An Efficient *K* Point-to-Point Shortest Simple Paths Algorithm in acyclic networks.

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 and without directed cycle. 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 a single T-exchange from at least one of the previous (*k*-1) trees representing each one of the previous (*k*-1) best point-to-point shortest simple paths. Consequently, we design an $O(m + n\log n + K(n + \log\frac{K}{n}))$ time and O(*K*+*m*) space algorithm to determine explicitly the *K* point-to-point shortest simple paths in a directed network with *n* nodes, *m* arcs and without directed cycles. The algorithm does not require complicated data structures and its implementation becomes easy.

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 an *m* arcs with arbitrary lengths on the arcs finds a shortest length path from a source node to a sink node or in detects a cycle of negative length. Many important real cases where this problem appears and numerous algorithms to solve it are addressed in Ahuja et al. [1] for example.

In this paper, we consider the *K point-to-point shortest simple paths* problem as the problem to determine 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 [3] for example). In particular, we suppose that the network does not contain a directed cycle. In this case, a clasical aplication is the determination of the *K* best critical path in PERT graph. Many of the applications included in Eppstein [3] use a directed acyclic graph (DAG). We chronologically cite the papers of the literature considering the *K* shortest simple paths problem: Hoffman and Pavley [7], Pollack [13], Yen [15] and [16], Lawler [9], Katoh et al. [8], Perko [12], Brander and Sinclair [2], Martins et al. [11], Hadjiconstantinou and

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

Chirstofides[5], Martins and Pascoal [10] and Hersberger et al. [6]. The best bound to solve this problem in directed networks is reached in the early paper of Yen [15]. Yen's [15] algorithm in a directed acyclic network runs in $O(Knm)$ time where the term $O(m)$ is the best bound to solve the PPSSP problem. This algorithm requires $O(Kn^2 + m)$ memory space. Moreover, the algorithm of Hershberger et al. [6] correctly computes the K best solutions in $O(Km)$ time and $O(Kn+m)$ space on acyclic networks. The K point-to-point shortest paths problem in which paths are not required be simple are easier. The best algorithm for this problem is due to Eppstein [3]. The algorithm of Eppstein [3] compute a 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 where the first term refers to a bound on the arcs in the path and the second refers to obtain the K best path. That is, the K shortest path can be enumerated in order by Eppstein [3] algorithm in $O(m + n\log n + K(n + \log K))$. Clearly, the algorithm of Eppstein [3] can be used to determine the K point-to-point shortest simple paths in a network without directed cycle since any path in acyclic network is a simple path. An extended bibliography of several K best shortest path problems due to Eppstein is available at <`http://www.ics.edu/~eppstein/bibs/kpath.bib`>.

In this paper, we prove that the kth best solution is obtained from one of the previous best solutions by a T-exchange between a tree arc and a non- tree arc of the associated best solution in acyclic networks. This results let us design an algorithm running in $O(m + n\log n + K(n + \log \frac{K}{n}))$ time and requiring $O(K+m)$ memory space. The algorithm determines in order each best simple path. For that the set of non-tree arcs are sorted by its reduced cost and K vectors with $O(K)$ space overall are used to determine in $O(n + \log \frac{K}{n})$ time the next best solution. It is clear that the introduced algorithm significantly improves the early time and space bounds of Yen [15] and the bounds of Hershberger et al. [6] for directed networks. Clearly the running time of our algorithm also improves the Eppstein [3] algorithm wherever $K > n$, otherwise, both algorithms have the same running time. Moreover, our algorithm does not use complicated data structures and, therefore, its implementation in a high-level language programming becomes easy.

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 theoretical main results, which the algorithm is based on. Section 4 contains a

detailed pseudo code and an explanation of the proposed algorithm. Additionally, the worse case time and space theoretical complexity of the algorithm is proven.

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 suppose without lost of generality that the directed network G does not contain any arc emanating from node sink t. Note that in any case, a simple path from s to t does not use arcs emanating from t. Thus, we can simply ignore them.

Assumption 1. The network contains a directed path from source node s to any non-source node $i \in V$. When this condition is not true for some node i, this node is eliminated from G since any s-t path in G does not contain node i.

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 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 length simple path from node s to node t or in determining a negative cycle, that is, a directed cycle of negative length. Now, we suppose that the network does not contain a directed cycle. In this case we have an acyclic network. Moreover, the directed network G does not contain any arc arriving at node source s. Otherwise, G contains a directed cycle.

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]):

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

subjet to

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

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

The above problem is an especial case of the minimum cost network flow (MCNF) problem. The network simplex algorithm can be used to solve the above problem 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*). Next two literature results holds (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. 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*.

The distance labels of the nodes 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*. Moreover, $c(x) = d_t(T) = C(T)$. Therefore, minimize $c(x)$ is equal to find 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 $x_j(T) = 0$.

In a T-*exchange*, an arc $(i,j) \in A \setminus T$ with reduced cost $\overline{c}_{ij}(T)$ and $i \notin D_j(T)$ is added to *T* and $(pred_j(T), j)$ is deleted from *T* yielding a new tree T'. Moreover, any arc $(i,j) \in A \setminus T$ satisfies $i \notin D_j(T)$ since network is acyclic. Once a T-exchange is performed, the distance

labels in T' are updated in the following way: $d_k(T') = d_k(T) + \overline{c}_{ij}(T)$, $\forall k \in D_j(T)$. Furthermore, the objective function value is $d_t(T') = d_t(T) + \overline{c}_{ij}(T)x_j(T)$ since $x_j(T) = x_j(T')$.

For an optimal tree T^*, we obtain the following optimality conditions: $\overline{c}_{ij}(T^*) \geq 0,\ \forall (i, j) \in A$.

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 requires to identify the *K* best tree 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. For that, we recall the following definition:

Definition 1. Two tree *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 satisfies that $x_j(T) = 1$, that is, node *j* must belong to the path tree from node *s* to node *t* in *T*. Therefore all alternative path from *s* to *t* obtained from *T* must use a non tree-arc $(i, j) \in A \setminus T$ with $x_j(T) = 1$ and $i \notin D_j(T)$. But, since G does not contain directed cycle, the suffcient condition is a non tree-arc $(i, j) \in A \setminus T$ with $x_j(T) = 1$. 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 [14] 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*

$u, v \in \{1,..., p\}$ *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*. Now, we can prove that:

Lemma 1. *Given a tree T, 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* $\{(i_1, j_1),(i_2, j_2),...,(i_p, j_p)\}$ *with non-negative reduced cost and with* $2 \le p < n$*. 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* $\{(i_1, j_1),(i_2, j_2),...,(i_p, j_p)\}$.

Proof. Note that since T' contains a different path tree from node *s* to node *t*, at least an arc (i_r, j_r) in the set $\{(i_1, j_1),(i_2, j_2),...,(i_p, j_p)\}$ satisfying $x_{j_r}(T) = 1$ must exist. Thus, le us first consider that the arcs in the set $\{(i_1, j_1),(i_2, j_2),...,(i_p, j_p)\}$ satisfy that $x_{j_1}(T) = 1$ and $x_{j_u} = 0$ for all $u \in \{2,..., p\}$. Then, if T' is the obtained tree by a multiple T-exchange, it is easy to prove that $d_t(T') \ge d_t(T) + \bar{c}_{i_1 j_1}(T) x_{j_1}(T)$. Note that in T', $d_{j_1}(T') = d_{i_1}(T') + c_{i_1 j_1}$. Moreover, $d_{i_1}(T') \ge d_{i_1}(T)$ since all entering arc has non-negative reduced cost. Thus, $d_{j_1}(T') \ge d_{i_1}(T) + c_{i_1 j_1}$. Additionally, $d_u(T') = d_u(T)$ for all node *u* in the path tree from node *s* to node $pred_{j_1}(T)$ in *T* (or T'). Therefore, the increase in the distance label of any node hanging of node j_1 in T' equals the increase in the distance label of node j_1, that is, $d_{j_1}(T') - d_{j_1}(T) \ge d_{i_1}(T) + c_{i_1 j_1} - d_{j_1}(T) \ge \bar{c}_{i_1 j_1}(T)$. Thus, $d_t(T') - d_t(T) \ge \bar{c}_{i_1 j_1}(T)$. But, if T'' is the tree obtained from *T* by making a single T-exchange with the entering arc (i_1, j_1) then, we have that $d_t(T'') = d_t(T) + \bar{c}_{i_r j_r}(T)$. In other words, the tree T'' has a distance label for node *t* less than or equal to the distance label of node *t* in any tree obtained making a multiple T-exchange with any subset of arcs in $\{(i_1, j_1),(i_2, j_2),...,(i_p, j_p)\}$ containing the arc (i_1, j_1). Therefore, now we consider that all arc in the set $\{(i_1, j_1),(i_2, j_2),...,(i_p, j_p)\}$ satisfies $x_{j_u}(T) = 1$ for all $u \in \{1,..., p\}$. Let j_r be the node with major depth in the tree among the

nodes j_u with $u \in \{1,..., p\}$. In this case, it is clear that the tree T' obtained from T by making a multiple T-exchange with arcs $\{(i_1, j_1),(i_2, j_2),...,(i_p, j_p)\}$ satisfies $d_t(T') \geq d_t(T) + \bar{c}_{i_r j_r}(T)$ since node j_r belongs to the path tree $p_{st}(T')$. But, if T'' is the tree obtained from T by making a single T-exchange with the entering arc (i_r, j_r) then, we have that $d_t(T'') = d_t(T) + \bar{c}_{i_r j_r}(T)$ and $d_t(T') \geq d_t(T'')$, therefore, the lemma holds. □

Lemma 1 does not hold for directed network containing directed cycle because any alternating s-t path could use a non tree-arc $(i, j) \in A \setminus T$ with $x_j(T) = 1$ and $i \in D_j(T)$.

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 $(V,\ A(T))$. Then we define $(i, j)_{A(T)} = \arg \min_{(u,l) \in A(T)} \{\bar{c}_{ul}(T) :\ x_l(T) = 1\}$. We only consider in the previous minimum arcs (u,l) such that $x_l(T) = 1$ because a single T-exchange with any of these arcs allow us to obtain a different path tree from node s to node t in the new tree. Let T' be the tree obtained from T by making a T-exchange with the arc $(i, j)_{A(T)} \in A(T)$, then from lemma 1, T' is the second best solution of the PPSSP problem in the acyclic network G=(V, $A(T) \cup T$). We obtain the next result:

Lemma 2. *Given a tree T, with a set of arcs $A(T)$ with non-negative reduced cost. Let T' be the tree obtained from T by making a T-exchange with the arc $(i, j)_{A(T)} \in A(T)$. Then any arc (u,l) in the set $A(T') = A(T) - \{(i, j)_{A(T)}\}$ satisfying $u \in D_j(T)$ or $l \notin D_j(T)$ or $x_l(T) = 1$ has a non-negative reduced cost with respect to the tree T'.*

Proof. Given a tree T, when a T-exchange is made with the entering arc $(i, j) = (i, j)_{A(T)} \in A(T)$ obtaining the tree T', only the distance labels of the nodes in $D_j(T)$ increase by $\bar{c}_{i,j}(T)$. We must consider the next cases to verify the sign of each arc (u,l) in $A(T') = A(T) - \{(i, j)\}$:

Case 1) if $u \in D_j(T)$ and $l \in D_j(T)$ or $u \notin D_j(T)$ and $l \notin D_j(T)$ then, we have $d_u(T') - d_l(T') = d_u(T) - d_l(T)$ and therefore $\bar{c}_{ul}(T') = \bar{c}_{ul}(T) \geq 0$.

Case 2) if $u \in D_j(T)$ and $l \notin D_j(T)$ then

$$\overline{c}_{ul}(T') = c_{ul} + d_u(T') - d_l(T') = c_{ul} + d_u(T) + \overline{c}_{ij}(T) - d_l(T) = \overline{c}_{ul}(T) + \overline{c}_{ij}(T) \geq 0 .$$

Case 3) if $u \notin D_j(T)$ and $l \in D_j(T)$ then

$$\overline{c}_{ul}(T') = c_{ul} + d_u(T') - d_l(T') = c_{ul} + d_u(T) - d_l(T) - \overline{c}_{ij}(T) = \overline{c}_{ul}(T) - \overline{c}_{ij}(T) .$$

We must distinguish two sub-cases: Sub-case 3.1) if $x_l(T) = x_j(T) = 1$ then since $(i, j)_{A(T)} = \arg \min_{(ul) \in A(T)} \{\overline{c}_{ul}(T) : x_l(T) = 1\}$ and $x_l(T) = x_j(T) = 1$ then $\overline{c}_{ul}(T) - \overline{c}_{ij}(T) \geq 0$. Sub-case 3.2) if $x_l(T) = 0$ then only we can conclude that $\overline{c}_{ul}(T') = \overline{c}_{ul}(T) - \overline{c}_{ij}(T)$.

Therefore, any arc (u,l) in the set $A(T')$ with $u \in D_j(T)$ or $l \notin D_j(T)$ or $x_l(T) = 1$ has $\overline{c}_{ul}(T') \geq 0$. □

We are interested in generate 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*, the entering arc $(i, j)_{A(T)} = \arg \min_{(u,l) \in A(T)} \{\overline{c}_{ul}(T) : x_l(T) = 1\}$ and the tree T' obtained by making a T-exchange with the entering arc (i, j), we set: $A(T) = A(T) - \{(i, j)\}$ and $A(T') = \{(u,l) \in A \setminus T' : u \notin D_j(T) \text{ and } l \notin D_j(T)\}$. Note that $A(T')$ does not contain any incoming arc in any descendant node of node *j* belonging to $p_{st}(T')$. 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*. From these comments, it is easy prove that each best solution is obtained once time. The reason of the definition of the set of arcs $A(T')$ is given by the next result:

Lemma 3. *Let T' be a tree obtained from T by a T-exchange with the arc (i, j). If we decided to fix the tree path from i to t then, we must set $A(T') = \{(u,l) \in A \setminus T' : u \notin D_j(T) \text{ and } l \notin D_j(T)\}$.*

Proof. Given a tree *T*, when a T-exchange is made with the entering arc $(i, j) = (i, j)_{A(T)} \in A(T)$ obtaining the tree T', only the distance labels of the nodes in $D_j(T)$ are modified as indicated in proof of lemma 2. Suppose, that we maintains in $A(T')$ some arc

(u,l) with $l \in D_j(T)$. If this arc belongs to an alternative path from *s* to *t* then, a path $p_{ll'}$ from *l* to a node $l' \in p_{st}(T')$ must exist in *G*. Now we consider two cases: (1) if $l' \in D_j(T)$ then the sub-path $(u,l) \cup p_{ll'} \cup p_{l't}(T')$ does not contain the sub-path $p_{it}(T')$. Therefore, the arc (u,l) must not be included in $A(T')$. (2) if $l' \notin D_j(T)$ then, we have a directed cycle $p_{ll'} \cup p_{l'j}(T') \cup p_{jl}(T')$. Clearly, this situation is not possible and, therefore, a path $p_{ll'}$ from *l* to a node $l' \in p_{st}(T')$ and $l' \notin D_j(T)$ can not exist. Therefore, the arc (u,l) is not required to be included in $A(T')$.

Similarly, suppose, that we maintains in $A(T')$ some arc (u,l) with $u \in D_j(T)$. If this arc belongs to an alternative path from *s* to *t* then, a path $p_{ul'}$ including (u,l) from *u* to a node $l' \in p_{st}(T')$ must exist in *G*. Using similar above arguments, we obtain that the arc (u,l) must be not included in $A(T')$. Therefore, $A(T') = \{(u,l) \in A \setminus T' : u \notin D_j(T) \text{ and } l \notin D_j(T)\}$. □

Therefore, any non-tree arc in $A(T')$ satisfies case 1 of proof of lemma 2, that is, this arc maintains its non-negative reduced cost value. Thus, from lemma 2 and lemma 3, searching $(i,j)_{A(T')} = \arg\min_{(u,l)\in A(T')} \{\bar{c}_{ul}(T') : x_l(T') = 1\}$ is equivalent to search $(i,j)_{A(T')} = \arg\min_{(u,l)\in A(T')} \{\bar{c}_{ul}(T^*) : x_l(T') = 1\}$ where T^* is an optimal tree.

Finally, from the later lemmas and the binary partition scheme, we can conclude without proof, one of the main results in this paper:

Theorem 1. *Given a directed acyclic network, the kth point-to-point shortest simple path can be obtained by making a single T-exchange in the tree of at least one of the previous* (*k*-1)*th point-to-point shortest simple paths*.

4. An Efficient Algorithm for the *K* Point-to-Point Shortest Simple Paths Problem in a directed network without directed cycle.

This section details the algorithm to solve efficiently the *K* point-to-point shortest simple paths problem in an acyclic network. For that, we introduce additional notation.

Given a tree *T*, the proposed method uses distance label $d_u(T)$ 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 suppose that the set of predecessor nodes of node *i* in the directed graph (*V, A*), $\Gamma_i^- = \{ j \in V \mid (j,i) \in A \}$, are stored in non-decreasing order of the reduced cost $\overline{c}_{ji}(T^*)$ where T^* is an optimal tree, for any node $i \in V$. The first element in Γ_i^- is $first_i$ and last element is $last_i$. Given an pointer *pt* to an element in Γ_i^-, *next*(*pt*) give the next element to *pt* if *pt* is different of $last_i$, otherwise *next*(*pt*) is NULL.

Additionally, for each calculated tree T^p, a pointer *pt* to the arc $(i, j)_{A(T^p)} = \arg \min_{(u,l) \in A(T^p)} \{ \overline{c}_{ul}(T^*) :\ x_l(T^p) = 1 \}$ is calculated (the arc is $(pt.i, pt.j)$) and it is stored together the index *p* indicating the associated *p*th best tree in a heap using as key the value $\overline{c}_{ij}(T^*) + d_t(T^p)$. Algorithm uses *n* (Fibonacci) heaps, we denote the i*th* heap by H_i in the algorithm with $i \in \{1, ..., n\}$. The algorithm stores the *l*th generated candidate in the heap with index $l \bmod n + 1$. Therefore, the maximum size of any of these heaps is O($\frac{K}{n}$). Assuming that *t* is the size of a heap, the operation *Insert* requires an effort O(log *t*) and the operations *Create Heap, Findmin* and *Deletemin* takes O(1) time.

In order to simplify the examination of the set of arcs $A(T)$ for a given tree *T*, the algorithm maintains a (dynamic) vector containing at most *n* pointers to the last arcs arriving at different nodes in $p_{st}(T)$ which were used to generate other best solutions. For example, if from tree *T* were used the arcs (u,l), (u',l') and (u'',l'), the vector contains only two entries: a pointer to arc (u,l) in Γ_l^- and a pointer to arc (u'',l') in $\Gamma_{l'}^-$. Thus, we can ask in constant time for the next arc arriving at a particular node in $p_{st}(T)$. We denote by $Ve[k]$ the vector associated with k*th* calculated best solution. Operations *Create*, *Add* (at the end) and *Access* to a given position of a vector $Ve[k]$ takes O(1) time.

Therefore, now, a way to easily implement the selection of the arc $(i, j) = (i, j)_{A(T)} = \arg \min_{(u,l) \in A(T)} \{ \overline{c}_{ul}(T^*) :\ x_l(T) = 1 \}$ consist in applying the following procedure:

Procedure (SMA) Searching_Minimum_Arc(var C, var $pt_{\min}$, T, $Ve[k]$, $pred(T)$, u);

$visited_v = FALSE \quad \forall v \in V$; $pos =$ NULL;

For all $pt \in Ve[k]$ **do**

$visited_{pt.j} = TRUE$; $pt = next(pt)$;

//if pt point to an arc belonging to T then, pt is increased

If (($pt \neq NULL$) **and** ($pred_{pt.j}(T) == pt.i$)) **Then** $pt = next(pt)$;

If (($pt \neq NULL$) **and** ($\overline{c}_{pt.ipt.j}(T^*) < C$))**Then**

$C = \overline{c}_{pt.lpt.u}(T^*)$; $pt_{\min} = pt$; $pos =$ current position in $Ve[k]$;

While ($u \neq s$) **do**

If ($visited_u == FALSE$)**Then**

$pt = first_u$; $visited_u = TRUE$

//if pt point to an arc belonging to T then, pt is increased

If (($pt \neq NULL$) **and** ($pred_{pt.j}(T) == pt.i$)) **Then** $pt = next(pt)$;

If (($pt \neq NULL$) **and** ($\overline{c}_{pt.lpt.u}(T^*) < C$))**Then**

$C = \overline{c}_{pt.lpt.u}(T^*)$; $pt_{\min} = pt$; $pos =$ NULL;

$u = pred_u(T)$;

If ($pos \neq NULL$) **Then** Set position pos in $Ve[k]$ equal to $pt_{\min}$;

Else Add $pt_{\min}$ at the end of $Ve[k]$;

The procedure *SMA* is called with the variable $C = \infty$ to determine the pointer to an arc $(i, j)_{A(T)} = \arg \min_{(u,l) \in A(T)} \{\overline{c}_{ul}(T^*) : x_l(T) = 1\}$ for a given tree T with the set of non-tree arcs $A(T)$. The value of u is the fixed node in $p_{st}(T)$ with minor depth in *T*. Initially, all node $v \in V$ is considered as not visited. Next, all positions of the vector $Ve[k]$ are scanned. Note that the last arc arriving at node $pt.j$ already used was stored in $Ve[k]$ and therefore *pt* equals $next(pt)$ and the procedure marks $pt.j$ as visited. If the arc $(pt.i, pt.j) \in p_{st}(T)$, the procedure makes $next(pt)$. In any case, procedure keeps *pt* being the best candidate arc arriving at node $pt.j$. The procedure store in *pos* the position in $Ve[k]$ indicating the current best candidate. Next, the procedure backtracks on the index u using the *pred*(*T*) labels until $u = s$. Note that only non-visited nodes are scanned. Therefore, $pt = first_u$ since non arc arriving at node *u* has been previously considered. Again, if the arc $(pt.i, pt.j) \in p_{st}(T)$, the procedure makes $next(pt)$. Otherwise, procedure keeps *pt*. The procedure *SMA* returns a pointer $pt_{\min}$ to the arc (i, j) belonging to $A(T)$ if it exists. Moreover, $pt_{\min}$ is conveniently introduced in $Ve[k]$. Note that, each node $u \in V$ in the tree path from node *s* to node *t* in *T* is

reached at most one time. For each reached node *u*, the best arc $(pt.i, pt.j)$ with $pt.i \in \Gamma^{-}_{pt.j}$ is obtained in constant time. Thus the computational effort performed by this procedure is O(*n*).

We use an additional data structures as in Gabow [4] 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 follow structure to store these trees as a *directed out tree*: $father[k']$ stores the index *p* associated with tree T^{p} and a pointer *pt* to the entering arc (i, j) in *G* that leads to obtain the tree $T^{k'}$ ($father[k']=\{p,\ pt\}$). Using this information, any tree T^{k} can be derived from the initial optimal tree T^{*} by the next procedure.

Procedure (BT) BuildingT (k, $father$, var $pred(T)$, var $d_t(T)$ var T, var u);
$T = T^{*}$; $pred_u(T) = pred_u(T^{*})\ \forall u \in V$; $d_t(T) = d_t(T^{*})$;
Let u=$pt.i$ be the tail node of the arc $father[k].(pt.i, pt.j)$ when $k > 1$; Otherwise $u = t$;
While $(k \neq 1)$ **do**
$T = T \cup \{father[k].(pt.i, pt.j)\} \setminus \{(pred_{pt.j}(T), pt.j)\}$; $pred_{pt.j}(T) = pt.i$; $d_t(T) = d_t(T) + \bar{c}_{pt.ipt.j}(T^{*})$
$k = father[k].p$;

The procedure BT is called with the index *k* of the tree to be constructed. Initally, the procedure set $T = T^{*}$, $pred_u(T) = pred_u(T^{*})$ for all $u \in V$, and $d_t(T) = d_t(T^{*})$ where T^{*} is an optimal tree. The output of this procedure is $T = T^{k}$. The procedure backtracks on the index k using the *father* labels until $k = 1$. This process lets us identify the needed exchanges in T^{*} to obtain *T* and sets $pred_j(T) = i$ to correctly compute T^{k}. Also, procedure computes $d_t(T^{k})$. Remember that when we made a T-exchange with arc $(i, j) \in A(T)$ in *T* then, we set $A(T') = \{(u,l) \in A \setminus T' : u \notin D_j(T) \text{ and } l \notin D_j(T)\}$. In other words, any node in the sub-path from node *i* to node *t* is fixed in T'. Therefore, the procedure BT identifies the tail node *u* = *i* of the arc $father[k].(i, j)$ being the fixed node in the $p_{st}(T)$ with minor depth. Clearly, the procedure BT requires an effort O(*n*) since the number of iterations in the loop is at most $n-1$, that is, the depth of the tree of trees is at most $n-1$.

Taking into account the above notation and remarks, a scheme of the algorithm is:

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

(1) Let T^* be an optimal tree and store $d(T^*)$, $pred(T^*)$ for a tree T^*;

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

(3) Sort the elements in Γ_i^- by it reduced cost $\forall i \in V$;

(4) Create Heap H_i with $i=1,...,n$; Create $Ve[1]$;

(5) SMA($C=\infty$, $pt_{\min}$, T, $Ve[1]$, $pred(T^*)$, t);

(6) **If** ($pt_{\min} \neq$ NULL) **then** Insert $\{pt_{\min}, C+d_t(T), k\}$ in H_1;

(7) **While** $((k < K)$ **and** $(\bigcup_{i=1}^{n} H_i \neq \varnothing))$

(8) $C_{\min}=\infty$; $v_{\min}=0$;

(9) For $v=1$ to n do

(10) Findmin $\{pt, C, p\}$ of H_v;

(11) **If** (($pt \neq$ NULL) and ($C_{\min} > C$)) **then**

(12) $C_{\min}=C$; $pt_{\min}=pt$; $v_{\min}=$ v;

(13) $father[k+1]=\{p, pt_{\min}\}$; Deletemin of $H_{v_{\min}}$;

(14) BT(p, $father$, $pred(T)$, $d_t(T)$ T, u);

(15) SMA($C=\infty$, pt, T, $Ve[p]$, $pred(T)$, u);

(16) **If** ($pt \neq$ NULL) **then** Insert $\{pt, C+d_t(T), p\}$ in $H_{v_{\min}}$;

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

(18) $T=T\cup\{(pt_{\min}.i, pt_{\min}.j)\}\setminus\{(pred_{pt_{\min}.j}(T), pt_{\min}.j)\}$; $pred_{pt_{\min}.j}(T)=pt_{\min}.i$; $d_t(T)=d_t(T)+C_{\min}$;

(19) Create $Ve[k]$; $u=pt_{\min}.i$;

(20) SMA($C=\infty$, pt, T, $Ve[k]$, $pred(T)$, u);

(21) **If** ($pt \neq$ NULL) **then** Insert $\{pt, C+d_t(T), k\}$ in $H_{k \bmod n+1}$;

The algorithm starts with an optimal tree $T=T^*$ and stores labels $d(T^*)$ and $pred(T^*)$. The index of the number of best solutions *k* is set to 1. The heaps H_i with $i=1,...,n$ are created. A pointer *pt* to the arc (i, j) is determined by calling the procedure *SMA*($C=\infty$, $pt_{\min}$, *Ve*[1], $pred(T^*)$, *t*). and the element $\{pt_{\min}, C+d_t(T), 1\}$ is inserted in H_1 wherever $pt_{\min}$ exists. Then, the algorithm starts with a loop until the *K* best solutions are identified or no more feasible solutions are possible. Thus, in an iteration, the first elements of the *n* heaps are extracted and the element with minimum reduced cost among them is selected and deleted from its corresponding heap. This element identifies the way to obtain the $(k+1)th$ best solution. In the algorithm $father[k+1]=\{p, pt_{\min}\}$ lets us determine T^{k+1}. Now, in the algorithm T^p is reconstructed by the procedure BT. Then, the new pointer *pt* to the arc $(i, j)_{A(T^p)}$ arc is found by calling the procedure *SMA* for T^p. The resulting point to an arc (if it exists) and the index *p* are stored in the heap $H_{v_{\min}}$ (heap where $pt_{\min}$ was deleted) using the

key value $C + d_t(T)$ where C is the value calculated in the procedure *SMA*. Next in the algorithm, the index k is increased and T^k is built from T^p by a T-exchange. The $Ve[k]$ is created. Finally, for this new best tree, a pointer *pt* to the arc $(i, j)_{A(T^k)}$ is determined and the element $\{pt,\ C + d_t(T^k),\ k\}$ is inserted in $H_{k \bmod n+1}$.

Theorem 2. *The* KPPSSP *algorithm computes the K point-to-point shortest simple paths in* $O(m + n\log n + K(n + \log\frac{K}{n}))$ *time and O*(*K*+*m*) *space in directed acyclic graph G.*

Proof. In the beginning of the algorithm, the determination of $T^1 = T^*$ and the labels $d(T^*)$ and $pred(T^*)$ requires $O(m)$ time (see Ahuja et al. [1]). Sort the predecessors adjacency node list $\Gamma_i^- = \{ j \in V \mid (j,i) \in A \}$ by it reduced cost for all node $i \in V$ needs $O(m + n\log n)$ time. Line (4) requires O(*n*) time. The procedure *SMA* requires an effort O(*n*) and the operation of create and insert for first time in the heap H_1 takes O(1) time. Clearly, the algorithm makes at most *K* iterations. In each iteration of the algorithm, *SMA* is called twice requiring O(*n*) time overall, procedure BT is called once requiring O(*n*) time and two insert heap operations are made in $O(\log\frac{K}{n})$ since the size of any heap is bound by $\frac{K}{n}$. Obtain the next best solutions, that is, lines (8)-(12) requires O(*n*) time. Remainder operations in the loop are made in O(1) time. Thus, the worst case time complexity of the algorithm is $O(m + n\log n + K(n + \log\frac{K}{n}))$ time. On the other hand, the space required by the algorithm is O(*K* + *m*), since the *father*, and *heap* structures require O(*K*) space; the tree *T* and its corresponding labels need O(*n*) space and the storing the adjacent list uses O(*m*) space. Each $Ve[k]$ contains a number of elements equal to min{*n*-1, *sons*(T^k) + 1} where *sons*(T^k) is the number of trees obtained from T^k by a T-exchange. Clearly $\sum_{k=1}^{K} \min\{n-1, sons(T^k)+1\} \le \sum_{k=1}^{K} sons(T^k)+1 \le 2K$ and therefore, the space used by all vectors is O(*K*). □

References

[1]. Ahuja, R., T. Magnanti, J. B. Orlin. 1993. *Network Flows*. Prentice-Hall, inc.

[2]. Brander, A., M. Sinclair (1995). A comparative study of *K*-shortest path algorithms. *In Proc. Of 11th UK Performance Engineering Workshop* 370-379.

[3]. Eppstein, D. 1999. Finding the *K* shortest paths. *Siam Journal on Computing* **28** 653-674.

[4]. Gabow, H. N. 1977. Two Algorithms for Generating Weighted Spanning Trees in Order. *Siam Journal on Computing* **6** (1) 139-150.

[5]. Hadjiconstantinou, E., N. Chirstofides. (1999). An efficient implementation of an algorithm for finding *K* shortest simple paths. *Networks* **34** 88-101.

[6]. Hershberger, J., M. Maxel, S. Suri. (2003). Finding the *k* Shortest Simple Paths: a new algorithm and its implementation. *Proc. 5th Worksh. Algorithm Engineering & Experiments (ALENEX), SIAM. To appear in ACM transactions on Algorithms* (2007).

[7]. Hoffman, W., R. Pavley. 1959. A method for the solution of the *N*th best path problem. *Journal of the ACM* **6** 506-514.

[8]. Katoh, N., T. Ibaraki, H. Mine. (1982). An efficient Algorithm for *K* Shortest Simple Paths. *Networks* **12** 411-427.

[9]. 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.

[10]. Martins, E.Q.V., M. M. B. Pascoal. 2000. A new implementation of Yen's ranking loopless paths algorithm. *Submitted for publication. Universidade de Coimbra, Portugal*.

[11]. 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*.

[12]. Perko, A. (1986). Implementation of algorithms for K shortest loopless paths. *Networks* **16** 149-160.

[13]. Pollack, M. (1961). The *k*th best route through a network. *Operations Research* **9** 578-580.

[14]. Sedeño-Noda, A., C. González-Martín. 2006. Shortest Path Simplex Algorithm with a Multiple Pivot Rule. *Tecnival Report nº 2, Departamento de Estadística, Investigación Operativa y Computación*.

[15]. Yen, J. Y. 1971. Finding the *K* shortest loopless paths in a network. *Management Science* **17** 712-716.

[16]. 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**.