Journal of Global Optimization (2006) 34: 127–155
DOI 10.1007/s10898-005-1658-x

A Branch-and-Bound Algorithm for Concave Network Flow Problems

DALILA B. M. M. FONTES[1], ELENI HADJICONSTANTINOU[2] and NICOS CHRISTOFIDES[2]
[1]*LIACC, Faculdade de Economia da Universidade do Porto Rua Dr. Roberto Frias, 4200-464 Porto, Portugal (e-mail: fontes@fep.up.pt)*
[2]*Tanaka Business School, Imperial College London, South Kensington Campus, LondonSW7 2AZ, UK (e-mail:{e.hconstantinou, n.christofides}@imperial.ac.uk)*

(Received: 26 January 2005; accepted: 27 July 2005)

Abstract. In this paper a Branch-and-Bound (BB) algorithm is developed to obtain an optimal solution to the single source uncapacitated minimum cost Network Flow Problem (NFP) with general concave costs. Concave NFPs are NP-*Hard*, even for the simplest version therefore, there is a scarcity of exact methods to address them in their full generality. The BB algorithm presented here can be used to solve optimally single source uncapacitated minimum cost NFPs with any kind of concave arc costs. The bounding is based on the computation of lower bounds derived from state space relaxations of a dynamic programming formulation. The relaxations, which are the subject of the paper (Fontes et al., 2005b) and also briefly discussed here, involve the use of non-injective mapping functions, which guarantee a reduction on the cardinality of the state space. Branching is performed by either fixing an arc as part of the final solution or by removing it from the final solution. Computational results are reported and compared to available alternative methods for addressing the same type of problems. It could be concluded that our BB algorithm has better performance and the results have also shown evidence that it has a sub-exponential time growth.

Key words: Branch-and-bound, concave network flows, dynamic programming, global optimization, state space relaxation.

1. Introduction

Network flow problems arise frequently in several application areas (Guisewite, 1994): transportation, communication, network design and distribution, production and inventory planning, facility location, scheduling and air traffic control. The main feature defining the complexity of minimum cost Network Flow Problems (NFPs) is the type of cost function for each arc. The key source of complexity for concave NFPs arises from minimizing a concave function over a convex feasible region, defined by the network constraints, which implies that a local optimum is not necessarily a global optimum. Concave NFPs are known to be NP-*hard* (Guisewite and Pardalos, 1991a), even for the simplest version – Single Source Uncapacitated (SSU) with fixed-charge (FC) arc costs. FC NFPs are problems with

arc cost functions that consist of a fixed cost and a linear variable cost. It should be noticed that every NFP with general nonlinear cost functions can be transformed into a concave NFP on an expanded network (Lammar, 1993). And also, that multiple source and capacitated NFPs can be transformed into single source and uncapacitated NFPs (Zangwill, 1968).

Numerous exact and heuristic methods have been proposed over the years. Most of the work developed on concave NFPs considers problems with FC cost functions, which are a particular case of the more general concave cost functions. A thorough survey of the state-of the art for network problems is given by Guisewite (1994). Kim and Pardalos have developed linearization methods to solve the NFP with FC costs (Kim and Pardalos, 1999) or piecewise concave costs (Kim and Pardalos, 2000a,b). Their solution strategy first linearizes the objective function and then solves the resulting linear program as a series of linear shortest path problems. Kim and Hooker (2002) propose a Branch-and-Bound (BB) method for FC NFPs, which combines constraint programming with linear programming techniques. At each node of the search tree the constraint programming is used to reduce the domain of a discrete variable and thus, the number of branches. A linear programming relaxation provides a bound on the optimal value of the problem. The method is twice as fast as mixed integer linear programming for FC transportation problems but not as fast for FC NFPs. Ortega and Wolsey (2003) developed a branch-and-cut algorithm for uncapacitated FC NFPs by extending the cutting planes used for solving uncapacitated lot sizing problems previously developed in (Cordier et al., 1999).

Linear approximations in a Dynamic Programming (DP) approach are given by Burkard et al. (2001). The authors consider acyclic networks with small degree vertices and general concave costs, for which they successively solve linear problems. The linear underestimations are updated by using information about the Lower Bounds (LBs). Since the LBs are feasible, the authors obtain upper bounds by recomputing the cost of the solution using the original cost functions. The method was tested on layered networks, transportation networks, and SSU flow networks and they were able to conclude that the performance was much better for layered networks. Fontes et al. (2003) address SSU general concave NFPs using a local search method. Different and informed initial solutions to restart the search are obtained from the information about the LB solutions. The LBs are computed by a state space relaxation procedure, which is described in Fontes et al. (2005b). Guisewite and Pardalos (1991b) developed a BB algorithm based on that of Gallo et al. (1980). The bounding process is enhanced by projecting the LB on the cost of extending the current path. Recently, Horst and Thoai (1998) developed a BB method for NFPs where the flow cost function is concave (without a fixed cost component) for a

fixed number of arcs and linear for the remainder. In their method, the branching is performed by integral rectangular partitions and the bounding by solving linear NFPs. The linearizations are obtained by convex envelopes. Fontes et al. (2005a, in press) also developed a DP approach to solve optimally the SSU concave NFP. The DP formulation has two main characteristic: (i) no assumption other than separability and additivity is needed and (ii) it is independent of both the type of cost functions considered and of the number of nonlinear arc costs. Hence, different types of cost functions can be considered.

In this paper, a new BB algorithm for the SSU concave NFP is presented. The bounding is performed by using LBs derived from state space relaxations of a DP formulation. This formulation is stated in Section 2 and is discussed in greater detail in Fontes et al. (2005a). The LBs derived by the state space relaxation are improved by using Lagrangian penalties and state space modifications, and also by the development of additional constraints. A brief account is given in Section 3 and a detailed discussion is provided in Fontes et al. (2005b). In Section 4 we give details of the BB algorithm as well as of its implementation, namely on how to modify the state space in order to incorporate the branching decisions previously made. The computational experiments reported in Section 5 include FC NFPs and NFPs involving both a concave variable cost component and a fixed cost component for all arcs. The latter problems are amongst the most difficult concave NFPs and have only been addressed by Burkard et al. (2001), Fontes et al. (2003, 2005a).

2. Problem Definition and Formulation

The network $G = (W, A)$ to be optimized consists of a set W of $n+1$ vertices (vertices $1, \ldots, n$ denote demand vertices and vertex $n+1$ denotes the source vertex t) and a set A of m directed arcs, $A \subset \{(i, j) : i \in W, j \in W \setminus \{t\}\}$. Each demand vertex has associated a non-negative integer demand r_i. The supply at the source vertex R matches the total demand required by the n demand vertices. A solution structure is characterized by the flow r_{ij} on each arc $(i, j) \in A$. A general non-decreasing, non-negative, and concave cost function g_{ij} is associated with each arc (i, j) and satisfies $g_{ij}(0) = 0$. The objective is to find a subset of arcs and arc flows that satisfy the demand at minimum cost.

SSU concave NFPs have a finite optimal solution if and only if there exists a direct path going from the source to every demand vertex and if there are no negative cost cycles. Thus, a feasible flow is an extreme flow if it contains no positive cycles. For the uncapacitated case a positive cycle is a cycle with all arcs (i, j) satisfying $r_{ij} > 0$. Therefore, for the SSU case an extreme flow is a tree rooted at the single source spanning all demand

vertices (Zangwill, 1968). The objective in solving this class of problems is therefore equivalent to find a minimum cost directed tree network that satisfy all customers demand.

The key characteristics of the following DP formulation, developed in Fontes et al. (2005a) are: independence of the type and form of cost functions, independence of the number of non-linear arc costs, and no assumptions other than separability and additivity.

Consider a set $S \subseteq W$ and a vertex $x \in S$. Let $\{S', \bar{S}'\}$ be one partition of set S, where $S' \subseteq S \setminus \{x\}$ and $\bar{S}' = S \setminus S'$. For each possible set S', let $z \in S'$ be the root vertex of a directed tree spanning the set S'. Let $f(S', z)$ be the minimum cost of supplying all demand vertices in S' with the required commodity available at vertex z through a directed tree rooted at z. The minimum cost of supplying a set S' from vertex $x \notin S'$ with the required commodity made available at some vertex $z \in S'$ is found by determining the best combination of the minimum cost directed tree of S' rooted at vertex $z \in S'$ with the cost of arc (x, z), that is $\min_{z \in S'} \{f(S', z) + g_{xz}(\sum_{i \in S'} r_i)\}$.

By definition, the minimum cost incurred in supplying the remaining demand vertices of set S not in S' from x is given by $f(\bar{S}', x)$. From the above, the minimum cost $f(S, x)$ of supplying all demand vertices in S, with the commodity available at $x \in S$, is obtained by examining all possible subsets $S' \subseteq S \setminus \{x\}$ and given by

$$f(S, x) = \min_{S' \subseteq S \setminus \{x\}} \left[f(S - S', x) + \min_{z \in S'} \left[f(S', z) + g_{xz} \left(\sum_{i \in S'} r_i \right) \right] \right]. \tag{1}$$

Initial conditions for recursion (1) are provided by $f(\{x\}, x) = 0, \forall x \in W$. Hence, the optimal cost of supplying all demand vertices in set W from the source vertex t, is given by

$$f^* \equiv f(W, t) = \min_{S' \subseteq W \setminus \{t\}} \left[f(W - S', t) + \min_{z \in S'} \left[f(S', z) + g_{tz} \left(\sum_{i \in S'} r_i \right) \right] \right]. \tag{2}$$

3. State Space Relaxation

Due to the large dimensionality of the state space, few combinatorial problems of large dimension can be solved efficiently by DP alone. In our model there are $(n+2)2^{n-1}$ states in the DP formulation and $2^{2n-2}(n^2+4n+8) - 2^{n-1}(3n^2+4n+8) + n + 2$ transitions between states. (Details of the state space analysis can be found in Fontes et al. (2005a).) State Space Relaxation (SSR) is a procedure for relaxing the state space associated with the DP recursion that was first introduced by Christofides et al. (1981) and extended

by Fontes et al. (2005b) to handle problems that involve transitions that go across several stages. The solution to the relaxed recursion provides a LB, in the case of minimization, to the optimal solution value.

In the DP formulation given by Equation (1) a state is represented by a pair (S, x). The vertex x, acts as a source to supply the set of vertices S, with x in S. The stage is given by the cardinality of the set S. In the relaxation, a non-injective mapping function h is used to represent the original states (S, x) as relaxed states $(h(S), x)$. The mapping function h can take different forms, in fact it can be any separable function. Three forms of the mapping function, defining three relaxations are discussed in Fontes et al. (2005b). From those we use here the so-called Combined relaxation where h is defined as

$$h(S) = (p, q, r) = \left(|S|, \sum_{i \in S} q_i, \sum_{i \in S} r_i \right),$$

where p is the cardinality of set S, r_i is the demand of vertex i and q_i is a non-negative integer weight associated with vertex i.

The original state (S, x) is mapped into (p, q, r, x). Thus, the relaxed recursion becomes,[1]

$$f(p, q, r, x) = \min_{\substack{p' \leqslant p-1 \\ q' \leqslant q-q_x \\ r' \leqslant r-r_x}} \Big[f\left(p - p', q - q', r - r', x\right) + \min_{\substack{z \neq x \\ p' \geqslant 1 \\ q' \geqslant q_z \\ r' \geqslant r_z}} \left[f\left(p', q', r', z\right) + g_{xz}\left(r'\right) \right] \Big] \tag{3}$$

and is initialized by

$$f(p, q, r, x) = \begin{cases} 0 & \text{if } p = 1,\ q = q_x, \quad \text{and } r = r_x, \\ +\infty & \text{otherwise.} \end{cases} \tag{4}$$

3.1. STATE SPACE ASCENT

The solution to the relaxed recursion (3) provides a LB to the value of a true optimum which is improved by a State Space Ascent (SSA) procedure that considers Lagrangean penalties and state space modifications. The general idea is to force the solution to the relaxed problem "closer"

[1]There is a slight abuse of notation as we use f to denote the minimum cost associated with a state in the original state space as well as in the relaxed state space. This should not be confusing as from here onwards we are no longer interested in the original state space.

to feasibility by penalizing the vertices not exactly satisfied using a penalty λ_i or by modifying the weight q_i of such vertices.

A three-phase procedure is used to improve the LB obtained by maximizing B, which is computed by solving Equation (5) with updated cost functions $g'_{ij}(r) = g_{ij}(r) + \lambda_j$.

$$B(\lambda, q) = \min_{\substack{p' \leqslant P-1 \\ q' \leqslant Q-q_t \\ r' \leqslant R-r_t}} \Big[f\left(P-p', Q-q', R-r', t\right) + \min_{\substack{z \neq t \\ 1 \leqslant p' \\ q_z \leqslant q' \\ r_z \leqslant r'}} \left[f\left(p', q', r', z\right) + g'_{tz}\left(r'\right) \right] \Big] - \sum_{i \in V} \lambda_i. \tag{5}$$

Phase I is performed for a pre-specified number of iterations. The penalties and weights are initialized as $\lambda_i^0 = 0$ and $q_i^0 = 0$, respectively, and at iteration k, the penalties are updated. The second phase picks-up from the best LB found during phase I. Thus, the weights and the penalties are initialized with the corresponding values. State space modifications are applied over a pre-specified time period and the penalties remain unchanged. In the third phase, the penalties and weights are reinitialized at the values corresponding to the best LB found so far. This is then improved by updating the penalties as in phase I, for a pre-specified number of iterations.

At iteration k the penalties/weights are updated taking into account its previous value, the net flow supplied to customer i, and the gap between the current LB and the best feasible solution found so far. (The upper bound is obtained using the local search procedure described in Fontes et al., 2003). Although such changes are based on the subgradient method (Held et al., 1974), they are not standard Lagrangean equations as the penalties/weights are not allowed to become negative and a reduction factor is applied to prevent large changes.

The LB is further improved by restricting the searchable space, which is accomplished by the use of additional constraints, namely: constraints enforcing the use of sets, constraints to supply only reachable vertices, and constraints using the cardinality of the partition.

Details of the SSA procedure, including penalties and weights updating formulae and the development of the additional constraints are given in Fontes et al. (2005b).

4. The Branch-and-bound Scheme

The BB methodology is based on the idea of dividing the set of all possible solutions into smaller and smaller subsets (branching) and to compute for each subset a LB on the cost of the best solution therein (bounding). The

objective of computing the LB is first to limit the search and identify how far the optimal solution is, and second to provide some information that can be used as a guidance for partitioning the subsets. Regardless of how we develop the BB tree, the quality of the bounds (quality of relaxations) is the primary factor that determines the efficiency of a BB algorithm. Nevertheless, the choice of tree development strategies, such as which sub-problem corresponding to an active node (a node still to be partitioned further) should be considered next is also significant. The variable which should be selected for the division of such node is also another important efficiency factor.

The process of supplying customers from the source vertex t can be recorded in terms of a tree. Thus, branches emanating from a node of the BB tree correspond either to the decision of fixing an arc as part of the solution, or to the decision of not using such arc. The problem is then split into two sub-problems by adding two mutually exclusive and exhaustive constraints.

4.1. THE BB ALGORITHM

The state of the search procedure at each intermediate node u of the BB algorithm is described by the partial solution, typically a directed tree supplying a subset of vertices and rooted at the source vertex t. This subset includes at least all vertices in the sequence of fixed arcs corresponding to the path from the root node to node u.

The BB algorithm uses a modified version of the SSA procedure to compute a LB on the cost of supplying all customers taking into account the decisions already made (arcs fixed in the solution and arcs eliminated from the solution), which is obtained as $Z_{LB} = Z'_{LB} - \sum_{i \in V} \lambda_i$, where Z'_{LB} is given by Equation (28).

At each BB node, LB improvements are again performed. These can be achieved by (a) updating penalties, (b) updating weights, or (c) updating both penalties and weights. Different behaviours are expected for different alternatives. For example, smaller computational times are expected if only the penalties are being changed as the state space is decreasing. (When a vertex is fixed as part of the solution its weight can be reduced to zero without affecting the search for the solution.) On the other hand, a larger number of BB nodes is expected. Due to the possible trade offs between the options available, three versions of the BB procedure were developed and implemented. In BB1 only penalties are changed, BB2 only changes the weights, while BB3 initially updates the penalties and at latter stages updates the weights. At each BB node the penalties and weights are initialized with the values corresponding to the best LB attained at the parent node. If an arc, say (x, y) is being fixed in the final solution, then the

weight associated with vertex y is set to zero (as this does not affect the search for the LB and reduces the total weight). At each intermediate node of the BB tree a pre-specified number of iterations is performed. In addition, there is a time limit at each BB node and also a constraint preventing more than 15 iterations to be performed if no improvement to the LB value has been achieved. Moreover, if the LB value is improving the algorithm is not allowed to move to another BB node, not even if the time and iteration limits have been reached. A compromise between the measures of theoretical quality (number of BB nodes) and practical applicability (computational time required) has been used for this algorithm.

The main components of the BB algorithm are sub-problems that we must select (selection), solve (bounding and checking elimination criteria), and divide (branching if the sub-problem has not been eliminated). The evolution of the BB algorithm is stored in a binary tree. The flow diagram of this algorithm is given in Figure 1, while in Appendix A the steps involved are described.

The branching and selection strategies are as discussed in Sections 4.2 and 4.3, respectively. The bounding is based on the SSA procedure previously introduced in Fontes et al. (2005b) and given here in Section 3, that is modified in order to accommodate for fixing arcs. Since decisions are being made on arcs, i.e., decisions on whether an arc must be part of the final solution or eliminated from the final solution, we need to modify the state space representation by including an extra state variable. This modification is explained in Section 4.4.1 and originates the modified recursion given in Section 4.4.3.

4.2. BRANCHING

Branching divides a tree node, corresponding to a sub-problem, into two mutually exclusive and exhaustive sub-problems. In this manner, the algorithm constructs a binary tree of sub-problems. The branching strategy involves the selection of an arc from the set of arcs available at a specific tree node, on which to branch next. An arc, say (x, y) is chosen at a specific tree node, where vertex x is either a vertex already fixed as part of the solution or the source vertex, in order to extend a partial solution by supplying vertex y directly from vertex x. This way, we can guarantee that the graph of the fixed arcs is connected and in fact, a tree. The first branching is to fix the arc in the solution. The alternative branching is to reject arc (x, y), i.e. customer y cannot be supplied directly from vertex x.

In choosing arc (x, y) for branching several rules were tried in order to improve the computational performance of the algorithm. The one adopted, always chooses the arc to branch next among arcs obtained in the solution associated with the LB computation at the current tree node. The

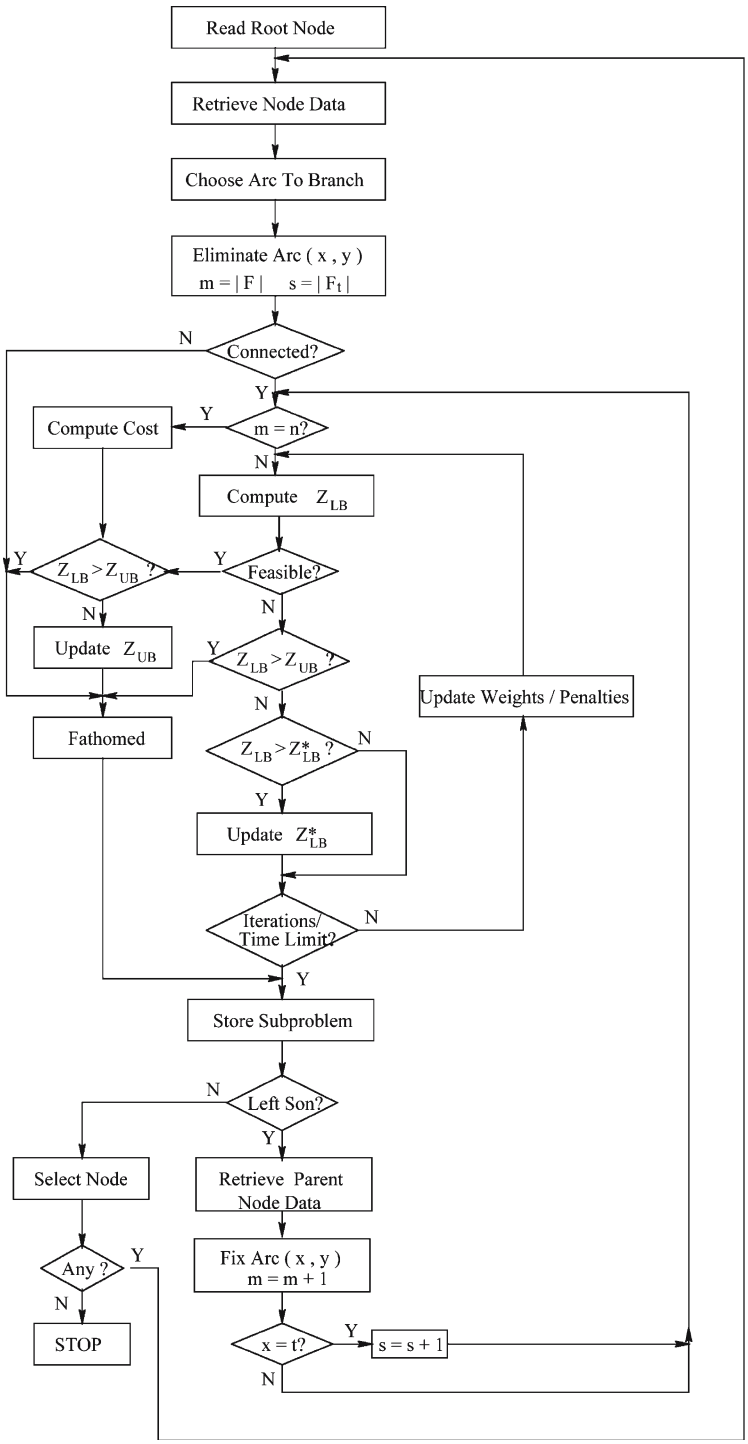


Figure 1. Flow diagram of the BB algorithm.

following sequence of steps is applied to select an arc, say (x, y) for further branching. (x represents a fixed vertex or the source and y is the candidate vertex for selection.)

(a) Most oversupplied vertex y, corresponding to an arc used repeatedly, or to an arc associated with loops,

$$\frac{\sum_{z \in W} r_{zy} - \sum_{z \in V} r_{yz}}{r_y},$$

where r_{zy} is the flow on arc (z, y) and r_y is the demand of vertex y. Note that the above ratio is greater than one, only if vertex y is supplied more than once, either by the repeated usage of arc (x, y) or by at least two different arcs.

(b) If there are no arcs responsible for infeasibilities to be fixed, then choose the one "closest" to such an arc. The distance here is measured in terms of the number of arcs separating the two arcs in question.

(c) If more than one arc is found (after applying (a) or (b)) then the following criteria are used in turn to choose an arc such that the vertex to be fixed has

(i) highest demand,
(ii) lowest depth, and
(iii) largest state space reduction.

4.3. NODE SELECTION RULE

At any stage of the BB algorithm having more than one sub-problem to be considered further, a choice of which sub-problem to use next has to be made. Such choice may have an important role in the efficiency of the BB algorithm. Several rules exist and they can be of two types: *a priori rules*, that determine in advance the order in which the tree will be developed; and *adaptive rules*, that choose a node using information (bounds) about the status of active nodes.

In the implementation of our BB algorithm the *best-first* search adaptive rule was adopted. The next node to visit is the node with the largest gap between the corresponding LB and the best upper bound available. In the search for the optimal solution this node would necessarily be visited because its LB value is the smallest and hence, it can not be fathomed. In our implementation, when a node is considered for further partitioning then both sons (the nodes corresponding to the two sub-problems found by adding the two mutually exclusive and exhaustive decisions on an arc) will

be solved at once, since this saves computational time in backtracking, i.e., searching for the constraints that define the sub-problem.

4.4. BOUNDING

Bounds are obtained at each tree node by solving the corresponding relaxed sub-problem using a modified version of the SSA procedure described in Fontes et al. (2005b) and mentioned earlier. Each of these sub-problems is identified by two sets of arcs, a set of arcs that cannot be used in the solution (arcs ruled out of the solution by the branching decisions) and a set of arcs that must be part of the solution (arcs fixed as part of the solution by the corresponding branching decisions).

Since the problem being solved is a minimization problem, to eliminate an arc from the solution it is enough to make the cost of such arc prohibitive, i.e. very high when compared with the cost of the other arcs. By doing so, when solving the sub-problem this arc will not be chosen because there are cheaper alternatives.

When dealing with the arcs that must be part of the sub-problem solution a different approach has to be taken. The objective of forcing a particular set of arcs to be in the solution cannot be achieved by simply changing the cost of these arcs or the cost of these arcs together with the costs of some other arcs.

A possible way of handling arcs fixed as part of the solution is to eliminate from the problem being solved the part corresponding to the decisions already made. More specifically, if an arc (t, x) is fixed as part of the solution then the problem to be solved becomes to supply the remaining demand vertices. Arc (t, x) cannot be removed from any sub-problem being solved since it might, in the optimal solution, be used to route an amount of flow larger than the fixed flow being routed through it (initially the demand of vertex x). Furthermore, the cost function of the arcs fixed as part of the solution must be changed to reflect the fact that the arc is already being used and a certain amount of flow is already being routed through it.

This type of approach, which is discussed next, was initially implemented but, as expected, it was not very efficient, specially as the problem size increases. Therefore, we developed a different approach that involves changing the state space representation and consequently the recursion such that the modified version takes into account the decisions already made. This latter approach is the subject of Section 4.4.1.

Initially, when arc (t, x) is fixed as part of the solution, r_x the demand of vertex x is being routed through it. Suppose another arc say (x, y) is also fixed as part of the solution, then both arcs (t, x) and (x, y) are fixed in the solution. The flow on these arcs is $r_x + r_y$ and r_y, respectively. Since the arc cost functions are not linear, they need to be modified as follows.

$$g''_{xy}(r_{xy}) = \begin{cases} g'_{xy}(r_{xy}) & \text{if } (x, y) \text{ is not fixed,} \\ g_{xy}(r_{xy} + r) - g_{xy}(r) + \lambda_y & \text{otherwise,} \end{cases} \tag{6}$$

where r is the accumulated flow previously fixed in arc (x, y) and $g'_{xy}(r_{xy}) = g_{xy}(r_{xy}) + \lambda_y$.

Let u be the current tree search node and F_u be the set of vertices corresponding to the right-end point of the set of arcs fixed as part of the solution. The LB solution is now obtained by computing

$$f(n + 1 - |F_u|, Q_u, R_u, t) - \sum_{i \in V \setminus F_u} \lambda_i,$$

where $Q_u = \sum_{i \in V \setminus F_u} q_i$, $R_u = \sum_{i \in V \setminus F_u} r_i$, and V is the set of the n demand vertices $V = W \setminus \{t\}$. Therefore, the state space of the current sub-problem is much smaller than the state space of the original problem as $|F_u|$ arcs and thus vertices have already being fixed as part of the final solution.

The relaxed recursion is then rewritten as

$$f(p, q, r, x) = \min_{(p', q', r') \in \mathcal{S}_x(p,q,r)} \Bigg[f\left(p - p', q - q', r - r', x\right) + \min_{\substack{z \neq x \\ (p', q', r') \in \mathcal{S}_z}} \left[f(p', q', r', z) + g''_{xz}(r') \right] \Bigg] \tag{7}$$

subject to the additional constraints that reduce the searchable space, where

$$\mathcal{S}_x(p, q, r) = \{(p', q', r') \in \mathbb{N}_0^3 : \text{Equations (8)–(10) are satisfied}\}$$

and

$$\mathcal{S}_z = \{(p', q', r') \in \mathbb{N}_0^3 : \text{Equations (11)–(13) are satisfied}\}.$$

$$p' \leqslant \begin{cases} p - 1 & \text{if no arc } (i, x) \text{ is fixed,} \\ p & \text{otherwise,} \end{cases} \tag{8}$$

$$q' \leqslant \begin{cases} q - q_x & \text{if no arc } (i, x) \text{ is fixed,} \\ q & \text{otherwise,} \end{cases} \tag{9}$$

$$r' \leqslant \begin{cases} r - r_x & \text{if no arc } (i, x) \text{ is fixed,} \\ r & \text{otherwise,} \end{cases} \tag{10}$$

$$p' \geqslant 1, \tag{11}$$

$$q' \geqslant \begin{cases} q_z & \text{if no arc } (i,z) \text{ is fixed,} \\ \min\limits_{i \notin F_u} q_i & \text{otherwise,} \end{cases} \tag{12}$$

$$r' \geqslant \begin{cases} r_z & \text{if no arc } (i,z) \text{ is fixed,} \\ \min\limits_{i \notin F_u} r_i & \text{otherwise,} \end{cases} \tag{13}$$

In the approach just described there are basically two ways of representing a vertex for which the supplying route has been fixed. The vertex is either represented by the triplet $(0,0,0)$ or by the triplet $(1,0,0)$. The penalties for these vertices are set to zero, since both its demand and weight have already been supplied. In the first representation we have

$$Z_{LB} = f(n+1-|F_u|, Q_u, R_u, t) - \sum_{i \in V \setminus F_u} \lambda_i,$$

while in the second

$$Z_{LB} = f(n+1, Q_u, R_u, t) - \sum_{i \in V \setminus F_u} \lambda_i.$$

Note that, in the former case vertices in F_u might or might not be in the solution, while in the latter case they must be in the solution. Thus, in the first representation the partition $(0,0,0)$ must be allowed in order to allow for supplying routes including vertices in F_u, that have zero cardinality, demand, and weight. By allowing these type of routes, that may as well happen, the problem of how to prevent the method of cycling is faced. To prevent this phenomenon the algorithm complexity increases, and this increase is dependent on the number of such vertices. In the second representation $(1,0,0)$, the state variable p is of no use, as the constraint associated with it can always be verified by using vertices in F_u, that can be included at no cost. Thus, many more partitions have to be considered, since a state (p,q,r,x) can be represented by any group[2] of vertices satisfying $\sum_{i \in V \setminus F_u} \alpha_i \cdot r_i = r - r_x$, $\sum_{i \in V \setminus F_u} \alpha_i \cdot q_i = q - q_x$, and $\sum_{i \in V \setminus F_u} \alpha_i \leqslant p-1$ with $\alpha_i \geqslant 0$ and integer and $p-1-\alpha_i$ vertices taken from F_u.

4.4.1. *Fixing arcs by modifying the state space*

To incorporate into the relaxed recursion the decisions on arcs fixed as part of the final solution we need an extra variable s to represent the state

[2]Here and hereafter, we use the word group to denote a collection of vertices that may include some vertices more than once. The elements in the group are not necessarily all different: actually if they are all different then an optimal solution to the original problem has been found.

space. A state is then represented by (p, q, r, x, s). Variable s can be any integer between 0 and n and is associated with the number of arcs emanating from vertex x that have been fixed as part of the solution.

The computation of a state, say (p, q, r, x, s) is divided into two parts: if $s = 0$, then vertex x has no outgoing arcs fixed as part of the solution and $f(p, q, r, x, 0)$ is computed as before. To explain the computation of $f(p, q, r, x, s)$ for $s > 0$ let us first introduce some notation.

Let F be the set of vertices such that an incoming arc is fixed as part of the solution and $F_x \subseteq F$ the set of vertices such that for some vertex i arc (x, i) has been fixed in the solution. Define T_x^s as the fixed subtree rooted at vertex x, which includes vertex x, and all fixed subtrees rooted at the first s vertices in F_x. By definition $T_x^0 = \{x\}$. Let v_x be a function mapping a fixed arc (x, i) onto vertex i. Assume that l arcs emanating from vertex x have already been fixed as part of the solution. If arc (x, y) has been the kth such arc, then $v_x(k) = y$.

Let us consider an example where we have already fixed the following sequence of arcs as part of the final solution: $(t, 1)$, $(t, 2)$, $(t, 10)$, $(2, 5)$, $(2, 8)$, $(10, 3)$, and $(8, 13)$.

The set of fixed vertices F is given by

$$\begin{aligned} F &= \{j \in V : (i, j) \text{ is fixed for all } i \in W\} \cup \{t\} \\ &= \{1, 2, 3, 5, 8, 10, 13, t\}. \end{aligned} \tag{14}$$

Sets F_x are defined as $F_x = \{i \in V : (x, i) \text{ is fixed, for all } x \in F\}$. Hence we have,

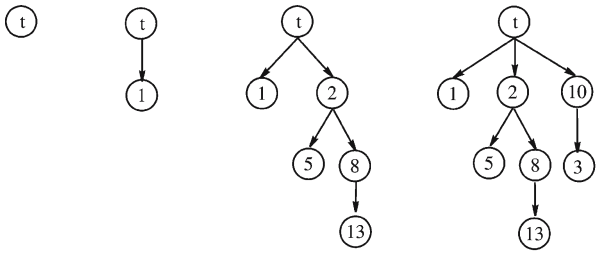
$$\begin{aligned} F_t &= \{1, 2, 10\}, \\ F_1 &= \emptyset, \qquad F_2 = \{5, 8\}, \; F_{10} = \{3\}, \\ F_5 &= \emptyset, \qquad F_8 = \{13\}, \quad F_3 = \emptyset, \\ F_{13} &= \emptyset. \end{aligned} \tag{15}$$

For $s = 0, 1, 2$, and 3 the fixed subtrees rooted at vertex t are given in Figure 2.

The function v mapping the fixed arcs onto the fixed vertices has the following values:

$$\begin{aligned} v_t(1) &= 1, \;\; v_t(2) = 2, \;\; \text{and } v_t(3) = 10, \\ v_2(1) &= 5 \text{ and } v_2(2) = 8, \\ v_{10}(1) &= 3, \\ v_8(1) &= 13. \end{aligned} \tag{16}$$

To compute $f(p, q, r, x, s)$ for $s > 0$, vertex x must supply vertex y, where $y = v_x(s)$, in such a way that y can supply a group of vertices containing at



(a) T_t^0 **(b)** T_t^1 **(c)** T_t^2 **(d)** T_t^3

Figure 2. Fixed trees rooted at the source vertex.

least all vertices in $T_y^{|F_y|}$. To supply such group of vertices we need a partition (p', q', r', y, m) that satisfies the conditions given in Equation (17). The remaining group of $p - p'$ vertices comprises at least: vertex x and all the vertices in the fixed sub-trees rooted at the vertices corresponding to the first $s-1$ vertices fixed as directly supplied by vertex x, which are given by T_x^{s-1}. Upper limits of p', q', and r' are given in Equation (18). Thus, only partitions satisfying Equations (17) and (18), are considered.

$$\begin{aligned} p' &\geqslant \left|T_y^{|F_y|}\right|, \\ q' &\geqslant \sum_{i \in T_y^{|F_y|}} q_i, \\ r' &\geqslant \sum_{i \in T_y^{|F_y|}} r_i, \\ m &= |F_y|. \end{aligned} \tag{17}$$

$$\begin{aligned} p' &\leqslant p - \left|T_x^{s-1}\right|, \\ q' &\leqslant q - \sum_{i \in T_x^{s-1}} q_i, \\ r' &\leqslant r - \sum_{i \in T_x^{s-1}} r_i. \end{aligned} \tag{18}$$

Consider again the example used above to illustrate the computation of $f(p, q, r, x, s)$. The LB solution for the current sub-problem is given by $f(P, Q, R, t, 3)$ since there are three vertices fixed as directly supplied by the source vertex. The computation of $f(P, Q, R, t, 3)$ requires the evaluation of all partitions represented by (p', q', r', y, m), where $y = v_t(3) = 10$, satisfying

$$\begin{aligned} p' &\geqslant |T_{10}^m| = 2, \\ q' &\geqslant \sum_{i \in T_{10}^m} q_i = q_{10} + q_3, \\ r' &\geqslant \sum_{i \in T_{10}^m} r_i = r_{10} + r_3, \\ m &= |F_{10}| = 1. \end{aligned} \tag{19}$$

Vertex t has to supply directly other $s-1=2$ known vertices, thus upper limits on p', q' and r' must also be satisfied. These upper limits, which are given by Equation (20), correspond to lower limits of what must be kept at the source vertex so that it can supply at least vertices 1 and 2 and the fixed vertices they supply. In this case, the source vertex must supply at least vertices 1, 2, 5, 8, and 13.

$$\begin{aligned} p' &\leqslant P - |T_t^2| = P - 6, \\ q' &\leqslant Q - \sum_{i \in T_t^2} q_i = Q - (q_t + q_1 + q_2 + q_5 + q_8 + q_{13}), \\ r' &\leqslant R - \sum_{i \in T_t^2} r_i = R - (r_t + r_1 + r_2 + r_5 + r_8 + r_{13}). \end{aligned} \tag{20}$$

4.4.2. *Reduction of the space to search*

Constraints to eliminate part of the space to search and thus, improve the efficiency of the LBs found have been introduced in Fontes et al. (2005b). They are now revisited since by using information on the arcs already fixed in the final solution as well as information on the arcs eliminated from the final solution it is possible to take these improvements a step further. Assume that state (p, q, r, x, s) is under consideration.

Constraints forcing the partitions to consider sets of vertices (rather than groups) can be rewritten in a tighter way. Now, we consider one four-dimensional matrix instead of the previous two three-dimensional ones. Let σ be the four-dimensional matrix defining which partitions can be represented by sets respecting the conditions for vertices x and z, the vertices acting as sources in the partition. This matrix is initially set to zero and then set to one for each subset $S \subseteq V \setminus F$ and each vertex x in $W \setminus S$, with $p = |S|$, $q = \sum_{i \in S} q_i$, and $r = \sum_{i \in S} r_i$.

- If $s = 0$, q_i and r_i are the weight and demand associated with vertex i then the constraints using the σ matrix are given by:

$$\begin{aligned} &\sigma(p - p' - 1, q - q' - q_x, r - r' - r_x, x) = 1, \\ &\sigma(p - p' - 1, q - q' - q_x, r - r' - r_x, z) = 1, \\ &\sigma(p' - 1, q' - q_z, r' - r_z, x) = 1, \\ &\sigma(p' - 1, q' - q_z, r' - r_z, z) = 1. \end{aligned} \tag{21}$$

- If $s > 0$ these constraints are given as in Equation (22), where p_i^m, q_i^m, and r_i^m are now computed by considering all the vertices in the fixed tree T_i^m and no longer only vertex i. These variables, take the values of the cardinality, the weight, and the demand associated with vertex i if $m = 0$; Otherwise they are computed as in Equation (23), where $m = s - 1$ for vertex x and $m = |F_z|$ for vertex z. (Note that $r_i^0 = r_i, q_i^0 = q_i$, and $p_i^0 = 1$ as $T_i^0 = \{i\}$.)

$$\begin{aligned} &\sigma(p - p' - p_x^{s-1}, q - q' - q_x^{s-1}, r - r' - r_x^{s-1}, z) = 1, \\ &\sigma(p' - p_z^s, q' - q_z^m, r' - r_z^m, z) = 1. \end{aligned} \tag{22}$$

$$\begin{aligned} p_i^m &= |T_i^m|, \\ q_i^m &= \sum_{j \in T_i^m} q_j, \\ r_i^m &= \sum_{j \in T_i^m} r_j. \end{aligned} \tag{23}$$

  Vertex x is already fixed in the solution therefore, these constraints only need to be written for vertex z, as given above in Equation (22). If for some specific values of p, q, and r the σ matrix verifies the aforementioned conditions for vertex z, which might or might not be fixed in the solution, then it also verifies them for vertex x.

The time complexity to compute this matrix grows exponentially with the problem size. Fortunately, it also decreases exponentially with the number of vertices fixed as part of the solution. During the BB algorithm, the time complexity is a function of the vertices not yet fixed, i.e. $\mathcal{O}(2^{n-n_f})$, where n_f is the number of vertices fixed as part of the final solution.

Constraints on reachable vertices are written in the same way but the values are different since sets V_z get smaller and smaller as the search tree is being developed. Some arcs have been eliminated through direct branching decisions and some more as a consequence of forcing an arc into the solution. Therefore, at each BB node we recompute the set of reachable vertices for each vertex and thus, also the limits on (p', q', r') given by such set.

$$\begin{aligned} p' &\leqslant 1 + |V_z|, \\ q' &\leqslant q_z + \sum_{i \in V_z} q_i, \\ r' &\leqslant r_z + \sum_{i \in V_z} r_i. \end{aligned} \tag{24}$$

The constraints on the cardinality of the partition, can also be rewritten in a tighter form as now they are computed using non-fixed vertices only.

Let z be the vertex acting as a source for the current partition. Define $m = |F_z|$ and let $r_z^m, q_z^m, p_z^m, r_x^{s-1}, q_x^{s-1}$, and p_x^{s-1} be computed as given in Equation (23). The values of p' are limited, as given in Equations (17), (18), and (24). Below they are specified when $s = 0$ and $s > 0$.

$$\begin{array}{ll} p_z \leqslant p' \leqslant \min\{1 + |V_z|, p - p_x\} & \text{if } s = 0, \\ p_z^m \leqslant p' \leqslant \min\{1 + |V_z|, p - p_x^{s-1}\} & \text{otherwise.} \end{array} \tag{25}$$

Assume that we have in hands a value of p' (number of vertices that must be supplied by z) satisfying Equation (25). We can further restrict the possible values of q' and r' as follows:

$$\begin{array}{l} q' \leqslant \min\{Q_{max}^m(z, p'), q - Q_{min}^s(x, p - p')\}, \\ q' \geqslant \max\{Q_{min}^m(z, p'), q - Q_{max}^s(x, p - p')\}, \\ r' \leqslant \min\{R_{max}^m(z, p'), r - R_{min}^s(x, p - p')\}, \\ r' \geqslant \max\{R_{min}^m(z, p'), r - R_{max}^s(x, p - p')\}. \end{array} \tag{26}$$

The values of $Q_{max}^m(z, p')$, $Q_{min}^m(z, p')$, $R_{max}^m(z, p')$, and $R_{min}^m(z, p')$ (as well as $Q_{max}^s(x, p - p')$, $Q_{min}^s(x, p - p')$, $R_{max}^s(x, p - p')$, and R_{min}^s) are obtained as follows:

$$\begin{array}{l} Q_{max}^m(z, p') = Q_{max}(z, p' - p_z^m) + q_z^m, \\ Q_{min}^m(z, p') = Q_{min}(z, p' - p_z^m) + q_z^m, \\ R_{max}^m(z, p') = R_{max}(z, p' - p_z^m) + r_z^m, \\ R_{min}^m(z, p') = R_{min}(z, p' - p_z^m) + r_z^m. \end{array} \tag{27}$$

4.4.3. *The modified recursion*

The modified recursion making use of all the constraints discussed above is then given by

$$f(p, q, r, x, s) = \begin{cases} \min\limits_{(p', q', r') \in \mathcal{S}_x(p,q,r)} \left[f\left(p - p', q - q', r - r', x, 0\right) \right. \\ \quad \left. + \min\limits_{\substack{z \notin F \cup \{x\} \\ (p', q', r') \in \mathcal{S}_z}} \left[f\left(p', q', r', z, 0\right) + g'_{xz}(r') \right] \right] & \text{if } s = 0, \\ \min\limits_{(p', q', r') \in S_x^s(p,q,r)} \left[f\left(p - p', q - q', r - r', x, s - 1\right) \right. \\ \quad \left. + f\left(p', q', r', y, m\right) + g'_{xy}(r') \right] & \text{otherwise,} \end{cases} \tag{28}$$

where F is the set of vertices already fixed as part of the final solution, $y = v_x(s)$ and $m = |F_y|$. $\mathcal{S}_x(p, q, r)$ and $\mathcal{S}_z$ are the sets defining the upper

and lower limits for the triplets (p', q', r') used when $s = 0$, and $\mathcal{S}_x^s(p, q, r)$ is the set of possible values for (p', q', r') when $s > 0$.

$$\mathcal{S}_x(p, q, r) = \{(p', q', r') \in \mathbb{N}_0^3 : p' \leqslant p - 1,\ q' \leqslant q - q_x,\ r' \leqslant r - r_x\},$$
$$\mathcal{S}_z = \{(p', q', r') \in \mathbb{N}_0^3 : p' \geqslant 1,\ q' \geqslant q_z,\ r' \geqslant r_z \text{ satisfying (21), (24), and(26)}\},$$
$$\mathcal{S}_x^s(p, q, r) = \{(p', q', r') \in \mathbb{N}_0^3 : \text{ satisfying (17), (18), (22), (24), and (26)}\}.$$

5. Computational Experiments

The BB algorithm was implemented in Fortran and computationally evaluated on a 200 MHz Pentium PC with 64 MB of RAM.

As in Fontes et al. (2005a,b), two types of cost functions were considered: polynomials of degree 1 (having linear variable and fixed cost components) and of degree 2 (having concave variable and fixed cost components). (We decided to choose polynomial functions, since any function can be easily approximated by a Taylor series.) For ease of notation, we refer to these cost functions as linear FC and concave FC, respectively. The problems used are available for download from the OR-Library (Beasley OR-L).

Type I: linear FC

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

Type II: concave FC

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

The cost function g_{ij} is nondecreasing and a_{ij}, b_{ij}, and c_{ij} are non-negative. Problems are divided in groups based on the ratio between variable and fixed costs. Ten groups for problems with 10, 12, 15, 17, and 19 vertices and five for problems with 25 and 30 vertices are considered. For each group we generated three problem instances. Therefore, computational experiments are carried out on 360 problem instances. To the best of our knowledge NFPs with concave FC costs have only been considered in Fontes (2000), Burkard et al. (2001) and Fontes et al. (2003, 2005a,b).

All tables give the average optimality gap associated with the best LB found at the root node, the average number of (BB) nodes solved in order to find an optimal solution, and the average time needed to solve them. For problems with 10, 12, and 15 vertices we also include the number of problems for which an optimal solution has been found at the root node. The averages are taken considering only problem instances for which an optimal solution has not been found at the root node.

Tables 1 and 2 summarize the results for problems with 10, 12, and 15 vertices using BB1 and BB2 for linear and concave FC NFPs, respectively.

For problems with 10 and 12 vertices, the results obtained for both BB procedures are very similar. However, this is no longer true for problems with 15 vertices. BB1 has a better practical behaviour (computational time),

Table 1. Average results for linear FC NFPs

			BB1		BB2	
Size	#Prob	Gap(%)	Nodes	Time	Nodes	Time
10	6	0.007	2	00:00:00	2	00:00:00
12	11	0.035	2	00:00:05	1	00:00:45
15	25	0.250	6	00:03:24	3	00:32:35

Table 2. Average results for concave FC NFPs

			BB1		BB2	
Size	#Prob	Gap(%)	Nodes	Time	Nodes	Time
10	1	0.006	2	00:00:00	2	00:00:00
12	9	0.55	2	00:00:12	1	00:00:11
15	25	3.42	17	00:20:19	9	01:09:59

while BB2 has a better theoretical behaviour (number of sub-problems). Thus, we developed a third version, referred to as BB3, that is a combination of BB1 and BB2. BB3 updates weights only if the number of vertices that have their supply route still undetermined is less than or equal to 15; otherwise the penalties are updated. The idea of BB3 is to establish a compromise between practical applicability and theoretical performance. The three versions were then used to solve problems with 17 and 19 vertices and the results are given in Table 3.

For concave FC NFPs with 19 vertices, instances in class 10 are not considered in the results reported for BB3, since no optimal solution was found within the time limit imposed. If only the first nine problem classes are considered, then the averages are 32 and 28 BB nodes and 1:32:55 and 2:44:16, for BB1 and BB2, respectively.

When concave FC costs are being considered the decrease obtained in the number of nodes by using BB3 rather than BB1 is small but the time increase is significant. BB2 performs worse than the other two versions.

Table 3. Linear and concave FC NFPs with 17 and 19 vertices

			BB1		BB3		BB2	
Size	Type	Gap(%)	Nodes	Time	Nodes	Time	Nodes	Time
17	FC	0.56	14	00:23:35	7	00:29:07	7	00:37:09
19	FC	0.76	15	00:46:26	9	01:11:01	19	00:55:30
17	Conc.	3.61	24	01:14:40	17	03:38:59	20	06:15:37
19	Conc.	4.36	47	02:10:11	40	04:05:02	40	04:08:41

Table 4. Linear and concave FC NFPs with 25 and 30 vertices

Size	Type	Gap(%)	Nodes	Time	Type	Gap(%)	Nodes	Time
25	FC	1.61	62	04:00:39	Conc.	7.94	367	13:58:28
30	FC	1.55	299	07:14:39	Conc.	5.30	892	18:27:06

Table 5. Linear and concave FC NFPs with 25 and 30 vertices, phase I

Size	Type	Gap(%)	Nodes	Time	Type	Gap(%)	Nodes	Time
25	FC	2.01	237	00:32:19	Conc.	9.49	1764	03:13:54
30	FC	1.70	586	05:51:38	Conc.	5.91	2802	12:22:49

If linear FC costs are under consideration, the time performance is very similar for all BB versions, but BB3 needs less nodes to be solved. The results also show that the increase in the time required to solve the problems as the problem size increases, is much larger for concave FC problems than for linear FC problems. This is expected, as concave FC problems are much "harder" than linear FC problems. Furthermore, although the development of the solution methodology is independent of the type of cost function being considered its performance is not. This is the main reason associated with the decision of using only procedure BB1 when solving the larger problems. Results for concave and linear FC NFPs with 25 and 30 vertices are reported in Table 4.

For problems with 25 and 30 vertices the results obtained in phase I were very similar to the ones obtained in phase III, particularly for FC problems, (see Fontes, 2000). Thus, in Table 5 we report results using at the root node the LB of phase I. Time requirements are much smaller since during phase I only the penalties are updated and hence, the total weight remains at zero. Note that by using these LBs and BB1 we are actually using the cardinality relaxation.

In Figures 3 and 4 the performance versus problem group is given for procedure BB1.

Although the performance varies with problem class, a pattern cannot be found. Thus, it can be concluded that the performance of our methodology in solving concave NFPs cannot be measured by the value of the ratio between variable and fixed costs, which is an important measure of the problem difficulty (Hochbaum and Segev, 1989; Ortega and Wolsey, 2003).

Table 6 reports on the performance of BB1 for linear FC and concave FC NFPs. For problems with 25 and 30 vertices we also report the results when at the root node only phase I is considered.

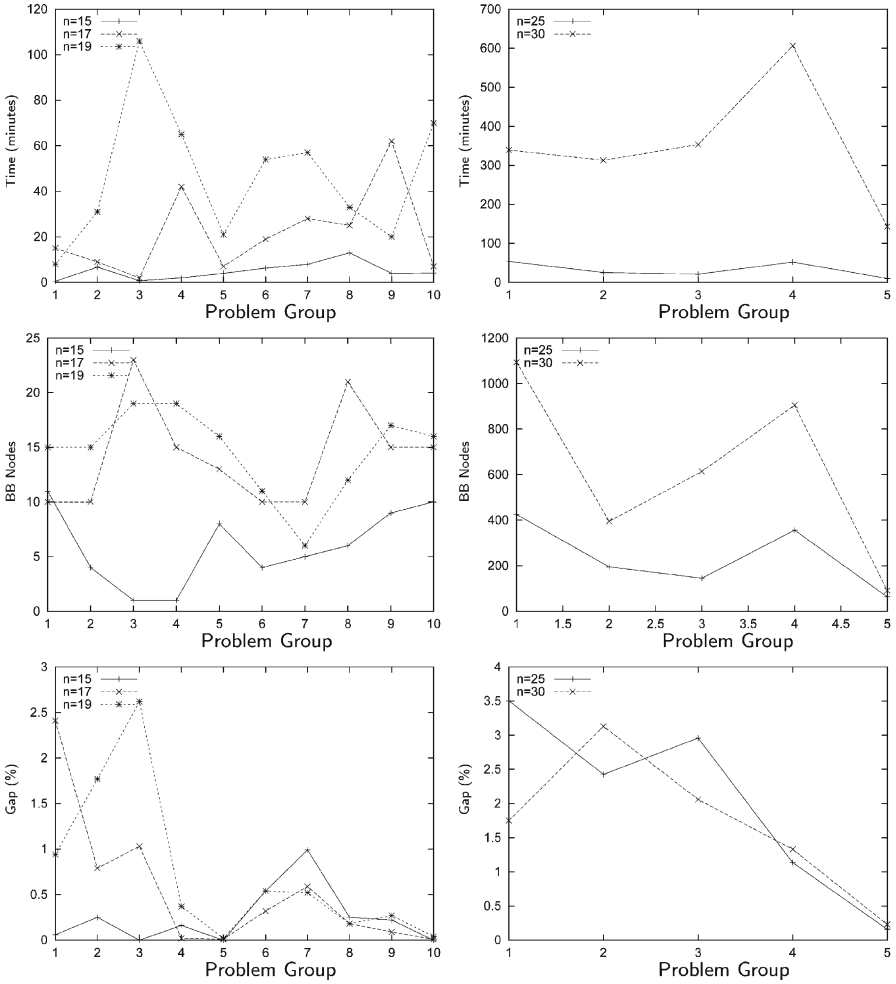


Figure 3. BB1 computational performance versus problem group for linear FC NFPs.

Figure 5 shows the effect of problem size on the computational time. On the left-hand side, time is shown in minutes, while on the right-hand side time is given on a logarithmic scale. As expected the number of sub-problems to solve and the computational time required by the BB algorithm increase with problem size. The rate of increase is higher in the case of concave FC costs. However, from the right hand side graph of Figure 5 it can be seen that the BB has a sub-exponential growth due to the concave curvature of the log scale plot.

Only Burkard et al. (2001) address SSU general concave NFPs, where all arcs have concave costs and all vertices are demand vertices. The method developed in Burkard et al. (2001) was tested on acyclic SSU flow networks

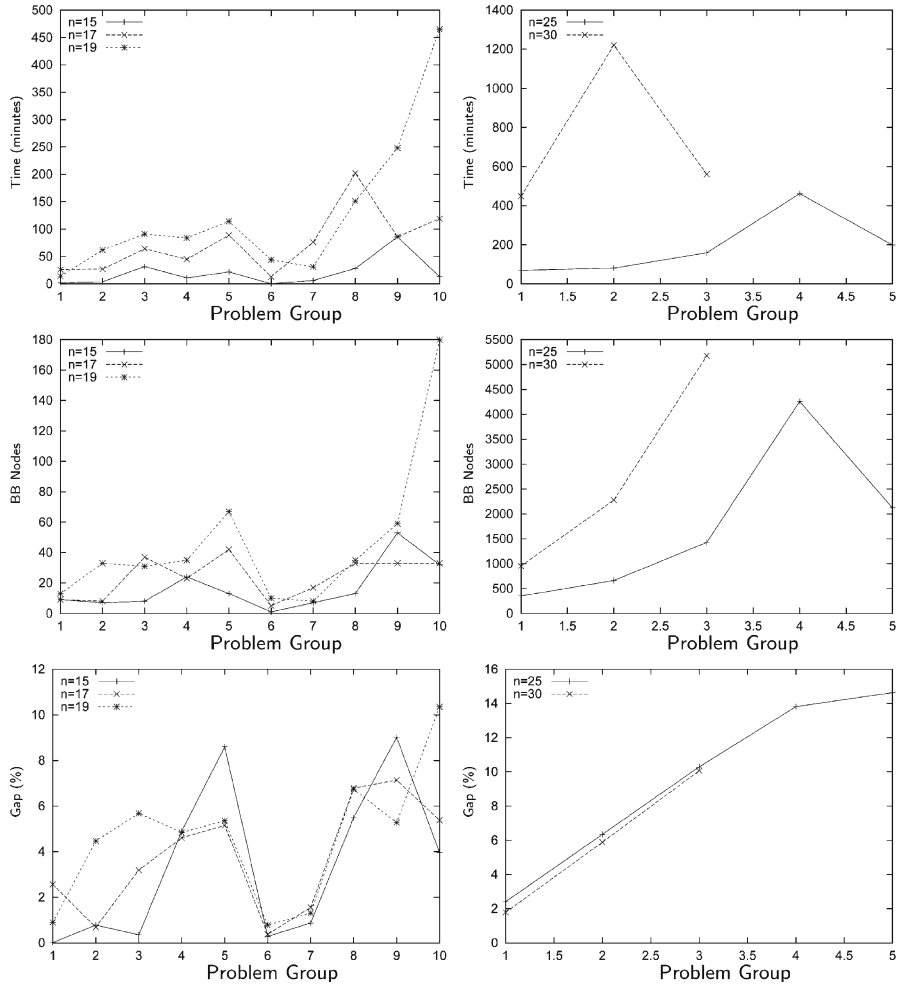


Figure 4. BB1 computational performance versus problem group for concave FC NFPs.

Table 6. Average results for linear and concave FC NFPs

Size	Gap(%)	Nodes	Time	Size	Gap(%)	Nodes	Time
10	0.007	2	00:00:00	10	0.006	8	00:00:00
12	0.04	2	00:00:05	12	0.45	2	00:00:12
15	0.25	6	00:03:24	15	3.42	17	00:20:19
17	0.56	14	00:23:35	17	3.61	24	01:14:40
19	0.76	15	00:46:26	19	4.36	47	02:10:11
25	1.61	62	04:00:39	25	7.94	367	13:58:28
25†	2.01	237	00:32:19	25†	9.49	1764	03:13:54
30	1.55	299	07:14:39	30	5.30	892	18:27:06
30†	1.70	586	05:51:38	30†	5.91	2802	12:22:49

† phase I.

Table 7. Comparison of the average computational time with the alternative methods

Size	Burkard et al. (2001)	Fontes et al. (2005a)	BB1
10	–	00:00:01	00:00:00
11	00:00:13	–	–
12	–	00:00:08	00:00:12
13	00:01:25	–	–
15	–	00:03:35	00:20:19
16	00:12:05	–	–
17	–	00:35:55	01:14:40
19	04:51:31	05:19:52	02:10:11
20	07:17:00	–	–
21	13:19:54	–	–
25	–	–	03:15:54
30	–	–	12:22:49

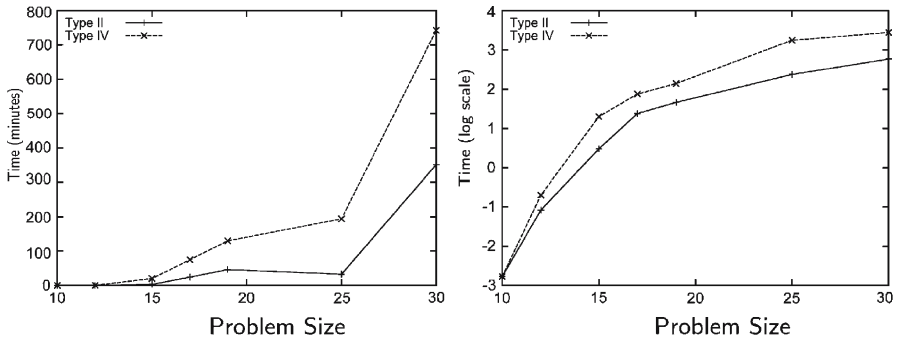


Figure 5. BB1 average computational time (minutes and log scale) for concave and FC problems.

and the computational results reported are produced in Table 7 together with the results we have obtained for the BB method presented in here and the DP method presented in Fontes et al. (2005a). Recall that, each figure reported in this table for Fontes et al. (2005a) and for BB1 has been obtained as an average out of 30 and 15 problem instances for smaller (with up to 19 vertices) and larger problem sizes, respectively. Regarding the results by Burkard et al. (2001) they are averages out of five problem instances for problems with up to 16 vertices, while only one problem instance has been solved for the remainder. In order to compare these three sets of results, they have also been plotted in Figure 6.

As it can be seen, for smaller size problems the BB takes longer than both Burkards' and the DP methods. However, this conclusion is reversed for larger problems, where the graph for the BB methodology is not as steep and therefore, its performance dominates. From the results above we

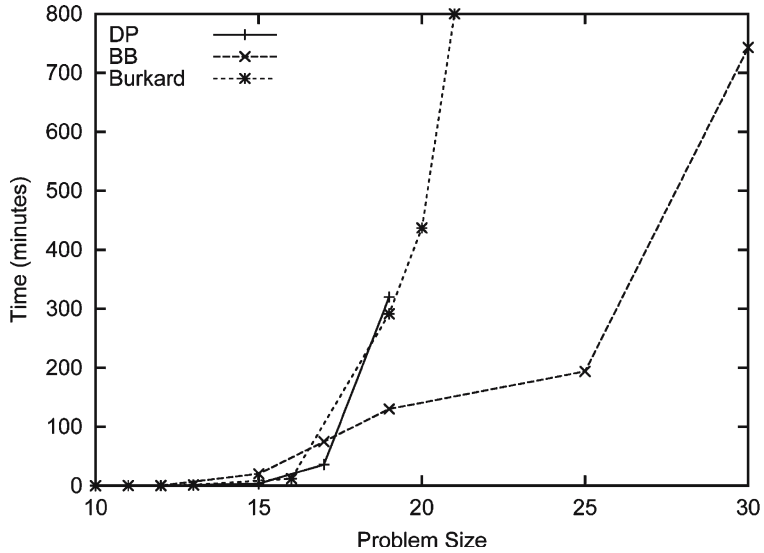


Figure 6. Comparison of the average computational time.

can also conclude that the computational time reported by Burkard et al. (2001) is similar to the one reported by Fontes et al. (2005a) but not as good as the one reported in this paper. Furthermore, although the quality of the solutions obtained in (Burkard et al. (2001) is very good they are not optimal.

6. Conclusions

In this paper, we give a BB algorithm to solve the SSU general concave NFP optimally. At each BB node a LB is found by using state space relaxations of a DP formulation also developed by the authors. The bounding procedure requires a modified version of the SSR given in Fontes et al. (2005b), since it is necessary to include information on the decisions already made. Thus, we have included an extra variable to represent the state space. This variable is associated with the number of fixed arcs emanating from the state vertex. The selection strategy is of the *adaptive* type and uses the LB information of active nodes. The node we choose is the one with the smaller LB value. The branching strategy is such that the arc chosen is amongst the ones used in the LB solution, since these arcs are more likely to be in the optimal solution. Two mutually exclusive and exhaustive decisions are made about arcs, namely: eliminating an arc from the final solution and forcing an arc to be in the final solution.

The procedure was tested on a set of randomly generated test problems, which are available for downloading from (Beasley, OR-L). Existing literature, with the exception of Burkard et al. (2001), Fontes et al. (2003, 2005a,b), considers simpler versions of the problem. For example, authors addressing concave NFPs do not consider simultaneously a concave variable cost and a fixed cost (Guisewite and Parados, 1991b), while authors considering a fixed-cost component consider only linear variable costs (Hochbaum and Segev, 1989). Some authors allow only a small percentage of the arc costs to be concave, the remaining being linear (Horst and Thoai, 1998), and others consider only some vertices to be demand vertices (Gallo et al., 1980; Guisewite and Pardalos, 1991b). Burkard et al. (2001) and Fontes et al. (2003) developed methodologies that are capable of finding good quality solutions. Nevertheless, they have developed approximate methods.

We compare the performance of our method with that of Burkard et al. (2001) and Fontes et al. (2005a). It can be concluded that the method described in here has better computational performance and can be used to solve larger size problems. At least when SSU NFPs, where all arcs have general non-linear concave costs and all vertices are demand vertices are, under consideration. The other two methods require a computational time that grows exponentially with the problem size. In contrast, we have shown evidence that the BB algorithm has a sub-exponential growth, making it the only real alternative to solve optimally larger size problems.

Appendix A: Description of the BB Procedure

It follows a detailed description of the BB procedure. Here, we are assuming that the root node has already been solved. Thus, the first step is to read the root node solution.

1. **Initialization**

   $A_f = \emptyset$ and $A_e = \emptyset$.

   Read Root Node

   (a) Read LB value and structure.
   (b) Read penalty and weight vectors.
   (c) Choose branching arc (x, y).
   (d) Store root node.

2. **Solve Right Son**

   (a) Eliminate arc (x, y), $A_e = A_e \cup \{(x, y)\}$.
   (b) Compute the number of fixed arcs outgoing from the source vertex (s), and compute the total number of fixed arcs $(m = |A_f|)$.

(c) If any customer is disconnected then

i. Eliminate node,
ii. GOTO 3.

(d) Compute $Z_{LB} = f(P, Q, R, t, s) - \sum_{i \in V} \lambda_i$ as in Equation (28).

(e) If the solution is feasible then

i. If $Z_{LB} < Z_{UB}$ then update Z_{UB}, best found so far,
ii. Eliminate node,
iii. GOTO 3.

(f) If $Z_{LB} > Z_{UB}$ then

i. Eliminate node,
ii. GOTO 3.

(g) If $Z_{LB} > Z^*_{LB}$ then update Z^*_{LB}

(h) If time/iterations are within limits then

i. Update penalties/weigths,
ii. GOTO 2(d).

(i) Store node data.

3. **Solve Left Son**

(a) Fix arc (x, y), $A_f = A_f \cup \{(x, y)\}$.

(b) Update the number of fixed arcs outgoing from the source vertex, Update the number of fixed arcs $m = m + 1$.

(c) If $m = n$ then

i. Compute the cost of the fixed structure, it is a feasible solution,
ii GOTO 3(e).

(d) Compute $Z_{LB} = f(P, Q, R, t, s) - \sum_{i \in V} \lambda_i$ as in Equation (28).

(e) If the solution is feasible then

i. If $Z_{LB} < Z_{UB}$ then update Z_{UB}, best found so far,
ii. Eliminate node,
iii. GOTO 4.

(f) If $Z_{LB} > Z_{UB}$ then

i. Eliminate node,
ii. GOTO 4.

(g) If $Z_{LB} > Z^*_{LB}$ then update Z^*_{LB}.

(h) If time/iterations are within limits then

i. Update penalties/weights,

ii. GOTO 3(d).

(i) Store node data.

4. **Select Node**

Select a sub-problem from the list of active sub-problems if one exits. Otherwise STOP.

5. **Retrieve Node Data**

(a) Retrieve penalty and weight vectors.
(b) Retrieve arcs fixed in and eliminated from the solution.

6. **Choose Branching Arc**

From all possible arcs choose arc (x, y).
GOTO 2.

References

Beasley, J.E. (OR-L), 'Or-Library', http://www.brunel.ac.uk/depts/ma/ research/jeb/info.html.

Burkard, R.E., Dollani, H. and Thach, P.H. (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.

Christofides, N., Mingozzi, A. and Toth, P. (1981), State space relaxation procedures for the computation of bounds to routing problems, *Networks,* 11, 145–164.

Cordier, C., Marchand, H., Laundy, R. and Wolsey, L.A. (1999), bc-opt: A branch and cut code for mixed integer programs, *Mathematical Programming,* 86, 335–353.

Fontes, D.B.M.M. (2000), Optimal Network Design Using Nonlinear Cost Flows, PhD thesis, The Management School, Imperial College of Science Technology and Medicine, London, U.K.

Fontes, D.B.M.M., Hadjiconstantinou, E. and Christofides, N. (2003), Upper bounds for single source uncapacitated minimum concave-cost network flow problems, *Networks,* 41, 221–228.

Fontes, D.B.M.M., Hadjiconstantinou, E. and Christofides, N. (2005a), A dynamic programming approach for solving single-source uncapacitated concave minimum cost network flow problems, *European Journal of Operational Research,* in Press.

Fontes, D.B.M.M., Hadjiconstantinou, E. and Christofides, N. (2005b), Lower bounds from state space relaxations for concave network flow problems. *This Journal*.

Gallo, G., Sandi, C. and Sodini, C. (1980), An algorithm for the min concave cost flow problem, *European Journal of Operational Research,* 4, 249–255.

Guisewite, G.M. (1994), Network problems, In Horst, R. and Pardalos, P.M. (eds.), *Handbook of Global Optimization,* Kluwer Academic, Dordrecht, pp. 609–648.

Guisewite, G.M. and Pardalos, P.M. (1991a), Algorithms for the single-source uncapacitated minimum concave-cost network flow problem', *Journal of Global Optimization,* 3, 245–265.

Guisewite, G.M. and Pardalos, P.M. (1991b), Global search algorithms for minimum concave-cost network flow problems, *Journal of Global Optimization,* 1, 309–330.

Held, M., Karp, R.M. and Crowder, H.P. (1974), Validation of subgradient optimization, *Mathematical Programming,* 6, 62–88.

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

Horst, R. and Thoai, N.V. (1998), An integer concave minimization approach for the minimum concave cost capacitated flow problem on networks, *OR Spectrum,* 20, 45–53.

Kim, D. and Pardalos, P.M. (1999), A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure, *Operations Research Letters,* 24, 195–203.

Kim, D. and Pardalos, P.M. (2000a), A dynamic domain contraction algorithm for nonconvex piecewise linear network flow problems, *Journal of Global Optimization,* 17, 225–234.

Kim, D. and Pardalos, P.M. (2000b), Dynamic slope scaling and trust interval techniques for solving concave piecewise linear network flow problems, *Networks,* 35, 216–222.

Kim, H.-J. and Hooker, J. (2002), Solving fixed-charge network flow problems with a hybrid optimization and constraint programming approach, *Annals of Operations Research,* 115, 95–124.

Lamar, B.W. (1993), A method for solving network flow problems with general non-linear arc costs, In Du, D.-Z. and Pardalos, P.M. (eds.), *Network Optimization Problems,* World Scientific, Singapore.

Ortega, F. and Wolsey, L.A. (2003), A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem, *Networks,* 41, 143–158.

Zangwill, W.I. (1968), Minimum concave cost flows in certain networks, *Management Science,* 14, 429–450.