An Efficient Time and Space *K* Point-to-Point Shortest Simple Paths Algorithm

ANTONIO SEDEÑO-NODA (asedeno@ull.es)

Departamento de Estadística, Investigación Operativa y Computación (*DEIOC*). *Universidad de La Laguna, CP 38271- La Laguna, Tenerife* (*España*)

Abstract. We address the problem for finding the *K* best point-to-point simple paths connecting a given pair of nodes in a directed network with arbitrary lengths. The main result in this paper is the proof that a tree representing the *k*th point-to-point shortest simple path can be obtained by using one of the previous (*k*-1) trees representing each one of the previous (*k*-1) best point-to-point shortest simple paths. The proof requires that, in each iteration, at most *n* single-source shortest path computations (re-optimizations) in a network with non-negative length arcs are needed. In the "optimistic" case, this strategy only needs O(*m*) time to compute the best "neighbor" associated with a path tree, that is, the second shortest simple path for a given shortest simple path. The algorithm runs in $O(Knf(n,m,C_{\max}))$ time and uses O(*K*+*m*) space to determine the *K* point-to-point shortest simple paths in a directed network with *n* nodes, *m* arcs and maximum absolute length $C_{\max}$. $O(f(n,m,C_{\max}))$ is the best time needed to solve the shortest simple paths connecting a source node with any other non-source node in a network with non-negative length arcs. We provide a clear improve on the space needed in Yen's algorithm by a multiplicative factor of $O(n^2)$ for each best solution. Moreover, a version of our algorithm using only O(*Kn* + *m*) space runs in an "optimistic" case in $O(Kf(n,m,C_{\max}))$ time. This affirmation is confirmed by an experimental study where O(*K*) shorted paths are used to determine the *K* point-to-point shortest simple paths in both versions of our algorithm.

Categories and Subjecy Descriptors: G.2.2. [**Discrete Mathematics**]: Graph Theory-Networking
General terms: Algorithms, Design, Theory.
Additional Key Words and Phrases: Point-to-point Shortest paths, *K* best solutions.

1. *Introduction*

The *point-to-point simple shortest path* (PPSSP) problem in a directed network of *n* nodes and *m* arcs with arbitrary lengths on the arcs finds a shortest length path from a source node to a sink node or detects a cycle of negative length. Many important real cases of this problem appears and the numerous algorithms to solve it are addressed in Ahuja et al. [1] among others.

The *K point-to-point shortest simple paths* (KPPSSP) problem determines the *K* best solutions of the PPSSP problem. The problem to determine the *K* shortest paths in a network has a wide range of applications (see Eppstein [7] for example). An extended bibliography of

This research has been partially supported by Spanish Government Research Project MTM2006-10170.

several K best shortest path problems collected by Eppstein is available at <http://www.ics.edu/~eppstein/bibs/kpath.bib>. We chronologically cite the papers of the literature considering only the K shortest simple (loopless) paths problem: Hoffman and Pavley [12], Pollack [18], Yen [22] and [23], Lawler [14], Katoh et al. [13], Perko [17], Brander and Sinclair [3], Martins et al. [16], Hadjiconstantinou and Christofides [10], Martins and Pascoal [15], Carlyle and Wood [4] and Hersberger et al. [11]. The best bound to solve this problem in directed networks is reached in the early paper of Yen [22]. Yen's [22] *deviation* algorithm runs in $O(Kn\, f(n,m,C_{\max}))$ where $O(f(n,m,C_{\max}))(\geq O(m))$ is the best bound to solve the PPSSP problem in a directed network and uses $O(Kn^2+m)$ space. In addition, a number of papers have been proposed that refer to several practical improvements to Yen's algorithm, however, none has succeeded in improving the worst-case asymptotic time and space complexity of the problem (see [3, 11, 15, 16, 17]). The algorithm given in Carlyle and Wood [4] needs only $O(m)$ space requirements (they ignore the space required to write out the enumerated paths), but its running times is $O(Kn\, f(n,m,C_{\max})(\log n+\log C_{\max}))$. However, when the KPPSSP problem appears as sub-problem of a more complicated problem can be necessary to store explicitly the best point-to-point simple paths or alternatively to mantain the way to easy re-compute these paths (see Eppstein [7]). For example, read/write file operations often require a large amount of time while keeping an implicit representation of the K best algorithms in RAM can be most useful. On the other hand, the Carlyle and Wood [4] algorithm is only valid for networks with non-negative integer lengths (this algorithm uses a binary search on an integer value that allows it to determine the K near-shortest simple point-to-point paths). Moreover, additional practical improvements can only be used in networks with non-negative length (modification A) and when paths containing cycles are allowed (modification D). We also consider the Hersberger et al. [11] algorithm based on a *replacement paths* algorithm. Hersberger et al. [11] claim that their algorithm runs in an "optimistic" case in $O(Kf(n,m,C_{\max}))$ time when the replacement paths algorithm does not fail (only two shortest path computations are needed in each right execution replacement path algorithm). Even so, in the "optimistic" case at least seven shortest paths computations are needed for each new discovered path.

For undirected networks, Katoh et al. [13] introduce an $O(K\, f(n,m,C_{\max}))$ time and $O(Kn+m)$ space algorithm. This algorithm was efficiently implemented in Hadjiconstantinou and Christofides [10].

The K point-to-point shortest paths problem in which paths are not required to be simple are easier. The best algorithm for this problem is due to Eppstein [7]. The algorithm of Eppstein [7] computes an implicit representation of the K paths in $O(m+n\log n+K)$ time and $O(K+m)$ space. Each path can be output in order in $O(n+\log K)$ additional time, therefore, the K shortest paths can be enumerated in order by Eppstein algorithm [7] in $O(m+n\log n+K(n+\log K))$. Clearly, this algorithm can be used to determine the K point-to-point shortest simple paths in a network without directed cycle since any path in an acyclic network is a simple path. The algorithm given by Sedeño-Noda [21] runs in $O(m+n\log n+K(n+\log\frac{K}{n}))$ time using $O(K+m)$ space improving the bounds in [7] when an explicit enumeration in order of the best solutions is required.

In this paper, we prove that a tree representing the kth point-to-point shortest simple path can be obtained using one of the previous (k-1) trees representing each one of the previous (k-1) best point-to-point shortest simple paths. This result is achieved by showing that in each iteration, at most n single-source shortest path computations (re-optimizations) in a network with non-negative length arcs are needed. In the "optimistic" case, the proposed scheme only needs $O(m)$ time to compute the best "neighbor" associated with a tree. That is, we introduce an ad-hoc procedure to compute the second point-to-point simple path for a given point-to-point shortest simple path. This procedure takes advantage of the structural relations between the shortest path tree and the second best path tree. Hence, our algorithm is not a deviation algorithm and it does not use a replacement paths algorithm as a subroutine. In any case, we propose an $O(Kn\,f(n,m,C_{\max}))$ time and $O(K+m)$ space algorithm to determine the K point-to-point shortest simple paths in a directed network with n nodes, m arbitrary length arcs and maximum absolute length $C_{\max}$. Note that $O(f(n,m,C_{\max}))$ is the best time needed to solve the shortest simple paths connecting a source node with any other non-source node in a network with non-negative length arcs. In each iteration of our method, the kth best point-to-point shortest simple path is calculated and then, the way to determine the second best point-to-point simple path associated with the kth best solution is computed and stored. Note that this approach improves the space needed in Yen's algorithm by a multiplicative factor of $O(n^2)$ for each best solution. Moreover, a version of our algorithm using only $O(Kn+m)$ space can run in the "optimistic" case in $O(K\,f(n,m,C_{\max}))$ time. That is, when the current k best solutions are stored, the determination of the (k+1)*th* best solution using our ad-hoc

procedure requires in the "optimistic" case $O(m)$ time plus only one shortest path computation. This affirmation is confirmed with an experimental study where $O(K)$ shorted paths computations are used to determine the *K* point-to-point shortest simple paths for both versions of the algorithm. However, our algorithm does not improve the bounds for the undirected network. That is, when a directed version of the undirected network is considered, our algorithm works as in the directed case. In the case that the network is acyclic, our algorithm runs in $O(K\, f(n,m,C_{\max}))$ using only $O(K+m)$ space. Clearly, Eppstein [7] and Sedeño-Noda[21] are better in this case, but when the acyclic property is unknown a priori, our algorithm takes advantage of this situation while other "general" algorithms contemplated in the literature do not.

After this introduction, in section 2, the linear programming formulation of the PPSSP problem and the *K* point-to-point shortest simple path problem are given. In section 3, we introduce the main theoretical results, which the algorithm is based on. Section 4 contains a detailed pseudo code and an explanation of the proposed algorithm. Additionally, the worst-case time and space theoretical complexity of the algorithm is proven. Section 5 includes a report on the computer results of our method from a subset of the experiment carried out using the networks generator provided in Cherkassky *et al.* [5]. Finally, in section 6, we offer our conclusions.

2. *The point-to-point shortest simple path problem and the K point-to-point shortest simple paths problem.*

Given a directed network *G* = (*V, A*), let $V = \{1,...,n\}$ be the set of *n* nodes and *A* be the set of *m* arcs. For each arc $(i,j) \in A$, let $c_{ij} \in \mathbb{R}$ be its length and $C_{\max} = \max_{(i,j)\in A} \{|c_{ij}|\}$. The network has two distinguished nodes: the *source* node *s* and the *sink* node *t*. We denote by $\Gamma_i^- = \{ j \in V \mid (j,i) \in A \}$ for all node $i \in V$. We assume without loss of generality that the directed network *G* does not contain any arc emanating from sink node *t*. Note that in any case, a simple path from *s* to *t* does not use arcs emanating from *t*. Similarly reasoning allows us to suppose that the directed network *G* does not contain any arc arriving at source node *s*.

Let $i, j \in V$ be two distinct nodes of *G* = (*V, A*), we define a simple path p_{ij} as a sequence $\{i_1,(i_1,i_2),i_2,\cdots,i_{l-1},(i_{l-1},i_l),i_l\}$ of non-repeated nodes and arcs satisfying $i_1 = i$, $i_l = j$ and for all $1 \le w \le l-1$, $(i_w, i_{w+1}) \in A$. A directed simple cycle is a simple path such that the only

repeated nodes are i_1 and i_l (p_{ii}). The directed network *G* is acyclic if it does not contain any directed (simple) cycle. The length of a directed path *p* is the sum of the arc lengths in the path, that is, $c(p) = \sum_{(i,j)\in p} c_{ij}$. The point-to-point shortest simple path (PPSSP) problem consists in finding a shortest simple path from node *s* to node *t* or in determining a negative cycle, that is, a directed cycle of negative length.

If a flow x_{ij} is associated with each arc (i, j) then the following linear programming problem represents the PPSSP problem (see Ahuja et al. [1]):

$$\text{Minimize} \quad c(x) = \sum_{(i,j)\in A} c_{ij} x_{ij} \qquad (1)$$

subjet to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ 0 & \text{if } \forall i \in V - \{s,t\} \\ -1 & \text{if } i = t \end{cases} \qquad (2)$$

$$x_{ij} \geq 0, \qquad \forall (i, j) \in A \qquad (3)$$

The above problem is a special case of the minimum cost network flow (MCNF) problem. The network simplex algorithm can be used to solve the above problem by taking advantage of the fact that any basis of the MCNF problem is a spanning tree $T \subseteq A$ of *G*. Let *X* be the convex polyhedron defined by constraints (2)-(3) (*decision space*). The following two literature results hold (Ahuja et al. [1]): (*i*) *Any feasible solution of the PPSSP problem is a vertex of X and vice-versa* and (*ii*) *Every vertex of X is associated with a directed spanning tree rooted at s*.

A *directed out-spanning tree* is a spanning tree rooted at node *s* such that the unique path in the tree from node root *s* to every other node is a directed path. Note that in this kind of tree, each node $i \in V \setminus \{s\}$ has only one node predecessor in the tree ($pred_i(T)$), that is, its in-degree is one. In the rest of paper, we refer to a directed out-spanning tree as tree.

Distance labels of the nodes (negative dual variables) corresponding to a tree *T* are obtained by setting $d_s(T) = 0$ and solving $c_{ij} + d_i(T) - d_j(T) = 0$, $\forall (i, j) \in T$. Thus, given a tree *T*, we define the reduced cost $\overline{c}_{ij}(T) = c_{ij} + d_i(T) - d_j(T)$, $\forall (i, j) \in A$.

Let $C(T) = \sum_{(i,j)\in T} c_{ij}x_{ij}$ be the objective function value of the tree *T*. Note that $c(x) = d_t(T) = C(T)$. Therefore, minimizing $c(x)$ is equal to finding the shortest simple path from node *s* to node *t*. Additionally, since each node $j \in V \setminus \{s\}$ has only one node predecessor, we define $x_j(T) = x_{pred_j(T)j}$. Note that if $(i,j) \in T$ and $t \in D_j(T)$, then $x_{ij} = 1$ and $x_j(T) = 1$, otherwise, if $(i,j) \in T$ and $t \notin D_j(T)$ then $x_{ij} = 0$ and $x_j(T) = 0$. We also define $D_i(T)$ to be the set of descendants of node *i* in the tree *T*, that is, the set of nodes in the sub-tree rooted at *i*, including node *i*.

In a T-*exchange*, an arc $(i,j) \in A \setminus T$ with reduced cost $\bar{c}_{ij}(T)$ and $i \notin D_j(T)$ is added to *T* and $(pred_j(T), j)$ is deleted from *T* yielding a new tree T'. Once a T-exchange is performed, the distance labels in T' are updated in the following way: $d_k(T') = d_k(T) + \bar{c}_{ij}(T)$, $\forall k \in D_j(T)$. Furthermore, the objective function value is $d_t(T') = d_t(T) + \bar{c}_{ij}(T)x_j(T)$ since $x_j(T) = x_j(T')$.

For an optimal tree T^*, we obtain the following optimality conditions: $\bar{c}_{ij}(T^*) \geq 0,\ \forall (i,j) \in A$. Hereinafter, we consider that the length of any arc $(i,j) \in A$ is $\bar{c}_{ij}(T^*) \geq 0$ instead of c_{ij}, since determining the shortest path tree with length arcs *c* is equivalent to determining the shortest path tree with reduced cost length arcs $\bar{c}(T^*)$ in *G* (Ahuja et al. [1]). Therefore, once an optimal tree T^* is obtained, we can use the Dijkstra [8] algorithm to determine possible alternative paths in *G* from node *s* to node *t* considering non-negative lengths. Therefore, without loss of generality, we consider $c_{ij} = \bar{c}_{ij}(T^*)$ for all $(i,j) \in A$ in the rest of the paper.

The *K* point-to-point shortest simple paths problem consists in determining the *K* best different solutions of the problem (1)-(3). In other words, if we denote by $p_{st}(T)$ the path tree from node *s* to node *t* in the tree *T* then, the problem require identifying the *K* best trees T^k with different $p_{st}(T^k)$ for $k \in \{1,...,K\}$ such that $d_t(T^1) \leq d_t(T^2) \leq ... \leq d_t(T^K)$ and for any other tree T^p with $p_{st}(T^p) \neq p_{st}(T^k)$ for all $k \in \{1,...,K\}$ is $d_t(T^p) \geq d_t(T^K)$.

3. *Main Theoretical Results*.

In this section, we introduce and prove the basic results to the efficient resolution of the *K* point-to-point shortest simple path problem. We begin by introducing the following definition:

Definition 1. Two trees *T* and T' are adjacent if and only if both have $n-2$ arcs in common, that is, both trees differ in only one arc.

Therefore, the tree T' can be built from the tree *T* by a T-exchange where the entering arc is just the arc $(i,j) \in T' \setminus T$ and $(p,q) \in T \setminus T'$ is the leaving arc. In addition, if the path tree from node *s* to node *t* in T' must be different to the path tree from node *s* to node *t* in *T*, then the entering arc $(i,j) \in T' \setminus T$ must satisfy $x_j(T) = 1$, that is, node *j* must belong to the path tree from node *s* to node *t* in *T*. Moreover, let *T* and T' be two trees that differ in the $p < n$ arcs. Then the following property given in Sedeño-Noda and González-Martín [20] holds:

Proposition 1. If T and T' differ in p < n arcs, where E = $\{(i_1, j_1), \ldots, (i_p, j_p)\}$ are the arcs in T' that are not in T, then: (1) *E does not contain a directed cycle;* (2) $j_u \neq j_v$ *holds for all* $u, v \in \{1, \ldots, p\}$ *with* $u \neq v$ *;* (3) *These arcs define the smallest T-exchange sequence to obtain T' from T, and the order in which these T-exchanges are performed is irrelevant..*

Given any tree *T*, we call a *multiple T-exchange* to the operation where $p < n$ arcs satisfying proposition 2 are entered simultaneously in *T*. The following results given in Sedeño-Noda [21] holds:

Lemma 1. Given a tree T of an acyclic directed network G, let T' be the tree containing a different path tree from node s to node t that is obtained from T by making a multiple T-exchange with the arcs $\{(i_1, j_1), (i_2, j_2), \ldots, (i_p, j_p)\}$ *with non-negative reduced cost and with* $2 \leq p < n$. *Then* $d_t(T')$ *is greater than or equal to the distance label of the node t of at least one of the p trees that can be obtained by a T-exchange with only one arc in the set* $\{(i_1, j_1), (i_2, j_2), \ldots, (i_p, j_p)\}$.

Given a tree T, we denote by $A(T)$ a subset of arcs of $A\backslash T$ and $\Gamma_i^-(A(T)) = \{ j \in V \mid (j,i) \in A(T) \}$ (the set of predecessor nodes of node i in the directed graph $(V, A(T))$. Lemma 1 establishes that if network G is acyclic and T contains the best s-t path in the directed graph $(V,\ A(T) \cup T)$, the second best s-t path is obtained from T by a T-exchange with the entering arc $(i,j)_{A(T)} = \arg\min_{(u,l) \in A(T)} \{ \overline{c}_{ul}(T) : u \notin D_l(T) \text{ and } x_l(T) = 1 \}$. Then, for any node j with $x_j(T) = 1$, we also define $i_j^* = \arg\min_{i \in \Gamma_j^-(A(T))} \{ \overline{c}_{ij}(T) : i \notin D_j(T) \}$. Therefore, $(i,j)_{A(T)} = \arg\min_{j \in p_{st}(T)} \{ \overline{c}_{i_j^* j}(T) \}$.

Given a tree T and a non-tree arc $(i,j)_{A(T)}$, we obtain the basis tree T' with $p_{st}(T') \neq P_{st}(T)$. But, if network G is not acyclic, $p_{st}(T')$ could not be the second best s-t path in the directed graph $(V,\ A(T) \cup T)$. Clearly, the second best s-t path in this graph contains the best sub-path from j to t for some j with $x_j(T) = 1$. Therefore, we must identify an alternative path from node s to each node j with $j \in p_{st}(T)$ not using arcs $(i,j) \in A(T)$ with $x_j(T) = 1$ and $i \notin D_j(T)$, that is, the nodes and arcs that can not be considered for a fixed node $j \in p_{st}(T)$ are:

(1) $(u,j) \in A(T)$ with $u \notin D_j(T)$ (non-tree arcs) and $(pred_j(T), j)$.

(2) Any node belonging to $p_{j't}(T)$ where j' is the next node to node j in $p_{jt}(T)$. Or equivalently , any arc arriving at any node in $p_{j't}(T)$ and any arc leaving from any node in $p_{jt}(T)$ (including node j since we search for paths from node s to node j).

Thus, we must identify an alternative path from node s to node j with $x_j(T) = 1$ using some descendant node(s) of node j not included in $p_{jt}(T)$. For example in Figure 1, several paths arriving at node $j \in p_{st}(T)$ can be found using the arc (i,j) satisfying $i \in D_j(T)$. They are $s \to i_3 \to i_2 \to i \to j$, $s \to i_3 \to i_4 \to i_2 \to i \to j$ and $s \to i_3 \to i_4 \to i_1 \to i_2 \to i \to j$. For a fixed node $j \in p_{st}(T)$, we denote by $B_j(T)$ the set of arcs satisfying (1) and (2) and we set $\overline{B}_j(T) = A(T) \cup T - B_j(T)$ the arcs in $A(T) \cup T$ that can be used to find an alternative path from node s to node j.

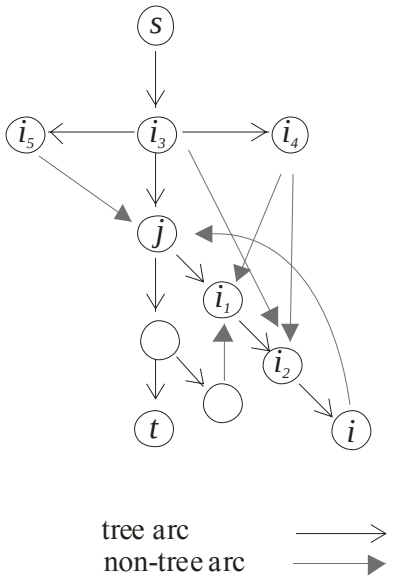

Figure 1a. Initial tree arcs and non-tree arcs.

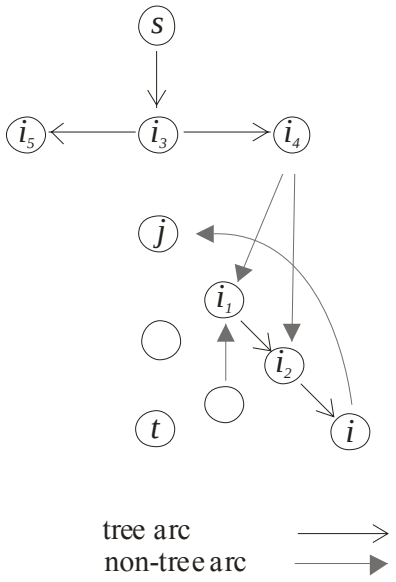

Figure 1b. Tree arcs and non-tree arcs to determine the alternative path from node *s* to node *j*.

Note that for a fixed node *j* with $x_j(T)=1$, for any node $v \in D_j(T)$ with $x_v(T)=0$ among all arcs (u,v) arriving at *v* with $u \notin D_j(T)$, we only must take into account those with the least reduced cost. For example in Figure 1a, any sub-path to node *j* crossing node i_2 from any node *u* with $u \notin D_j(T)$ ($u=\{i_3,i_4\}$) always uses the arc with the least reduced cost, that is, (i_3,i_2) or (i_4,i_2). In Figure 1b) for node *j*, the arcs $\bar{B}_j(T)$ appear with the exception of the arc (i_3,i_2), because we suppose that the arc (i_4,i_2) has the least reduced cost among the incoming arcs in node i_2. Then, we obtain the next result:

Lemma 2. Given a tree T, with a set of arcs $A(T)$ *with non-negative reduced cost, the best alternative path from node s to some node j with* $x_j(T)=1$ *using arcs in* $\bar{B}_j(T)$ *contains only one non-tree arc* (u,v) *with* $u \notin D_j(T)$ *and* $v \in D_j(T)$.

Proof. Let $l_{sj}(T)$ be the best alternative path $l_{sj}(T)$ to the tree path $p_{sj}(T)$ from node *s* to node *j* using arcs $\bar{B}_j(T)$. Note that $l_{sj}(T)$ must contain at least one non-tree arc (y,z) with $y \notin D_j(T)$ and $z \in D_j(T)$. Otherwise, an alternative path to the tree path $p_{sj}(T)$ from node *s* to node *j* does not exist. Let $l_{vj}(T) \subset l_{sj}(T)$ be the sub-path from node *v* to node *j* with $v \in D_j(T)$ containing only descendant nodes of node *j*. Clearly, the predecessor node of node

v in the path $l_{sj}(T)$ is a node u with $u \notin D_j(T)$. Therefore, $l_{sj}(T) = l_{su}(T) \cup (u,v) \cup l_{vj}(T)$ where (u,v) is a non-tree arc with $u \notin D_j(T)$ and $v \in D_j(T)$. Since a path tree from *s* to *u* exists with zero reduced costs and $l_{sj}(T)$ is the best alternative path to the tree path $p_{sj}(T)$, then the length of $l_{sj}(T)$ must be equal to the length of $p_{su}(T) \cup (u,v) \cup l_{vj}(T)$. Therefore, we have identified an alternative optimal path from node *s* to node *j* that uses only one arc (u,v) satisfying $x_v(T) = 0$, $u \notin D_j(T)$ and $v \in D_j(T)$. □

Note that lemma 2 holds for any node $u \in D_j(T)$. Then, we define $l^*_{sj}(T)$ to be the shortest path from node *s* to node *j* with $x_j(T) = 1$ using arcs in $\overline{B}_j(T)$.

Lemma 3. Given a tree T, with a set of arcs $A(T)$ with non-negative reduced cost, the best alternative path from node s to node j with $x_j(T) = 1$ not using the $(pred_j(T), j)$ arc and nodes in the $p_{jt}(T)$ with exception of j is the path with minimum length between paths $\{ p_{si}(T) \cup (i^*_j, j)$, $l^*_{sj}(T) \}$ using the arcs in $A(T) \cup T$.

Proof. Note that by lemma 1, $p_{si}(T) \cup (i^*_j, j)$ is the best alternative path to the path tree $p_{sj}(T)$ that does not use descendant nodes of node *j* with exception of *j* and does not use the arc $(pred_j(T), j)$. On the other hand, by definition, $l^*_{sj}(T)$ is the best alternative path using some nodes that are descendants of node j not in $p_{jt}(T)$ (with exception of *j*) and does not use the arc $(pred_j(T), j)$. Clearly, the best alternative path to the tree path $p_{sj}(T)$ that does not contain the arc $(pred_j(T), j)$ and nodes in $p_{jt}(T)$ with exception of *j* is the path with minimum length of these two alternatives. □

Note that if an alternative path to the tree path $p_{sj}(T)$ better than the alternative paths commented on lemma 3 exists, this alternative path uses the arc $(pred_j(T), j)$ and/or contains some nodes in $p_{jt}(T)$ additionally to node *j*. But in this case, we have identified an alternative path from node *s* to some node j' belonging to $p_{st}(T)$ not using $(pred_{j'}(T), j')$ arc neither nodes in $p_{j't}(T)$ with exception of j' with length equal to or less than the

alternative path to $p_{sj}(T)$. In this case, when we search for an alternative best path for the current path $p_{st}(T)$, we will obtain the same or best utility by determining the best alternative path from node *s* to j' instead of to node *j*.

Denote by $Ap_{sj}(T)$ the alternative path with minimum length between paths $\{ p_{si}(T) \cup (i_j^*, j)$, $l_{sj}^*(T) \}$ for a node *j* with $x_j(T) = 1$. Let $Ap_{sj^*}(T)$ be the alternative path with minimum length among $Ap_{sj}(T)$ paths with $j \in p_{st}(T)$. Then, we obtain the next result:

Theorem 1. Given a tree T, with a set of arcs $A(T)$ *with non-negative reduced cost. Let node* j^* *be the node determining the alternative minimum length path* $Ap_{sj^*}(T)$. *Let* $NT_{j^*}(T)$ *be the set of non-tree arcs in* $Ap_{sj^*}(T)$ *and let* T' *be the tree obtained from T making a* T-*exchanges sequence with the arcs in* $NT_{j^*}(T)$. *Then* T' *is the second best solution of the PPSSP problem in the network G=(V,* $A(T) \cup T$ *).*

Proof. Theorem holds since $Ap_{sj^*}(T) \cup p_{j^*t}(T)$ is the minimum alternative length path to the tree path $p_{st}(T)$. Therefore, when all the T-exchanges with the arcs in $NT_{j^*}(T)$ are made, the distance label of any node in the tree path from node j^* to node *t* increases in a minimum amount. Therefore, the increase in the distance label of node *t* is minimal and T' is the second best solution of the PPSSP problem in the network *G*=(*V*, $A(T) \cup T$). □

We are interested in generating the *K* best point-to-point shortest simple paths in order without repeating the calculation of the same best solution. For that, given a tree *T*, let NT_{j^*} be the set commented on in theorem 1. Let $(i, j^*) \in NT_{j^*}(T)$ and let T' be the tree obtained making a T-exchanges sequence with the arcs in $NT_{j^*}(T)$. Then, we set $A(T) = A(T) - \{(i, j^*)\}$ and $A(T') = A \setminus T' - \{(u,l) \in A : \forall l \in p_{j^*t}(T')\}$. Note that $A(T')$ does not contain any incoming arc in any node of the tree path $p_{j^*t}(T') = p_{j^*t}(T)$. Therefore, any tree obtained from T' subsequently contains the same sub-path from node *i* to node *t*. In addition, any tree obtained from *T* does not contain the sub-path from node *i* to node *t* in T'

since now $A(T)$ does not contain (i, j^*) . From these comments, it is easy prove that each best solution is obtained one time.

Since, we have fixed the sub-path $p_{it}(T')$, we can delete from $A(T') \cup T'$ all arcs $\{(u,l) \in A : \forall u \in p_{it}(T')\}$ with the exception of the arcs in $p_{it}(T')$ since any alternative path from node *s* to node *t* will not use these arcs. That is, we define

$$E(T') = A - \{(u,l) \in A : \forall l \in p_{j^*t}(T')\} - \{(u,l) \in A : \forall u \in p_{it}(T')\} \cup p_{it}(T').$$

Let $T'_{E(T')}$ be the shortest path tree rooted in *s* for the directed graph (*V*, $E(T')$). Note that $p_{it}(T')$ is contained in $T'_{E(T')}$ and the descendant nodes of node *i* in $T'_{E(T')}$ are just the nodes belonging to $p_{it}(T')$. In other words, the sub-tree hanging from node *i* in $T'_{E(T')}$ is the sub-path $p_{it}(T')$. Then, we interchange T' by $T'_{E(T')}$ and $A(T')$ by $A(T'_{E(T')}) = E(T') \setminus T'_{E(T')}$ in our arguments. Note that now, any non-tree arc in $A(T')$ has non-negative reduced cost and theorem 1 can be applied. Thus, from the later lemmas, theorems and the binary partition scheme, we can conclude without proof, one of the main results in this paper:

Theorem 2. A tree associated to the kth best point-to-point shortest simple path can be obtained from a tree associated to at least one of the previous (*k*-1)*th best point-to-point shortest simple paths*.

4. *An Efficient Algorithm for the K Point-to-Point Shortest Simple Paths Problem.*

This section details the algorithm using the previous results to solve efficiently the *K* point-to-point shortest simple paths problem. Additional notation is first introduced.

Given a basis tree *T*, the proposed method uses the distance label $d_u(T)$, the depth label $depth_u(T)$ and the predecessor label $pred_u(T)$ for each $u \in V$. We assume that in the adjacency node list $T_i^+ = \{ j \in V \mid (i, j) \in T \}$ the values of c_{ij} are stored. We initially set $d_s(T) = 0$, $depth_s(T) = 0$ and $pred_s(T) = s$. These tree indices can then be computed in O(*n*) time by depth-first search for *T* starting in node *s* (see Ahuja et al [1] for example). In order to determine if an arc (l,u) with $u \in p_{st}(T)$ satisfies $l \in D_u(T)$ or not, we define $low_l(T)$ to be the major depth of a node $u \in p_{st}(T)$ such that $l \in D_u(T)$. Therefore, if $low_l(T) \geq low_u(T)$ with $u \in p_{st}(T)$ then $l \in D_u(T)$ and if $low_l(T) < low_u(T)$ with $u \in p_{st}(T)$ then $l \notin D_u(T)$.

Note that labels $low_l(T)$ can be computed in O(n) time by depth-first search for T once the predecessor labels and depth labels are known.

Additionally, we associate a subset of non-tree arcs $A(T)$ with each tree T. For each calculated tree T^p, the arc $(i, j^*) \in NT_{j^*}(T^p)$ is calculated and it is stored together with the index p indicating the associated pth best tree in a heap using as key the length of the path $Ap_{sj^*}(T^p) \cup p_{j^*t}(T^p)$, that is, the way to obtain the second best simple path in $G = (V, A(T^p) \cup T^p)$. We denote this heap by H in the algorithm. Assuming that t is the size of a heap, the operation *Insert* requires an effort O(log t) and the operation of extracting the element of the min-key (*Extract First*) takes O(1) time. The *Create Heap* operation takes O(1) time.

On the other hand, in order to simplify the examination of the set of arcs $A(T)$ for a given tree T, we maintain an additional Boolean label named $eligible_i(T)$ for each node $i \in V$. $eligible_i(T)$ is FALSE if and only if the arc $(pred_i(T), i)$ can not be chosen to leave the tree T (for example, node i belongs to a fixed sub-path). Otherwise, $eligible_i(T)$ is TRUE. Given a basis tree T and its corresponding set of non-tree arcs $A(T)$, we assume that in the adjacency node list $\Gamma_i^+(A(T)) = \{ j \in V \mid (i, j) \in A(T) \}$ the value of c_{ij} and Boolean label $eligible_{ij}(T)$ are stored. $eligible_{ij}(T)$ is TRUE for arc $(i, j) \in A(T)$ if and only if this arc can be chosen to enter into the tree T. Initially the label $eligible_{ij}(T)$ is TRUE for all arc (i, j) in G.

Using the above notation, one way to easily implement the selection of $Ap_{sj^*}(T)$ and, therefore, the arc $(i, j^*) \in NT_{j^*}(T)$, consists in applying the following procedure for a tree T with a fixed sub-path from node u to node t:

Procedure (MAP) Minimum_Alternative_Path(u, var $MinLength$, T, $A(T)$, $d(T)$, $pred(T)$, $low(T)$, var i, var j^*);

(1) $MinLength = \infty$; $MinDesc = \infty$; $eligible_i(T) = \text{TRUE} \quad \forall i \in V$;
(2) $eligible_i(T) = \text{FALSE} \quad \forall i \in p_{u't}(T)$ where u' is the next node to node u in $p_{ut}(T)$;
(3) **While** ($u \neq s$) **do**
(4) **For** all $l \in \Gamma_u^-(A(T))$ **do**
(5) $\overline{c}_{lu} = \infty$;
(6) **If** (($eligible_{lu}(T) == \text{TRUE}$) and ($eligible_l(T) == TRUE$) and ($pred_u(T) \neq l$))
(7) $\overline{c}_{lu} = c_{lu} + d_l(T) - d_u(T)$;
(8) **If** (($\overline{c}_{lu} < MinLength$) and ($low_l(T) < low_u(T)$))
(9) $MinLength = \overline{c}_{lu}$; $i = l$; $j^* = u$;
(10) **If** (($\overline{c}_{lu} < MinDesc$) and ($low_l(T) \geq low_u(T)$)) $MinDesc = \overline{c}_{lu}$;
(11) **If** ($MinDesc < MinLength$)
(12) Determine $l_{su}^*(T)$ using the eligible nodes and arcs;
(13) **If** (length of $l_{su}^*(T) < MinLength$))
(14) $Minlength =$ length of $l_{su}^*(T)$;
(15) Let i be the predecessor node of node u in $l_{su}^*(T)$; $j^* = u$;
(16) $eligible_u(T) = \text{FALSE}$;
(17) $u = pred_u(T)$;
(18) $MinDesc = MinLength$;

The above procedure can not be applied unless the tree T satisfies that the descendant nodes of node u' are just the nodes in the tree path $p_{u't}(T)$, where node u' is the next node to node u in the tree path $p_{ut}(T)$. Therefore, the procedure MAP is called with parameters u being the last node in the tree path from s to t that is fixed in T. The variable $MinLength$ stores the length of the best alternative path from s to t with fixed sub-path from u to t. This procedure returns a pointer to the eligible arc (i, j^*) belonging to $A(T)$ if it exists, otherwise it returns NULL. Given a node $u \in p_{st}(T)$ not fixed (eligible), lines (4)-(10) determine the eligible arc (l,u) satisfying $l \notin D_u(T)$ with minimum reduced cost and these lines allow in $MinDesc$ to store the minimum reduced cost value of eligible arcs (l,u) such that $l \in D_u(T)$. The procedure (line 12) determines $l_{su}^*(T)$ using eligible nodes and eligible arcs only when $MinDesc$ is less than $MinLength$ (the minimum increase in the distance label of node t

currently known). If the length of $l^*_{su}(T)$ is less than *MinLength* then, the procedure updates (i, j^*) and *MinLength*. Next, the procedure backtracks on *u* using the predecessor labels and updates *MinDesc*, and $eligible_u(T) = \text{FALSE}$ for the current node *u*.

Note that, if $u' = pred_u(T)$ then, the distance labels obtained solving $l^*_{su}(T)$ can be used in the determination of $l^*_{su'}(T)$, that is, when $eligible_u(T) = \text{FALSE}$, we re-optimize the shortest path from *s* to u'. For example, let $T_{\bar{B}_u}$ be the shortest path tree obtained considering the arcs in $\bar{B}_u(T)$. Now, the distance label of any node belonging to the sub- tree $T_{\bar{B}_u} \cap \bar{B}_{u'}(T)$ is still optimal. Let *p* be a node not connected in $T_{\bar{B}_u} \cap \bar{B}_{u'}(T)$, then using lemma 2, its initial label distance d'_p becomes $d'_p = \min\left\{\bar{c}_{qp}(T_{\bar{B}_u}) + d_p(T_{\bar{B}_u}) : q \text{ in the sub-tree } T_{\bar{B}_u} \cap \bar{B}_{u'}(T)\right\}$. Moreover the execution of the label setting shortest path algorithm determining $l^*_{su}(T)$ can be stopped when node *u* becomes permanently labeled or the distance label of the new permanently labeled node becomes greater than or equal to *MinLength*.

Next, we report on the computational effort of the procedure *MAP*. Lines (1-2) require $O(n)$ time. Lines (4)-(10) for all *u* in the tree path $p_{su}(T)$ are done in $O(m)$. In the worse case, line (12) is executed $|p_{su}(T)|$ times, that is, $O(nf(n,m,C_{\max}))$ where $O(f(n,m,C_{\max}))$ is the best time needed to solve the point-to-point shortest simple paths in a directed network with non-negative length arcs. The remaining lines require $O(1)$ time. Therefore, the procedure *MAP* employs $O(nf(n,m,C_{\max}))$ time and uses $O(n)$ space. However, note that procedure *MAP* could execute line (12) less than $|p_{su}(T)|$ times. Moreover, it is possible that for a given pair *T* and $A(T)$, line (12) will not be executed. In this "optimistic" case the procedure *MAP* requires $O(m)$ time. For example, when the network is acyclic an arc (l,u) satisfying $l \in D_u(T)$ does not exist and *MAP* employs $O(m)$ time.

We use additional data structures as in Gabow [9] to reduce the memory space needed by the algorithm. Thus, let us assume that the first *k* trees $T^{k'}$, $k' \in \{1,...,k\}$, have been calculated. Then, we use the following structures to store these trees as a *directed out tree*: $father[k']$ stores the index *p* associated with basis tree T^p and a pointer to the arc $(i, j^*) \in NT_{j^*}(T)$ in *G* that determines the way to obtain the tree $T^{k'}$ ($father[k'] = \{p, (i, j^*)\}$). Each element of the list $sons[k']$ contains the index and a pointer to the arc $(i, j^*) \in NT_{j^*}(T)$

in G that allowed the tree T^p to be obtained from $T^{k'}$. The list $sons[k']$ is arranged in such a way that the indices increase from left to right ($sons[k']=\{\{p_1, (i,j^*)_1\},\ldots,\{p_r, (i,j^*)_r\}\}$ and $p_1 < \ldots < p_r$).

Using this information, any tree T^k can be derived from the initial optimal basis tree T^* and $A(T^*)$ by applying the next recursive procedure.

Procedure (BT) BuildingT (k, $father$, $sons$, var $d(T)$, var $pred(T)$, var T, var $A(T)$);

---

(1) **If** ($k \neq 1$)

(2) BT ($father[k].p$, $father$, $sons$, $d(T)$, $pred(T)$, T, $A(T)$);

(3) $(i,j^*) = father[k].(i,j^*)$; $pred_{j^*}(T) = i$;

(4) $E(T) = A(T) - \left\{(u,l) \in A(T) : \forall l \in p_{j^*t}(T)\right\} - \left\{(u,l) \in A(T) : \forall u \in p_{it}(T)\ \right\} \cup p_{it}(T)$;

(5) Let $T = T_{E(T)}$ be the shortest path tree rooted at s for the directed network $(V, E(T))$; Let $d(T)$ and $pred(T)$ be the distance and predecessor labels associated with T;

The procedure BT is called with $T = T^*$, $d(T) = d(T^*)$ and $pred(T) = pred(T^*)$, where T^* is an optimal tree and the index $k = p$ of the tree to be constructed. The procedure BT backtracks on index $k = p$ using *father* until $k = 1$. Therefore, note that path tree from $k = 1$ to $k = p$ in the tree of trees at most have n-1 nodes since in each tree an additional node i is fixed in the sub-path from i to t. In an iteration k, at the beginning of the execution of line (3), the procedure BT has built the tree $T^{father[k].p}$. Line (3) modifies the *pred* label of node j^* using the arc (i,j^*) to identify the fixed sub-path from node i to node t ($p_{it}(T^k)$). Since the distance labels of some descendant nodes of node i (not included in $p_{it}(T^k)$) can not be optimal for the current sub-problem, lines (4) and (5) built T^k as was indicated in section 3. Note that a re-optimization process can be attained from T in line (3) to obtain T^k in line (5) as was mentioned in section 3, for example using lemma 2. The running time of each iteration of procedure BT (lines (3)-(5)) is $O(f(n,m,C_{\max}))$. Therefore, the complexity of the procedure BT is $O(nf(n,m,C_{\max}))$ since at most n-1 recursive calls are performed.

The complete algorithm with the above notation and remarks is presented on next page. The algorithm starts with an optimal tree $T = T^*$. The index of the number of best solutions k is set to 1. All necessary labels associated with the basis tree T are the computed. The arc

(i, j) is determined by calling the procedure *MAP*(t, *MinLength*, T, A, $d(T)$, $pred(T), low(T)$). The heap H is created and the element $\{(i, j)$, 1, $MinLength + d_t(T)\}$ is inserted in H wherever arc (i, j) exists. The algorithm then starts with a loop until the K best solutions are identified or no more feasible solutions are possible. Thus, in each iteration, the first element in the heap is extracted. This element identifies the way to obtain the $(k+1)th$ best solution. In the algorithm $father[k+1] = \{p, (i, j)\}$ lets us determine T^{k+1}. Then, the algorithm adds $\{k+1, (i, j)\}$ at the end of $sons[p]$ to reconstruct in a future, the tree T for all descendant of T^{k+1} in the tree of trees. Now, in the algorithm T^p is reconstructed by the procedure BT and the tree indices of T^p are calculated. Then, the algorithm identifies the tail (leaving) node $u = i$ of the arc $father[p].(i, j)$ being the fixed node in the $p_{st}(T^p)$ with minor depth for $p > 1$. Otherwise, $u = t$ since in T^* the fixed node is only t. Next, all used arcs in the determination of trees that are sons of T^p are marked ineligibles. Then, the new (i, j) arc in $A(T^p)$ is found by adequately calling the procedure *MAP*. The resulting arc (if it exists) and the index p are stored in the heap H using the key value $MinLength + d_t(T)$ where *MinLength* is the value calculated in the procedure MAP. The algorithm then increases the index k and updates T^k. Note that in this case T^k is determined from T^p and $father[k].(i, j)$ as in BT procedure (Lines (19)-(21)). The necessary tree indices of T^k are calculated. Note that T^k has no sons and therefore any arc in A is eligible (line 17). Finally, for this new best tree, the arc (i, j) arc in $A(T^k)$ is determined and the element $\{(i, j)$, k, $MinLength + d_t(T^k)\}$ is inserted in H.

K Point-to-Point Shortest Simple Paths (KPPSSP) **Algorithm**;

(1) Let T^* be an optimal basis tree;

(2) Set $k = 1$; $T = T^*$;

(3) Compute labels $d(T)$, $pred(T)$, $low(T)$ for a tree T;

(4) Create Heap H;

(5) MAP(t, *MinLength*, T, A, $d(T)$, $pred(T)$, $low(T)$, i, j);

(6) **If** ($(i,j) \neq$ NULL) Insert $\{(i,j),\ k,\ MinLength + d_t(T)\}$ in H;

(7) **While** ($(k < K)$ **and** $(H \neq \varnothing)$) **do**

(8) Extract first $\{(i,j),\ C,\ p\}$ of *H;*

(9) $father[k+1] = \{p,(i,j)\}$; Add $\{k+1,(i,j)\}$ at the end of $sons[p]$;

(10) $T = T^*$; $pred(T) = pred(T^*)$;

(11) BT($p, father, sons, d(T), pred(T), T, A$);

(12) Compute labels $low(T)$ for a tree T;

(13) $u =$ leaving node of arc $father[p].(i,j)$ when $p > 1$; otherwise $u = t$;

(14) **For** all $\{l,(i,j)\} \in sons[p]$ **do** $eligible_{ij}(T) =$ FALSE;

(15) MAP(u, *MinLength*, T, A, $d(T)$, $pred(T)$, $low(T)$, i, j);

(16) **If** ($(i,j) \neq$ NULL) Insert $\{(i,j),\ p,\ MinLength + d_t(T)\}$ in H;

(17) **For** all $\{l,(i,j)\} \in sons[p]$ **do** $eligible_{ij}(T) =$ TRUE;

(18) Set $k = k + 1$;

(19) $T = T \cup \{father[k].(i,j)\} - \{(pred_j(T), j)\}$; $pred_j(T) = i$;

(20) $E(T) = A - \{(u,l) \in A : \forall l \in p_{jt}(T)\} - \{(u,l) \in A : \forall u \in p_{it}(T)\} \cup p_{it}(T)$;

(21) Let $T = T_{E(T)}$ be the shortest path tree rooted at s for the directed network $(V, E(T))$; Let $d(T)$ and $pred(T)$ be the distance and predecessor labels associated with T; Compute labels $low(T)$ for a tree T;

(22) $u =$ leaving node of arc $father[p].(i,j)$;

(23) MAP(u, *MinLength*, T, A, $d(T)$, $pred(T)$, $low(T)$, i, j);

(24) **If** ($(i,j) \neq$ NULL) Insert $\{(i,j),\ k,\ MinLength + d_t(T)\}$ in H;

Theorem 3. The KPPSSP *algorithm computes the K point-to-point shortest simple paths in* $O(Knf(n,m,C_{\max}))$ *time and* $O(K+m)$ *space in a directed graph G.*

Proof. In the beginning of the algorithm, the determination of $T^1 = T^*$ requires $O(f(n,m,C_{\max}))$ time (see Ahuja et al. [1] to find a detailed bounds for this function). The initialization and calculation of the labels of the tree T involves $O(n)$ time. The calculation of

the arc (i, j) by the procedure *MAP* requires an effort $O(nf(n,m,C_{\max}))$ time and the operation of create and insert for first time in the heap takes $O(1)$ time. Clearly, the algorithm makes at most *K* iterations. In each iteration of the algorithm, the procedure *MAP* is called twice and procedure BT is called once requiring $O(nf(n,m,C_{\max}))$ time overall, and two insert heap operations are made in O(log *k* +log (*k*+1)). The labels of a tree *T* are calculated twice in each iteration of the loop in $O(n)$ time. The complexity of lines (20)-(21)) is $O(f(n,m,C_{\max}))$. Remainder operations in the loop are made in $O(1)$ time. Thus, the worst-case time complexity of the algorithm is $O(f(n,m,C_{\max}))$ $+\sum_{k=1}^{K-1}(\log k + \log(k+1) + nf(n,m,C_{\max}))$. Since $K < 2^m$ and $O(f(n,m,C_{\max})) \geq O(m)$, then $O(Knf(n,m,C_{\max}))$ time. On the other hand, the space required by the algorithm is $O(K+m)$, since the *father*, *sons* and *heap* structures require $O(K)$ space; the tree *T* and its corresponding labels need $O(n)$ space and the storing *A* uses $O(m)$ space. □

Suppose now that tree T^k (*or the fixed sub-path* $p_{it}(T^k)$) is stored for each *k* best solution. In this case, the algorithm uses $O(Kn+m)$ memory space and does not need the procedure BT (*lines* (20-21) *substitute line* (11) *when p* > 1). In this case the running time of the algorithm is still $O(Knf(n,m,C_{\max}))$, but in the "optimistic" case when procedure *MAP* takes $O(m)$ time, only one (*two*) additional shortest path computation (line 21) is (*are*) made. In this last case, we say that the algorithm runs optimistically in $O(Kf(n,m,C_{\max}))$ time. In the next section, we introduce a computational experiment that shows that the number of shortest path computations made by the proposed algorithms are $O(K)$.

5. Computational results.

We have implemented two version of our algorithm: original KPPSSP algorithm denoted by KPPSSP1 and a version storing all best trees using $O(Kn+m)$ memory space denoted by KPPSSP2. In the current implementation of both algorithms, *H* is a vector and the operation to determine its minimum is sequential, that is, takes $O(K)$ instead of $O(\log K)$. Since in our experiment $O(K) = O(n)$, this fact does not have a significant effect in the empirical performance of the algorithms. Additionally, our implementation of Dijkstra algorithm [8] is simple, that is, we use a naïve scheme of label setting algorithm running in $O(n^2)$ time. In

this case, this fact has a significant effect in the empirical performance of the algorithms. However, note that our interest in the current experiment is to observe if the "optimistic" behavior of these algorithms occurs in practice. These codes were written in C and compiled with the Linux gcc compiler using the O4 optimization option. We used the SPRAND, SPGRID and SPACYC generators attributed to Cherkassky et al. [5]. C codes of these generators are contained in the SPLIB-1.4 library available in the personal web page of A. V. Goldberg (www.avglab.com/andrew/).

The enumerated codes were tested on an Intel® Pentium® M with 2 GHz processor 760 with 1Gb RAM running Red Hat Linux. As in the reference studies, we report the user CPU times in seconds, averaged over several instances generated with the same parameters taking into account the following ten seeds: 12345678, 36581249, 23456183, 46545174, 35826749, 43657679, 378484689, 23434767, 56563897, and 78656756. In each cell of a table appear: the average running time in seconds (in bold) and the average number of shortest path computations per number of computed paths. In particular the number of shortest path computations for the KPPSSP2 equals the number of times that line 12 (re-optimization) of procedure *MAP* is executed plus line 21 of the KPPSSP algorithm (one additional re-optimization for each new candidate path calculated). The number of shortest path computations for the KPPSSP1 equals the number for KPPSSP2 plus the number of times the line 5 of procedure BT is executed.

Table 1: Rand family data.

			KPPSSP1					KPPSSP2				
n	*m*	*K*	200	400	600	800	1000	200	400	600	800	1000
2000	20000		**15,0**	**33,5**	**53,4**	**74,0**	**95,5**	**5,0**	**9,9**	**14,9**	**19,8**	**24,8**
			3,1	3,5	3,7	3,9	4,0	1,1	1,1	1,1	1,1	1,1
2000	40000		**16,9**	**37,4**	**59,4**	**82,3**	**106,0**	**5,4**	**10,8**	**16,2**	**21,6**	**27,0**
			3,2	3,5	3,7	3,9	4,0	1,1	1,1	1,1	1,1	1,1
2000	60000		**17,5**	**39,4**	**63,0**	**87,3**	**112,8**	**5,9**	**11,9**	**17,8**	**23,8**	**29,7**
			3,0	3,4	3,6	3,7	3,8	1,1	1,1	1,1	1,1	1,1
2000	80000		**19,7**	**43,8**	**69,2**	**95,9**	**123,3**	**6,5**	**13,0**	**19,5**	**26,0**	**32,5**
			3,1	3,4	3,6	3,7	3,8	1,1	1,1	1,1	1,1	1,1
2000	100000		**22,5**	**49,9**	**79,3**	**109,9**	**141,7**	**7,0**	**14,0**	**21,0**	**28,0**	**35,0**
			3,2	3,5	3,8	3,9	4,0	1,0	1,1	1,1	1,1	1,1
4000	40000		**57,7**	**128,4**	**205,6**	**287,1**	**371,3**	**20,1**	**40,2**	**60,4**	**80,5**	**100,7**
			2,9	3,2	3,4	3,6	3,7	1,0	1,1	1,1	1,1	1,1
4000	80000		**64,6**	**144,6**	**230,1**	**320,1**	**410,4**	**21,3**	**42,6**	**63,9**	**85,1**	**106,3**
			3,0	3,4	3,6	3,8	3,9	1,1	1,1	1,1	1,1	1,1
4000	120000		**71,0**	**158,8**	**253,0**	**349,4**	**448,0**	**22,6**	**45,1**	**67,8**	**90,5**	**113,0**
			3,2	3,6	3,8	3,9	4,0	1,1	1,1	1,1	1,1	1,1
4000	160000		**72,4**	**160,7**	**257,1**	**356,2**	**457,8**	**23,3**	**46,7**	**70,0**	**93,6**	**116,9**
			3,1	3,5	3,7	3,8	3,9	1,0	1,1	1,1	1,1	1,1
4000	200000		**75,7**	**167,4**	**267,2**	**369,1**	**474,8**	**24,3**	**48,6**	**72,9**	**97,2**	**121,5**
			3,1	3,4	3,6	3,8	3,9	1,0	1,0	1,0	1,0	1,0
6000	60000		**129,1**	**287,1**	**456,6**	**636,0**	**818,4**	**44,8**	**89,5**	**134,3**	**179,2**	**224,0**
			2,9	3,2	3,4	3,5	3,6	1,0	1,0	1,0	1,0	1,0
6000	120000		**141,3**	**315,1**	**501,6**	**693,5**	**893,6**	**46,9**	**93,8**	**140,5**	**187,4**	**234,1**
			3,0	3,4	3,6	3,7	3,8	1,1	1,1	1,1	1,1	1,1
6000	180000		**142,0**	**318,0**	**504,6**	**699,3**	**901,5**	**48,2**	**96,4**	**144,7**	**192,7**	**244,2**
			3,0	3,3	3,5	3,7	3,8	1,1	1,1	1,1	1,1	1,1
6000	240000		**147,9**	**333,2**	**529,6**	**733,4**	**944,8**	**49,8**	**99,7**	**149,2**	**199,1**	**248,7**
			3,0	3,3	3,6	3,7	3,8	1,1	1,1	1,1	1,1	1,1
6000	300000		**152,2**	**338,5**	**539,7**	**749,3**	**963,7**	**51,7**	**103,3**	**155,0**	**207,0**	**258,9**
			3,0	3,3	3,6	3,7	3,8	1,1	1,1	1,1	1,1	1,1
8000	80000		**241,4**	**534,1**	**848,8**	**1178,1**	**1517,0**	**78,9**	**158,2**	**237,2**	**316,4**	**395,1**
			3,0	3,4	3,5	3,7	3,8	1,0	1,0	1,0	1,0	1,0
8000	160000		**243,9**	**546,4**	**868,6**	**1207,3**	**1555,1**	**82,6**	**165,8**	**248,6**	**331,5**	**414,0**
			3,0	3,3	3,5	3,7	3,8	1,1	1,1	1,1	1,1	1,1
8000	240000		**257,1**	**569,5**	**907,7**	**1256,0**	**1615,7**	**84,1**	**168,4**	**252,4**	**336,7**	**420,7**
			3,0	3,4	3,6	3,7	3,8	1,0	1,0	1,0	1,0	1,0
8000	320000		**259,7**	**577,5**	**921,2**	**1282,1**	**1649,0**	**86,3**	**172,6**	**258,9**	**345,5**	**431,8**
			3,1	3,4	3,6	3,8	3,9	1,1	1,1	1,1	1,1	1,1
8000	400000		**244,8**	**551,0**	**877,3**	**1224,0**	**1586,4**	**88,4**	**176,8**	**264,8**	**353,3**	**441,7**
			2,8	3,1	3,3	3,5	3,6	1,1	1,1	1,1	1,1	1,1
10000	100000		**369,2**	**834,8**	**1332,0**	**1842,6**	**2370,4**	**122,5**	**245,5**	**367,7**	**490,3**	**612,9**
			3,0	3,4	3,6	3,7	3,8	1,0	1,0	1,0	1,0	1,0
10000	200000		**395,0**	**872,3**	**1389,6**	**1921,7**	**2476,0**	**125,9**	**251,9**	**377,7**	**504,0**	**629,7**
			3,1	3,4	3,7	3,8	3,9	1,0	1,0	1,0	1,0	1,0
10000	300000		**397,8**	**875,6**	**1391,9**	**1925,1**	**2479,0**	**128,5**	**257,1**	**385,5**	**514,1**	**642,5**
			3,1	3,4	3,6	3,7	3,8	1,0	1,0	1,0	1,0	1,0
10000	400000		**393,8**	**879,8**	**1391,8**	**1931,4**	**2493,2**	**131,5**	**263,0**	**403,6**	**529,2**	**656,8**
			3,0	3,3	3,5	3,7	3,8	1,0	1,0	1,0	1,0	1,0
10000	500000		**406,4**	**907,7**	**1448,4**	**2014,0**	**2601,4**	**133,9**	**267,8**	**401,8**	**536,0**	**670,1**
			3,0	3,4	3,6	3,7	3,8	1,0	1,0	1,0	1,0	1,0

We first used the SPRAND generator attributed to Cherkassky et al. [5]. We present results for random graphs with uniform arc lengths at random from interval $[0,10000]$ with $n \in \{2000, 4000, ..., 10000\}$, $m \in \{10n, 20n, ..., 50n\}$ and $K \in \{200, 400, ..., 1000\}$. The origin and destination nodes were 1 and *n*, respectively. In total 10x5x5x10=2500 instances were solved by each algorithm. Note that the size of the networks in this experiment is reasonably high, for example, the greater size corresponds to a network with 10000 nodes and 500000 arcs. The results are shown in Table 1. We note that for all instances, the number of enumerated paths equals *K*, that is, the number of origin-destination paths in each instance is greater or equal to *K*. We observe that the CPU time employed by KPPSSP1 algorithm is at most four times the CPU time of the KPPSSP2 algorithm. This fact is directly in relation with the average of the number of shortest path computations per number of computed paths (#sp). Note that this ratio is at most four for the KPPSSP1 algorithm and at most 1.1 for the KPPSSP2 algorithm. In other words, the number of shortest path computations carried out for KPPSSP2 algorithm is practically one per computed path. Clearly, the practical behavior of this algorithm can be qualified as "ideal" for this kind of instances. Moreover, at most four shortest path computations per computed path are made in the KPPSSP1 algorithm. This number is also low. An explanation is that the average depth of the each computed path in the tree of trees is close to three. Therefore, the number of times calling to procedure BT is close to 3*K*. Moreover, the value of #sp in the KPPSSP2 algorithm remains constant for all values of *K*, while the value of #sp in the KPPSSP1 algorithm slowly increases as *K* increases. In other words, the CPU time and the number of shortest path computations in both algorithms are linear in *K*. Clearly, the practical behaviors of both algorithms coincide with the "optimistic" case for the SPRAND generator problems.

We also used the SPGRID generator attributed to Cherkassky et al. [5]. We present results for Grid-SSquare (square grids) family data (Table 2). We present results for random graphs with uniform arc lengths at random from interval $[0,10000]$ with $X \in \{16, 32, 64, 128\}$ and $Y = X$. The origin and destination nodes were 1 and *n*-1, respectively. In total 4x5x10=200 instances were solved using each algorithm. Also, we note that for all instances, the number of enumerated paths equals *K*, that is, the number of origin-destination paths in each instance is greater or equal to *K*. We observe that the CPU time employed by KPPSSP1 algorithm is at most 3.5 times the CPU time of the KPPSSP2 algorithm. In this case, the value of #sp in the KPPSSP1 algorithm is at most 1.9 times the the value of #sp in the KPPSSP2 algorithm.

Again, the behavior of both algorithms is linear in relation with *K*. Moreover, the practical behaviors of both algorithms is close to the "optimistic" case for the SPGRID generator problems. For example, at most 7 (≥ 6.5) shortest path computations per computed path are made in the KPPSSP1 algorithm. This number is 4 (≥ 3.7) for the KPPSSP2 algorithm. We observe that the average depth of the tree of trees for the SPGRID instances is at most 3 since the difference of the ratio #sp for the two algorithms is near to 3.

From tables 1 and 2, we also conclude that the instances provided by SPRAND were easily solved for all the codes that the instances obtained by SPGRID as already was observed in others shortest path algorithms experiments.

Table 2: Grid-SSquare family data.

	KPPSSP1					KPPSSP2				
X=*Y* / *K*	200	400	600	800	1000	200	400	600	800	1000
16	**0,2**	**0,5**	**0,8**	**1,0**	**1,3**	**0,1**	**0,2**	**0,2**	**0,3**	**0,4**
	5,7	6,1	6,3	6,4	6,5	3,8	3,8	3,7	3,7	3,7
32	**2,3**	**5,2**	**8,3**	**11,5**	**14,7**	**0,9**	**1,8**	**2,7**	**3,6**	**4,5**
	5,1	5,5	5,7	5,8	5,9	3,2	3,3	3,3	3,3	3,2
64	**33,2**	**74,4**	**118,2**	**164,3**	**212,3**	**12,2**	**24,5**	**36,9**	**49,5**	**62,0**
	4,7	5,1	5,3	5,4	5,6	2,9	2,9	2,9	3,0	3,0
128	**479,2**	**1055,8**	**1676,8**	**2343,6**	**3003,2**	**184,2**	**368,0**	**552,6**	**738,2**	**924,0**
	4,9	5,3	5,6	5,7	5,9	3,2	3,4	3,4	3,5	3,5

We also used the SPACYC generator attributed to Cherkassky et al. [5]. We do not report a table of the CPU time and the ratio #sp for short. It is clear that the KPPSSP2 algorithm only makes 1 shortest path computation per enumerated path in acyclic networks.

6. Conclusions.

From this paper, we conclude that the *K* point-to-point shortest simple path problem can be solved using only $O(K+m)$ space instead of $O(Kn^2)$ space in the same $O(Knf(n,m,C_{\max}))$ time . Therefore, the reduction of the requirements reached by our algorithm is notability reduced. Moreover, if in our algorithm, we store not only the tree of trees but also the *K* shortest path trees then, we obtain a version of the algorithm using $O(Kn)$ space and not using the procedure BT. The worst-case time complexity of this algorithm is still $O(Knf(n,m,C_{\max}))$, but in the "optimistic" case commented on section 3, the complexity of the algorithm could be $O(Kf(n,m,C_{\max}))$ time since procedure MAP takes $O(m)$ time and the algorithm needs to compute one shortest path tree in each iteration. In this paper, we exploit the structural relations between two path trees, that is, the *k*th best solution can be obtained from at least one of the $k-1$ best previous solutions using the respective trees. Thus, we

design an algorithm that in the "optimistic" case takes advantage of the exchange operation for a tree of the PPSSP problem. On the other hand, the results addressed in this paper can be useful to develop new algorithms for the multiobjective shortest path problem from one source node to one sink node in a network (see Azvedo et al. [2] , Climaco and Martins [6] and Raith and Ehrgott [19]).

Acknowledgments

This work has been partially supported by Spanish Government Research Project MTM2006-10170.

References

1. Ahuja, R., T. Magnanti, J. B. Orlin. 1993. *Network Flows*. Prentice-Hall, inc.
2. Azvedo, J., E.Q.V. Martins. 1991. An algorithm for the multiobjective shortest path problem on Acyclic networks. *Investigacao Operacional* **11** 52-69.
3. Brander, A., M. Sinclair (1995). A comparative study of *K*-shortest path algorithms. *In Proc. Of 11th UK Performance Engineering Workshop* 370-379.
4. Carlyle, R., R.K. Wood. 2005. Near-shortest and K-shortest simple paths. *Networks* **46** (2) 98-109.
5. Cherkassky, B. V., A. V. Goldberg, T. Radzik. 1996. Shortest paths algorithms: Theory and experimental evaluation, Math. Program. **73** 129 - 174.
6. Climaco, J.C.N., E.Q.V. Martins. 1982. A bicriterion shortest path algorithm. *European Journal of Operational Research* **11** 399-404.
7. Eppstein, D. 1999. Finding the *K* shortest paths. *Siam Journal on Computing* **28** 653-674.
8. Dijkstra E. W. 1959. A note on two problems in connection with graphs. *Numer. Math.* **1** 269-271.
9. Gabow, H. N. 1977. Two Algorithms for Generating Weighted Spanning Trees in Order. *Siam Journal on Computing* **6** (1) 139-150.
10. Hadjiconstantinou, E., N. Chirstofides. (1999). An efficient implementation of an algorithm for finding *K* shortest simple paths. *Networks* **34** 88-101.
11. Hershberger, J., M. Maxel, S. Suri. (2007). Finding the *k* Shortest Simple Paths: a new algorithm and its implementation. *ACM transactions on Algorithms* **3** 4.
12. Hoffman, W., R. Pavley. 1959. A method for the solution of the *N*th best path problem. *Journal of the ACM* **6** 506-514.
13. Katoh, N., T. Ibaraki, H. Mine. (1982). An efficient Algorithm for *K* Shortest Simple Paths. *Networks* **12** 411-427.
14. Lawler, E. L. (1972). A procedure for computing the *K* best solutions to discrete optimization problems and its application to the shortest path problem. *Magnagement Science* **18** 401-405.
15. Martins, E.Q.V., M. M. B. Pascoal. 2003. A new implementation of Yen's ranking loopless paths algorithm. *4OR – Quarterly Journal of the Belgian, French and Italian Operations Research Socities*, **1** (2) 121-134.

16. Martins, E.Q.V., M. M. B. Pascoal, J. Santos. 1997. A new algorithm for ranking loopless paths algorithm. *Technical report, Universidade de Coimbra, Portugal*.
17. Perko, A. (1986). Implementation of algorithms for K shortest loopless paths. *Networks* **16** 149-160.
18. Pollack, M. (1961). The *k*th best route through a network. *Operations Research* **9** 578-580.
19. Raith A., M. Ehrgott. 2009. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research* **36** (4) 1299-1331.
20. Sedeño-Noda, A., C. González-Martín. 2006. Shortest Path Simplex Algorithm with a Multiple Pivot Rule. *Technical report nº 2, Departamento de Estadística, Investigación Operativa y Computación*.
21. Sedeño-Noda, A. 2008. An Efficient K Point-to-Point Shortest Simple Paths Algorithm in Acyclyc Networks. *Technical report nº 5, Departamento de Estadística, Investigación Operativa y Computación*.
22. Yen, J. Y. 1971. Finding the *K* shortest loopless paths in a network. *Management Science* **17** 712-716.
23. Yen, J. Y., 1972. Another algorithm for finding the *K* shortest loopless network paths. *In Proc. of 41st Mtg. Operations Research Society of America* **20**.