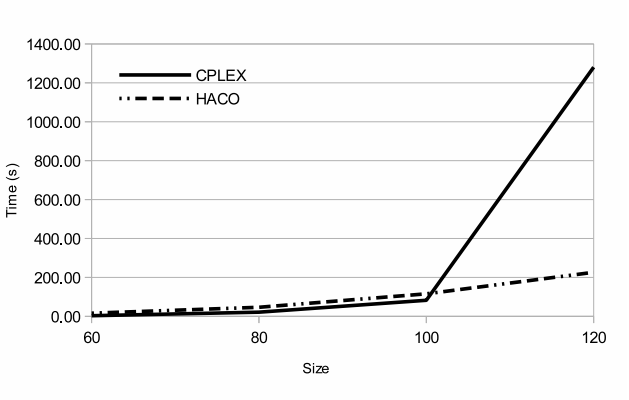

Fig. 6 Computational time results obtained for Type I cost functions, by problem size from 60 up to 120

The HACO evaluates $n \times 200$ solutions and the LS 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 LS heuristic. This means that although with similar results, see the gaps, the HACO 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.

The HACO 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 HACO algorithm is quite different, see Fig. 6. 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 HACO increases twice, while the time of CPLEX increases about 15 times.

Table 4 Average computational results for cost function Type II, grouped by problem size ranging from 10 up to 50 nodes

	HGA	HACO	
Size	Time	HACO/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

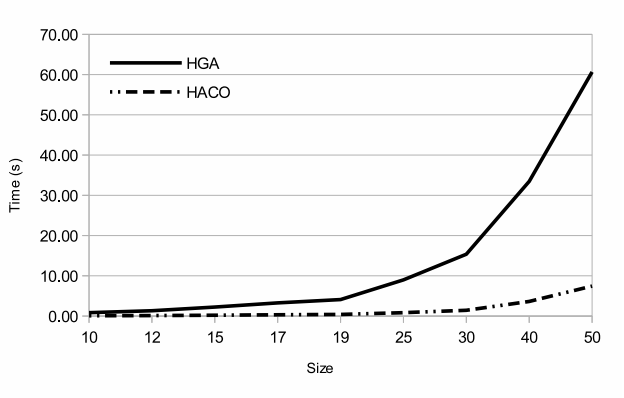

Fig. 7 Computational time results obtained for Type II cost, by problem size

In Table 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 HACO have good performances regarding the problems for which an optimal value is known, and HACO was always able to find an optimal solution. Furthermore, HACO 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, HACO is up to 11 times faster than HGA, as can be seen in Fig. 7. It should be noticed that the rate of increase of the computational time with problem size is again much smaller for the HACO. 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 5, where BS stands for the best solution known. For problems for which an optimal value is known HACO was always able to find it.

Again the computational time, see Fig. 8, is considerably smaller for HACO in relation to HGA. And again, increasing needs of computational times grow much faster for the HGA.

7 Conclusions

We have described the algorithm that was developed, based on Ant Colony Optimization and on Local Search, to solve the Single-Source Uncapacitated Minimum Cost Network Flow Problem with concave cost functions. In the test problems used three cost functions are considered: a fixed-charge function and two second order polynomials, one with and another without a fixed charge component. We have solved problems ranging from 10 up to 120 nodes. We provide a study on the performance of the algorithm with the variation of

Table 5 Average computational results for cost function Type III, grouped by problem size ranging from 10 up to 50

	HGA	HACO		
Size	Time	HACO/HGA	Time	HACO/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

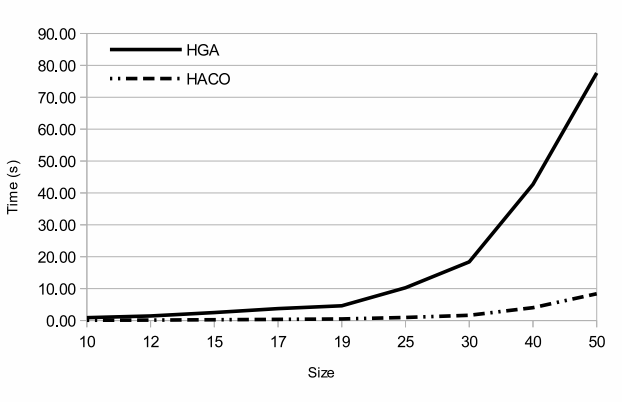

Fig. 8 Computational time results obtained for Type III cost, by problem size

the parameters values, which 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 compare our results with the ones in literature and our algorithm proved to be very efficient and effective. The solutions obtained were always as good or better than the ones obtained by an HGA presented in (Fontes and Gonçalves, 2007) even though the number of solutions evaluated by the HACO was much smaller. Nonetheless, the greatest advantage of HACO is that it requires up to 11 times less computational time than HGA. Furthermore, although the computational time requirements of the HACO also increase with problem size, the rate of increase is much smaller than that of the 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. For example, for problems with 120 nodes the CPLEX requires approximately 21 minutes to solve the problems, whereas for

the HACO 4 minutes suffice. This is very important and leads us to believe that for larger problems the HACO can be the best choice.

We have then proved HACO to be an alternative method to solve SSU concave MCNFPs, with the advantage of representing a much lower computational effort, since it only evaluates about 20% of the number of solutions evaluated by the HGA. Given the good results obtained, we are encouraged to proceed our study on ACO algorithms and their applicability to other network flow problems.

References

Afshar MH (2005) A new transition rule for ant colony optimization algorithms: application to pipe network optimization problems. Eng Optimiz 37(5):525–540

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

Altiparmak F, Karaoglan I (2007) A genetic ant colony optimization approach for concave cost transportation problems. In: IEEE Congress on Evolutionary Computation, 2007. CEC 2007., pp 1685 –1692

Barr F, Glover F, Klingman D (1981) A new optimization method for fixed charge transportation problems. Oper Res 29:448–463

Beasley J (2010) Or-library. http://www.brunel.ac.uk/deps/ma/research/jeb-/orlib/netflowccinfo.html

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

Bin Y, Zhong-Zhen Y, Baozhen Y (2009) An improved ant colony optimization for vehicle routing problem. Eur J Oper Res 196:171–176

Bouhafs L, Hajjam A, Koukam A (2006) A combination of simulated annealing and ant colony system for the capacitated location-routing problem. In: Knowledge-Based Intelligent Information and Engineering Systems, pp 409–416

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, NY, USA, GECCO '06, pp 11–18

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. J Global Optim 19:121–139

Chen CH, Ting CJ (2008) Combining lagrangian heuristic and ant colony system to solve the single source capacitated facility location problem. Transp Res Part E: Log 44(6):1099 – 1122

Chen WN, Zhang J (2009) Ant colony optimization approach to grid workflow scheduling problem with various QoS requirement. IEEE T Sys Man Cybern C 31:29–43

Cordon O, Herrera F, Stützle T (2002) A review on the ant colony optimization metaheuristic: Basis, models and new trends. Mathw Soft Comput 9:141–175

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, 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 Netw 24:699–708

Deneubourg JL, Aron S, Goss S, Pasteels JM (1990) The self-organizing exploratory pattern of the argentine ant. J Insect Behav 3:159–168

Dorigo M, Blum C (2005) Ant colony optimization theory: A survey. Theor Comput Sci 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, Stützle T (2004) Ant Colony Optimization. MIT Press, Cambridge, MA

Dorigo M, Maniezzo V, Colorni A (1996) The ant system: Optimization by a colony of cooperating agents. IEEE T Sys Man Cybern B 26(1):29–41

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

Fontes DB, Hadjiconstantinou E, Christofides N (2006a) A branch-and-bound algorithm for concave network flow problems. J Global Optim 34:127–155

Fontes DBMM, Gonçalves JF (2007) Heuristic solutions for general concave minimum cost network flow problems. Networks 50:67–76

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 (2006b) Lower bounds from state space relaxations for network routing problems. J Global Optim 34:97–125

Fontes DBMM, Hadjiconstantinou E, Christofides N (2006c) A new dynamic programming approach for single-source uncapacitated concave minimum cost network flow problems. Eur J Oper Res 174:1205–1219

Gallo G, Sandi C, Sodini C (1980) An algorithm for the min concave cost flow problem. Euro 4:249–255

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. Eur J Oper Res 180(1):116–148

Geunes J, Pardalos P (2005) Supply Chain Optimization. Springer, Berlin

Goss S, Aron S, Deneubourg J, Pasteels J (1989) Self-organized shortcuts in the argentine ant. Naturwissenschaften 76:579–581

Guisewite G, Pardalos P (1991) Algorithms for the single-source uncapacitated minimum concave-cost network flow problem. J Global Optim 3:245–265

Guisewite GM (1995) Network problems. In: Horst R, Pardalos PM (eds) Handbook of Global Optimization, Kluwer Academic Publishers, pp 506–648

Guisewite GM, Pardalos PM (1993) A plynomial time solvable concave network flow problem. Networks 23:143–147

Hochbaum D, Segev A (1989) Analysis of a flow problem with fixed charges. Networks 19:291–312

Horst R, Thoai NV (1998) An integer concave minimization approach for the minimum concave cost capacitated flow problem on networks. OR Spektrum 20:47–53

Hu XM, Zhang J, Xiao J, Li Y (2008) Protein folding in hydrophobic-polar lattice model: A flexible ant-colony optimization approach. Protein Peptide Lett 15:469–477

Kennington J, Unger V (1976) A new branch-and-bound algorithm for the fixed charge transportation problem. Manag Sci 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

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

Lamar BW (1993) Network Optimization Problems: Algorithms Applications and Complexity, World Scientific, chap A method for solving network flow problems with general nonlinear arc costs, 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

Meshoul S, Batouche M (2002) Ant colony system with extremal dynamics for point matching and pose estimation. In: 16th International Conference on Pattern Recognition, vol 3, pp 823–826

Monteiro MSR, Fontes DBMM, Fontes FACC (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

Mullen R, Monekosso D, Barman S, Remagnino P (2009) A review of ant algorithms. Expert Syst Appl 36:9608–9617

Nahapetyan A, Pardalos P (2008) Adaptive dynamic cost updating procedure for solving fixed charge network flow problems. Comput Optim Appl 39:37–

50
Ortega F, Wolsey LA (2003) A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. Networks 41:143–158
Palekar US, Karwan MH, Zionts S (1990) A branch-and-bound method for the fixed charge transportation problem. Manag Sci 36(9):1092–1105
Parpinelli RS, Lopes HS, Freitas AA (2002) Data mining with an ant colony optimization algorithm. IEEE T Evolut Comput 6:321–332
Putha R, Quadrifoglio L, Zechman E (2012) Comparing ant colony optimization and genetic algorithm approaches for solving traffic signal coordination under oversaturation conditions. Comput-Aided Civ Infrastruct Eng 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. J Heuristics 7:261–304
Rebennack S, Nahapetyan A, Pardalos P (2009) Bilinear modeling solution approach for fixed charge network flow problems. Optim Lett 3:347–355
Reimann M, Laumanns M (2006) Savings based ant colony optimization for the capacitated minimum spanning tree problem. Comput Oper Res 33:1794–1822
Shyu SJ, Lin BMT, Hsiao TS (2006) Ant colony optimization for the cell assignment problem in PCS networks. Comput Oper Res 33(6):1713–1740
Smith DK, Walters GA (2000) An evolutionary approach for finding optimal trees in undirected networks. Eur J Oper Res 120(3):593 – 602
Soland RM (1974) Optimal facility location with concave costs. Oper Res 22:373–382
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
Tuy H (2000) Strong polynomial-time solvability of a minimum concave cost network flow problem. Acta Mathematica Vietnamica 25:209–217
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, Lect Notes Comput Sc, vol 4150, Springer Berlin / Heidelberg, pp 348–355
Wagner HM (1958) On a class of capacitated transportation problems. Manag Sci 5:304–318
Yin PY, Wang JY (2006) Ant colony optimization for the nonlinear resource allocation problem. Appl Math Comput 174:1438–1453
Zangwill W (1968) Minimum concave cost flows in certain networks. Manag Sci 14:429–450
Zhang J, Chung H, Lo WL, Huang T (2009) Extended ant colony optimization algorithm for power electronic circuit design. IEEE T Power Electr 24:147–162