International Journal of Pure and Applied Mathematics
Volume 40 No. 3 2007, 321-340

OPTIMAL FLOW TREES FOR NETWORKS WITH GENERAL NONLINEAR ARC COSTS

Dalila B.M.M. Fontes
Faculdade de Economia
Universidade do Porto – LIADD
Rua Dr. Roberto Frias, Porto, 4200-464, PORTUGAL
e-mail: fontes@fep.up.pt

Abstract: This paper describes the application of a dynamic programming approach to find minimum flow cost spanning trees on a network with general nonlinear arc costs. Thus, this problem is an extension of the Minimum Spanning Tree (MST) problem since we also consider flows that must be routed in order to satisfy user needs. In fact, the MST, usually, considers fixed arc costs and in our case the arc cost functions are nonlinear, having in addition to the fixed cost a flow dependent component. The arc cost functions involved may be of any type of form as long as they are separable and additive. This is a new problem, which is NP-*Hard* and a dynamic programming approach was developed to solve it exactly for small and medium size problems. We also report computational experiments on over 1200 problem instances taken from the *OR-Library*.

AMS Subject Classification: 90C27, 90C35, 90C39
Key Words: dynamic programming, network flows, optimal trees, general nonlinear arc costs

1. Introduction

In this paper, we propose to solve a new problem that consists of finding a flow tree spanning all vertices of a given network. The network consists of a central service provider designated by source vertex and a set of other vertices designated by user vertices, since they have a positive demand. In the network all vertices are connected to other vertices by a set of arcs, each of which hav-

Received: July 24, 2007

ing associated a nonnegative and nonlinear cost function and routing capacity limits. Therefore, we wish to find a tree rooted at the source vertex satisfying all user needs, i.e. flow requirements and satisfying arc flow capacity limits at minimum total cost.

This problem has similarities both with the minimum spanning tree problem and with the single-source minimum cost network flow problem.

It differs from the first one by considering that user needs have values, other than 1 unit. Thus, flows must be routed through the arcs in order to provide the aforementioned needs. Another difference is that, in addition to the arc weights, usually considered, we also consider arc costs that are nonlinearly dependent of the routed flows.

Regarding the single source minimum cost network flow problem, we search only amongst tree-shaped solutions. However, we consider general nonlinear arc cost functions, an extension to previous research. It should be noticed that if all arc costs are linear or concave and no capacity constraints exist then the problem proposed coincides with such a minimum cost network flow problem. This happens since for such a problem an optimal solution is known to be a tree, see [30].

1.1. Minimum Cost Spanning Tree Problem

A tree is a very important graph theoretic concept that is a basic structure in a variety of combinatorial problems, such as network flow algorithms. Furthermore, it also represents a fundamental element in a large number of graphic theoretical problems.

The search for an optimal tree arises in a number of applications, both as a stand-alone problem and as a subproblem in a more complex problem setting. In some direct applications, we wish to connect a set of points using the least cost or least length collection of arcs, that is we wish to find the minimum cost spanning tree. This happens, for instance, in designing physical systems, such as:

1. Connecting terminals in cabling the panels of electrical equipment using the least possible length of the wire.

2. Constructing a digital computer system, composed of a high-frequency circuitry, when it is important to minimize the length of wires between components to reduce both the capacitance and delay effects.

3. Constructing a network to collect or distribute a product. Pipe network for delivering a fluid (water or gas) from a reservoir to the users. Cable network connecting central distribution points to users (electricity or telecommunica-

tions). Sewers network to collect waste from buildings to a central treatment plant.

In all of the above examples, we want to bring connections together and aggregate products. The arcs in the network have different physical requirements and hence, different operation and construction costs. A detailed study on optimal trees can be found in [25].

The Minimum Spanning Tree (MST) problem is one of the best-known combinatorial optimization problems. Given a finite connected graph, the MST problem is defined as a minimum cost subgraph spanning all vertices and having no cycles.

Several variations of this problem have also been studied. For example, there might be other constraints imposed on the design such as the number of vertices in a subtree (generalized problem), [8], the number of arcs in any path (hop-constrained problem), [15, 18], the number of arcs between any pair of vertices (diameter-constrained problem), [13, 24, 7, 14], type of services available on the arc or vertex, [26], amongst many others.

The complexity of the MST problem increases with the number of vertices. When the network's arc costs are fixed and the search is unconstrained the problem can be solved in polynomial time and many algorithms have been developed, starting back in the 50's with Kruskal and Prim and in the 70's with Dijkstra. However, new and more complex variations keep appearing based on real application problems, mainly from the telecommunications due to the advances of optical technologies and the explosive growth of Internet, and thus motivating further research. For a recent survey see [20].

1.2. Single-Source Minimum Cost Network Flow Problem

Network flow problems are very common since many practical situations arise where a product is routed through some sort of network. For example traffic may flow on a street network, water or gas may flow on a pipe network. Many other practical applications can be found in a variety of fields, see [16] and references therein. Although many other network flow problems exist, here we are interested in single source minimum cost ones.

Given a directed graph we wish to select a subset of arcs, of the set of all potencial arcs, such that the source vertex can supply all other vertices in the vertex set, with the required demand. Usually, many such solutions exist amongst which we wish to choose the minimum cost one. It should be noticed that, for each arc used both a fixed cost and a flow routing cost are incurred and that the the total cost is given by the sum of arc costs. Typically, there

are capacity constraints associated with each arc that impose a limite on the amount of flow that may be routed through each arc.

The main feature defining the complexity of Minimum Cost Network Flow problems (MCNF) is the type of cost function for each arc. In this sense, this type of problems can be divided into four categories with increasing complexity, namely: linear, convex, concave, and general nonlinear.

Linear MCNF, have constant marginal arc costs and can be solved in polynomial time, see [3]. Convex MCNF, which have nondecreasing marginal arc costs, involve the minimization of a convex objective function over a convex feasible region, defined by the network constraints. Therefore, a local optimum is also a global optimum. Although harder than linear MCNF, these problems are still "easy to solve", see [3].

Concave MCNF have nonincreasing marginal arc costs and are much harder than the previous MCNF. The complexity of this type of problems arises from minimizing a concave function over a convex feasible region, which implies that a local optimum is not necessarily a global optimum. Concave MCNF are known to be NP-*hard*, [17], even for the simplest version – single source uncapacitated with fixed-charge arc costs. However, they do exhibit some special mathematical properties that make them more tractable than general nonlinear MCNF, see [16]. For a recent discussion on general concave MCNF, see for example [6, 10, 9] for approximate methods and [12, 11] for exact methods, and references therein.

General nonlinear MCNF have arc costs that are neither convex nor concave thus no convexity or concavity properties can be explored in the determination of an optimal solution. Therefore, this type of MCNF is much harder to solve. However, this type of costs can be found in many practical problems, since they typically involve network construction costs as well as operation costs. The latter ones are, usually, a function of the flow routed through the arcs, sometimes involving some sort of discontinuity. For example, transport of passengers typically includes a fixed cost proportional to the number of vehicles (fuel, drivers, insurance) and thus, if the flow is in number of passengers, then the cost function is in the form of a "staircase". Another type of cost function, known as "sawtooth" may arise in goods distribution as costs are usually made up of a marginal cost, that typically decreases with quantity and a fixed cost that can introduce discontinuities at quantity "breakpoints". A cost function that is concave up to a certain value and convex afterwards may also appear in production settings, for example due to market reaction for demand of a raw material.

To the best of our knowledge, no optimization methods have been reported

in the literature for general nonlinear MCNF. Existing literature considers only specific types of cost functions, namely staircase [5, 22, 23] and sawtooth [23] cost functions. Lamar in [23] proposes to address the MCNF with sawtooth or staircase arc costs by transforming the problem into a concave MCNFP on an expanded network. However, as the resulting MCNF is defined over a much larger network and thus, this transformation is only useful for very small problems. Rech and Barton [27] developed a branch-and-bound algorithm for nonconvex transportation problems with piecewise linear costs. They use the out-of-kilter method to solve linear approximations obtained by convex envelope. Bornstein and Rust [5] change their approach to address MCNF with staircase cost functions. Recently, Kim and Pardalos [22] obtained good quality solutions for nonconvex piecewise linear MCNF. They solve linear problems that are recursively updated by using the previous solution. At each iteration the feasible domain is reduced by a contraction rule based on the work of Thakur [29]. A recent survey on piecewise linear MCNF, including staircase and sawtooth cost functions is given by Kim in [21].

We propose a methodology based on dynamic programming to solve the problem of finding an optimal tree on a single source flow network with general nonlinear arc costs and arc capacity limits. As far as the authors are aware of, the problem proposed here as never been addressed before, with the exception of the work in [28], where the authors consider a uncapacitated version. In [28] the authors developed a genetic algorithm to find optimal trees on uncapacitated networks that can be used with any type of cost function. However, the method has only been tested on two sets of cost functions: one considering the usual fixed cost associated to each used arc and another considering a concave routing cost only.

In the sections following, we explain the DP formulation and its implementation, before reporting on the computational experiments performed. The generation of the test problems used is briefly explained, since they can be downloaded from Beasley. As it will be seen, the DP implementation is based on the idea of gradually expanding the state space graph. A very efficient backward-forward procedure is used. The results have shown the method to be rather robust, since its performance does not depend of the type or form of the cost functions considered neither depends of the number of nonlinear arc costs. It can also be seen that the algorithm searches only a small part of the state space. For capacitated problems, and depending on the constraints tightness, it can be very efficient. However, for uncapacitated problems the performance of the method deteriorates rapidly with problem size.

2. Problem Formulation

Given a network we wish to find a tree which connects user vertices to the central service provider located in the root of the tree. Each user vertex may represent an individual user or a collection of individual users connected by an underlying network. Furthermore, if a connection between two vertices exists the demand that we may route through it must be within certain predefined limits. Many such trees exist and among them we wish to find the minimum total cost one. The total cost includes both a fixed cost and a routing cost. The latter ones are usually given by a nonlinear function.

Let $G = (W, A)$ be a directed network, where W is a set of $n + 1$ vertices (vertices 1 to n are the user vertices and vertex $t1$ is the central service provider) and A is a set of m directed arcs. The user vertices have associated a nonnegative integer demand r_x and the supply available at the source vertex R matches the total demand required by them, that is $R = \sum_{x=1}^{n} r_x$. The demand that can be routed through each arc (x, z) has upper u_{xz} and lower l_{xz} limits. A general nonlinear and nonnegative demand routing dependant cost $g_{xz}(r)$ is assigned to each arc and satisfies $g_{xz}(0) = 0$.

Before introducing the dynamic programming formulation, and just for the sake of exposition we give here the mathematical programming formulation for the problem being solved:

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

subject to:

$$\sum_{(k,i)\in A} x_{ki} - \sum_{(i,k)\in A} x_{ik} = r_i\,, \quad \text{for all } i \in W\,, \tag{2}$$

$$x_{ij} \geq 0\,, \quad \text{for all } (i,j) \in A\,. \tag{3}$$

Equation (1) states the minimum cost nature of the problem. Constraints (2) are the flow conservation constraints. The demand of the central service provider, i.e. root vertex, is given by $r_t = -R = -\sum_{x=1}^{n} r_x$.

3. The Dynamic Programming Approach

Our Dynamic Programming (DP) algorithm requires as input a single source directed network G, the demands of the user vertices, and for each arc the capacity limits and total cost function (i.e. arc construction and operation

costs). It will provide us with a minimum cost tree that spans all vertices and satisfies user demands and capacity limit constraints. Therefore, the output consists of the used arcs and of the amount of demand routed through each of these arcs, and the cost of such a tree. Let as call this tree a Minimum Flow Cost Spanning Tree (MFCST).

We first explain how the problem of finding a MFCST is divided into subproblems. Then we show how to solve these subproblems by using the solutions of already solved subproblems. Finally, we show how to retrieve the solutions structure.

3.1. Dynamic Programming Formulation

DP (see [2, 4]) solves a given problem by first, dividing it into subproblems which are then solved. Then a solution to the given problem is obtained by combining the solutions to the already solved subproblems into solutions to bigger subproblems until the original problem is solved.

Subproblems. A subproblem of the original problem of finding an optimal tree in G, as defined above, consists of finding a MFCST that supplies a subset S of user vertices of the original set $S \subseteq V$ from a specified root vertex, say $x \in S$, satisfying user demands and capacity limit constraints. Given S and x, we denote the corresponding subproblem by the (S, x), a minimum flow cost spanning tree by $T(S, x)$ and its cost by $f(S, x)$. For each subproblem (S, x) the DP algorithm divides set S into two subsets $\left\{S', \bar{S}'\right\}$, where $S' \subseteq S \setminus \{x\}$ and $\bar{S}'$ is the complement of S' in the set S, that is $\bar{S}' = S \setminus S'$.

A crucial point of DP is that the solutions to already solved subproblems are stored and not re-computed. Therefore, for each such partition it needs to compute and store the following:

— The value $f(\bar{S}', x)$ of the tree $T(\bar{S}', x)$ and information on how it was computed,

— The value $f(S', z)$ of the tree $T(S', z)$ and information on how it was computed, for all possible $z \in S'$.

Let us know write the DP model. Consider a subset $S \subseteq V$ and a vertex $x \in S$. For each possible set $S' \subseteq S \setminus \{x\}$, let $z \in S'$ be the root vertex of the tree $T(S', z)$.

The minimum cost of supplying set S' from vertex $x \notin S'$ with the required commodity r made available at some vertex $z \in S'$ satisfying the capacity constraints, i.e. $l_{xz} \leq r \leq u_{xz}$, is found by determining the best combination of the minimum flow cost directed tree of S' rooted at vertex $z \in S'$, $T(S', z)$

with the cost of arc (x, z), that is

$$\min_{\substack{z \in S' \\ l_{xz} \leq r \leq u_{xz}}} \left\{ f\left(S', z\right) + g_{xz}\left(r\right) \right\}.$$

By definition $f\left(\bar{S}', x\right)$ gives the minimum cost incurred in supplying the remaining demand vertices of set S not in S', through a directed tree rooted at x. Therefore, the DP recursion is given by

$$f(S, x) = \min_{S' \subseteq S \setminus \{x\}} \left[f\left(S - S', x\right) + \min_{\substack{z \in S' \\ l_{xz} \leq r \leq u_{xz}}} \left[f(S', z) + g_{xz}(r) \right] \right] \tag{4}$$

and is applied to all $S \subseteq W \setminus \{t\}$ and all $x \in S$. Hence, the cost of an optimal flow tree supplying all demand vertices in set W from the source vertex t, if one exists, is given by equation (5), which is initialized by equation (6).

$$f^* \equiv f(W, t) = \min_{S' \subseteq W \setminus \{t\}} \left[f\left(W - S', t\right) + \min_{\substack{z \in S' \\ l_{tz} \leq \sum_{i \in S'} r_i \leq u_{tz}}} \left[f(S', z) + g_{tz}\left(\sum_{i \in S'} r_i \right) \right] \right], \tag{5}$$

$$f\left(S, x\right) = \begin{cases} 0, & \text{if } S = \{x\} \\ \infty, & \text{otherwise.} \end{cases}, \tag{6}$$

It should be noticed that the DP model derived is independent of the type and form of the cost functions considered and of the number of nonlinear demand routing costs. Also, the cost functions may be neither differentiable nor continuous, having only to be separable and additive.

3.2. Implementation of the DP Algorithm

The implementation closely follows the reasoning behind the DP idea of dividing the problem to be solved into subproblems, which are then solved. Then, the solution to the given problem is obtained by combining the solutions to the already solved subproblems into solutions to larger subproblems until the original problem is solved. Therefore, the algorithm starts from the final state (W, t), i.e. the given problem, and while moving backward visits, without computing, possible states, i.e. possible subproblems. The procedure keeps moving backward through states, i.e. keeps dividing the current problem into smaller

subproblems, not yet computed until a state already computed is reached. At this point the algorithm has reached a subproblem already solved. Then, the procedure is performed in reverse order. Starting from the state last identified in the backward process, i.e. a solved subproblem, it goes forward through computed states using them to compute other possible states needed. The algorithm uses subproblems already solved in order to solve larger subproblems by combining their solutions. This process goes on until we reach a subproblem not yet solved. At this point, again it goes backward until a solved subproblem is reached. This procedure is repeated until the final problem (W, t) is found with a value that cannot be bettered by any other alternative solution, that is, the best solution for the given problem has been obtained.

We have implemented the DP formulation based on the idea of gradually expanding the state space graph using the above described backward-forward procedure on each layer of the state space. Its main advantage is that the expansion of the state space graph is based upon the information relevant to the part of the graph which has already been generated. Therefore, states which are not feasible for the problem are not computed, as only states which are needed for the computation of the solution are considered (this can be observed through the decrease achieved for the computational time requirements when capacity constraints become tighter, see the results section). As will be shown latter, on average only 9% to 11% of the states are computed for capacitated problems, where only up to 50% of the total demand can be routed through a single arc.

3.3. DP Algorithms

In the DP recursion of equation (4), a state is represented by (S, x) and consists of two state variables: the set variable S and the root variable x. The set S refers to the combination of vertices that have already been included in the current state and x to the vertex that supplies this subset, $x \in S$. In our model there are $n + 1$ stages, $1, 2, \ldots, n + 1$, where n is the number of user vertices. A stage k contains all states $s_i \equiv (S, x)$ such that $|S| = k$. Thus, being at any state of stage k of this multistage decision process implies that k vertices have already been included in the tree.

In the implementation being used, the state space is represented by a two-dimensional matrix where the first dimension represents the set variables while the second represents the root variables. In this way, the access to a specific state is direct and there is no need for array searching. There is, of course, some memory waste because not all states (S, x) exist.

The flow of the algorithm is managed by Algorithm 1, that after labelling all states (subproblems) as not yet computed, it initialize states in stage 1, that is subproblems involving sets with cardinality 1, as given by equation (6). After that it calls Algorithms 2 and 3 with parameters (W, t). Algorithm 2 is a recursive algorithm that computes the optimal solution cost. For each subproblem, this algorithm also stores the partition (subset) and vertex acting as source (to this subset), associated with its best solution. Algorithm 3 is also a recursive algorithm and it backtracks through the information stored while solving subproblems, in order to retrieve the solutions structure, i.e. arcs and flows.

Algorithm 1: DP for finding MFCST.

Input: A single source t directed network, the vertices demand, the capacity limits, and the cost functions
Label all states as not yet computed
Initialize states at stage one in equation (6)
Call $f(W, t)$
Output: Solution cost
Call $Arsc\&Flows(W, t)$
Output: Solution structure

At the end of Algorithm 1, if $f(W, t) = \infty$ then no tree exists that can supply the user vertices with the required demand, satisfying the flow limits; otherwise $f(W, t)$ gives the cost associated with such an optimal tree.

The complexity of the DP algorithm is, in the worst case, of the order $\mathcal{O}(n2^n)$. As expected, the complexity increases exponentially with the number of user vertices. It should be noticed, that the algorithm expected order is much lower, since many subproblems are not feasible and also due to the existence of capacity constraints. On the other hand, the DP model can be used to find optimal flow trees on networks with arbitrary cost functions without deteriorating its performance. As the computational results show, the algorithm behaviour is independent of the type, form, and number of nonlinear arc costs.

Algorithm 2: Recursive function: compute optimal cost of $T(S, x)$.

Recursive $Compute(S, x)$
if $f(S, x) \neq \infty$ **then**
  return $f(S, x)$ to caller
end
Set $min = \infty$
for *each* $S' \subseteq S$ **do**
  Call $Compute(S \setminus S', x)$
  if $f(S \setminus S', x) \geq min$ **then**
    get another S'
  end
  for *each* $z \in S'$ **do**
    if $(x, z) \notin A$ **then**
      get another z
    end
    $r = \sum_{i \in S'} r_i$
    if $r > u_{xz}$ *or* $r < l_{xz}$ **then**
      get another z
    end
    if $f(S \setminus S', x) + g_{x,z}(r) \geq min$ **then**
      get another z
    end
    Call $Compute(S', z)$
    if $f(S \setminus S', x) + g_{x,z}(r) + f(S', z) \geq min$ **then**
      get another z
    end
    $min = f(S \setminus S', x) + g_{x,z}(r) + f(S', z)$
    Store information on: *subset*=S', *vertex*=z, and
    $f(S, x) = min$.
  end
end
Return: $f(S, x)$

Algorithm 3: Recursive function: retrieve used arcs and arc flows, $T(S, x)$.

Recursive $Arsc\&Flows(S, x)$
if $f(k, x) \neq 0$ **then**
  $S' = subset(S, x)$
  $r = \sum_{i \in S'} r_i$
  $z = vertex(S, x)$
  $Flow(x, z) = r$
  CALL $Arsc\&Flows(S \setminus S', x)$
  CALL $Arsc\&Flows(S', z)$
end
Return: $Flow$

4. Computational Experiments

The algorithm presented in this paper was implemented in *Fortran* and computationally evaluated on a 1500MHz, *Pentium PC* with 256 MB of RAM by solving a set of randomly generated test problems. In order to evaluate its performance we have considered three problem types, which are amongst the most difficult problems as all arcs have cost functions that are neither convex nor concave. Next we describe the random problems used and then we give the computational results obtained for these sets of problems.

4.1. Generation of Test Problems

Three different types of cost functions are considered: type G1 and type G2 are variations of the fixed-charge cost function where discontinuities other than at the origin are introduced and type G3, for which we consider that arc costs are initially concave and then convex having a discontinuity at the break point. Types G1 and G2 correspond, respectively, to the so called staircase and sawtooth cost functions, see [21], in our case with two segments. Type G3 corresponds to a cost function that is initially concave and then convex.

$$g_{ij}(r) = \begin{cases} 0, & \text{if } r = 0, \\ -a_{ij}r^2 + b_{ij}r + c_{ij} & \text{if } r \leq \bar{R}, \\ a_{ij}r^2 + b_{ij}r + c_{ij} + k & \text{otherwise,} \end{cases}$$

where $a_{ij} = 0$ for G1 and G2, $k = b_{ij}$ for G1, and $k = -b_{ij}$ for G2 and G3.

The problems data can be downloaded from the *OR-Library* (Beasley) and a thorough description of the generation procedure is provided in [10]. For each type of cost function five groups (groups 1 to 5 in [1]) of problems were considered, each group containing three problem instances of the same size. Problem group is mainly defined by the expected ratio between the variable cost and the fixed cost (V/F), which was set to 0.01, 0.1, 1, 2, and 10. The reason behind the groups division is that the V/F ratio is an important measure of problem difficulty, as previously found by Hochbaum and Segeve [19]. Problems with extreme V/F ratio values are 'easier' since either the fixed-cost or the routing cost is dominant, while for intermediate values a balance between these two costs must be achieved. Problem size, given by the number of user vertices vertices, was set to 10, 12, 15, 17, and 19.

Two values have been considered for the discontinuity point, thus $\bar{R}$ was set to either 30% or to 50% of the total demand R. When arc capacity limits are being considered, the limitation must occur at a value $\hat{R} > \bar{R}$, since otherwise the cost functions would become either fixed-charge, for cost types G1 and G2, or concave, for cost type G3. To show better evidence and for comparison purposes only, we have also considered MFCSTs with linear and with fixed-charge costs.

In order to be able to consider a larger number of possible values for $\hat{R}$, we chose $\bar{R}$ to be 30% of the total demand R, for the experiments with capacitated problems. The values chosen for $\hat{R}$ have been 50%, 70%, and 90% of R.

4.2. Computational Results

The algorithm performance was evaluated for all three problem types. Recall that a problem of a certain type (G1, G2, or G3) is a problem for which all arc costs are of the designated type.

Two measures of performance were computed for each problem: Time-the computational time required to find an optimal solution; and S_{comp}-the percentage[1] of the state space that is actually computed by the DP algorithm.

In order to show even stronger evidence on the methods performance independence of the cost function type we also report results for the same set of problems when considering linear ($b_{ij}r$) and fixed-charge cost functions ($b_{ij}r + c_{ij}$).

Two values have been considered for the discontinuity point. In Tables 1 and 2 we summarize the results obtained for uncapacitated problems involving cost functions of types G1, G2, and G3 by considering that the discontinuity

[1] $S_{comp} = \frac{\text{no. of states computed}}{\text{total no. of states}} \times 100\%$.

point occurs at 30% and 50% of R, respectively. Table 3 summarizes the results obtained for uncapacitated problems involving linear and fixed-charge arc costs, since these are not affected by the change in the discontinuity point value.

	Time (s)			S_{comp}(%)		
N	G1	G2	G3	G1	G2	G3
10	0.07	0.21	0.21	24.00	24.00	23.59
12	1.40	1.53	1.47	23.35	23.36	22.85
15	56.73	49.60	42.40	22.60	22.60	22.37
17	439.80	440.93	429.13	22.69	22.69	22.42
19	4205.87	4031.47	3990.20	21.03	21.06	20.95

Table 1: Computational performance for uncapacitated problems with $\bar{R}$=30% R and cost function types G1, G2, and G3

	Time (s)			S_{comp}(%)		
N	G1	G2	G3	G1	G2	G3
10	0.14	0.21	0.21	24.00	24.00	23.45
12	1.40	1.40	1.33	23.34	23.37	22.79
15	64.40	51.47	52.80	22.70	22.61	22.30
17	438.73	439.47	424.93	22.69	22.69	22.35
19	4022.87	4091.33	3863.93	21.05	21.06	20.93

Table 2: Computational performance for uncapacitated problems with $\bar{R}$=50% R and cost function types G1, G2, and G3

The results reported show that a significant reduction in the state space enumeration has been achieved for all problems, since only an average of 20% to 24% of states have to be computed. On the other hand, the computational time increases rapidly with problem size. Nevertheless, as expected, the computational time is independent of the cost function type, even when comparing general nonlinear costs to linear costs. This can also be seen from the variation of the discontinuity point, since in this case the performance of the algorithm is unaffected.

It should be noticed that the figures shown in these tables were obtained as averages over 15 problem instances of a given problem size and cost function type. Thus, overall we have solved 600 uncapacitated MFCSTs.

A graphical representation of the average computational time, both in minutes and in logarithmic scale, needed to solve the uncapacitated problems is given in Figure 1.

	Time (s)		$S_{comp}(\%)$	
N	Lin	FC	Lin	FC
10	0.21	0.14	22.33	24.00
12	1.40	1.47	22.61	23.35
15	51.47	55.60	22.19	22.60
17	410.00	432.20	22.37	22.69
19	3665.87	4044.20	20.89	21.06

Table 3: Computational performance for uncapacitated problems with linear and fixed-charge cost functions

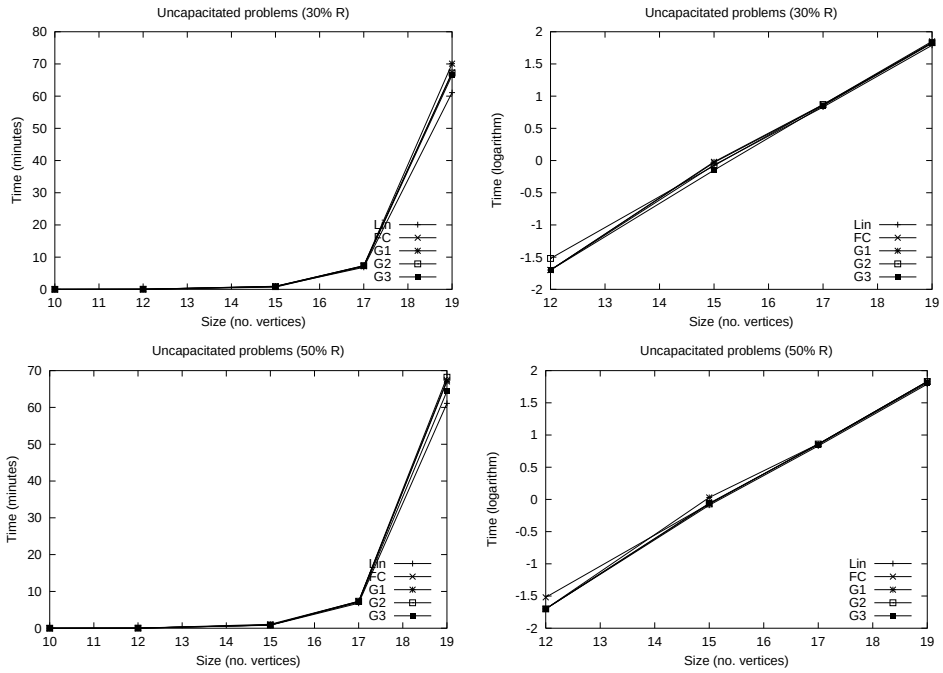

Figure 1: The effect of problem size on computational time for uncapacitated problems

Next, we analyse the behaviour of the DP algorithm when solving capacitated problems. In order to be able to test it on several values of the constraints tightness, here and hereafter we consider that the discontinuity point occurs at 30% of the total demand. The computational performance can be observed in the graphs plotted in Figure 2.

The results reported show an even larger state space reduction in the state

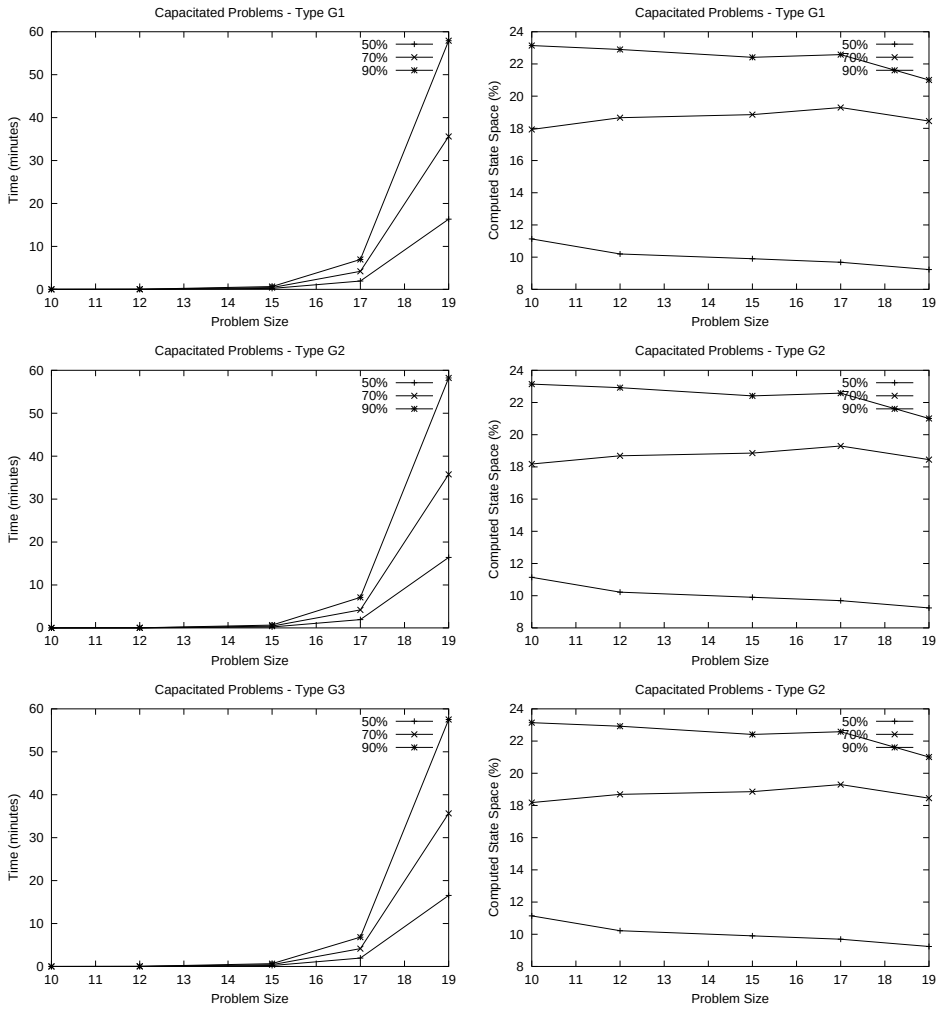

Figure 2: Computational performance for capacitated problems with cost types G1, G2, and G3

space enumeration, now only an average of 9% to 23% of states have to be computed. This is expected, since due to the capacity constraints several states are now infeasible. The computational time for the capacitated problems has a slower increase with problem size, again this is due the capacity constraints. Furthermore, this trend is stronger when the capacity constraints are tighter. The reasoning behind such trend is that not only the the percentage of state space to be computed decreases with the tightness of the capacity constraints,

but also because the magnitude of the decrease increases with problem size.

All figures plotted are averages out of the 15 problem instances solved. However, for problems with 19 vertices and capacity constraints of 50% of R, the results refer to 14 instances, since for one problem instance no tree satisfying the capacity constraints exists. Overall, we have solved 675 problem instances, since we solved 15 problem instances for each type and cost function type. In addition, each of these problems has been solved for three capacity limits, i.e. 50%, 70%, and 90% of the total demand required.

5. Conclusions

In this paper we have presented a DP methodology to solve a new problem. In this problem we seek for a minimum flow cost spanning tree on networks involving general nonlinear costs. In fact, the cost functions may be neither differentiable nor continuous. Also, they might be neither convex nor concave having only to be separable and additive.

Not many works have been reported in literature involving networks with nonlinear arc costs that are neither convex nor concave. The works found, although searching for any solution structure, address only staircase and sawtooth cost functions. More recently, [28] developed a genetic algorithm for searching for optimal trees on networks. However, the authors only test the algorithm using either fixed costs or concave routing costs.

The algorithm implementation is based on the idea of gradually expanding the state space graph using a backward-forward procedure on each layer of the state space. One of its main advantages is that the expansion of the state space graph requires information relating only to the part of the graph which has already been generated. A large number of randomly generated test problems of varying size and complexity was used to evaluate the algorithms performance. Overall, computational experiments were carried out on 1275 problem instances of which 600 are unconstrained and 675 are constrained. The results have shown the DP algorithm to be effective at searching optimal flow trees on single source networks with any type of cost function. Furthermore, it is also efficient but only for small and medium size problem instances, since computational requirements grow rapidly with problem size. Nevertheless, once capacity constraints have been considered the computational requirements growth became more modest. A major advantage of the methodology proposed in this work, is that it independent of the cost function type and form, as well as, of the number of nonlinear arcs, which have been shown to be the major factors

defining problem complexity, see [6, 19].

Acknowledgments

The financial support of FCT, POCI 2010 and FEDER, through project POCI/EGE/61823/20
is gratefully acknowledged.

References

[1] J.E. Beasley, Or-Library, http://www.brunel.ac.uk/depts/ma/research/jeb/info.html.

[2] R. Bellman, *Dynamic Programming*, Princeton University Press (1957).

[3] D.P. Bertesekas, *Network Optimization: Continuous and Discrete Models*, Athena Scientific (1998).

[4] D.P. Bertesekas, *Dynamic Programming and Optimal Control*, Athena Scientific (2005).

[5] C.T. Bornstein, R. Rust, Minimizing a sum of staircase functions under linear constraints, *Optimization*, **19** (1988), 181-190.

[6] R.E. Burkard, H. Dollani, P.H. Thach, 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** (2001), 121-139.

[7] H. Cancela, L. Petingi, On the characterization of the domination of a diameter-constrained network reliability model, *Discrete Applied Mathematics*, **154** (2006), 1885-1896.

[8] C. Feremans, M. Labbe, G. Laporte, A comparative analysis of several formulations for the generalized minimum spanning tree problem, *Networks*, **39** (2002), 29-34.

[9] D.B.M.M. Fontes, J.F. Gonçalves, Heuristic solutions for general concave minimum cost network flow problems, *Networks*, **50** (2007), 67-76.

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

[11] D.B.M.M. Fontes, E. Hadjiconstantinou, N. Christofides, A dynamic programming approach for solving single-source uncapacitated concave minimum cost network flow problems, *European Journal of Operational Research*, **174** (2006), 1205-1219.

[12] D.B.M.M. Fontes, E. Hadjiconstantinou, N. Christofides, Lower bounds from state space relaxations for concave network flow problems, *Journal of Global Optimization*, **34** (2006), 97-125.

[13] L. Gouveia, T.L. Magnanti, Network flow models for designing diameter-constrained minimum-spanning and steiner trees, *Networks*, **41** (2003), 159-173.

[14] L. Gouveia, T.L. Magnanti, C. Requejo, An intersecting tree model for odd-diameter-constrained minimum spanning and steiner trees, *Annals of Operational Research*, **146** (2006), 19-39.

[15] L. Gouveia, C. Requejo, A new Lagrangean relaxation approach for the hop-constrained minimum spanning tree problem, *European Journal of Operational Research*, **132** (2001), 539-552.

[16] G.M. Guisewite, Network problems, In: *Handbook in Global Optimization* (Ed-s: R. Horst, P.M. Pardalos), Kluwer Academic (1994), 506-648.

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

[18] R. Hassin, A. Levin, A minimum spanning tree with hop restrictions, *Journal of Algorithms*, **48** (2003), 220-238.

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

[20] H. Kerivin, A.R. Mahjoub, Design of survivable networks: A survey, *Networks*, **46** (2005), 11-21.

[21] D. Kim, Piecewise linear network flow problems, In: *Encyclopedia of Optimization* (Ed-s: C.A. Floudas, P.M. Pardalos), Kluwer Academic Publisher (2003).

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

[23] B.W. Lamar, A method for solving network flow problems with general nonlinear arc costs, In: *Network optimization Problems* (Ed-s: D.-Z. Du, P.M. Pardalos), World Scientific (1993).

[24] Y. Lee, M. Atiquzzaman, Least cost heuristic for the delay constrained capacitated minimum spanning tree problem, *Computer Communications*, **28** (2005), 1371-1379.

[25] T.L. Magnanti, L.A. Wolsey, Optimal trees, In: *Network Models* (Ed-s: M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser), Chapter 9, Elsevier (1995).

[26] R. Montemanni, L.M. Gambardella, A benders decomposition approach for the robust spanning tree problem with interval data, *European Journal of Operational Research*, **161** (2005), 771-779.

[27] P. Rech, L.G. Barton, *A Non-Convex Transportation Algorithm*, The English Universities Press (1970).

[28] D.K. Smith, G.A. Walters, An evolutionary approach for finding optimal trees in undirected networks, *European Journal of Operational Research*, **120** (2000), 593-602.

[29] L.S. Thakur, Domain contraction in nonlinear programming: Minimizing a quadratic concave objective over a polyhedron, *Mathematics of Operations Research*, **16** (1991), 390-407.

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