Efficient determination of the k most vital edges for the minimum spanning tree problem

Cristina Bazgan[1,2] Sonia Toubaline[1] Daniel Vanderpooten[1]

1. Université Paris-Dauphine, LAMSADE
Place du Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France
2. Institut Universitaire de France
{bazgan,toubaline,vdp}@lamsade.dauphine.fr

Abstract

We study in this paper the problem of finding in a graph a subset of k edges whose deletion causes the largest increase in the weight of a minimum spanning tree. We propose for this problem an explicit enumeration algorithm whose complexity, when compared to the current best algorithm, is better for general k but very slightly worse for fixed k. More interestingly, unlike in the previous algorithms, we can easily adapt our algorithm so as to transform it into an implicit enumeration algorithm based on a branch and bound scheme. We also propose a mixed integer programming formulation for this problem. Computational results show a clear superiority of the implicit enumeration algorithm both over the explicit enumeration algorithm and the mixed integer program.

Key words: most vital edges, minimum spanning tree, exact algorithms, mixed integer program.

1 Introduction

In many applications involving the use of communication or transportation networks, we often need to identify critical infrastructures. By critical infrastructure we mean a set of links whose damage causes the largest perturbation within the network. Modeling this network as a weighted graph, identifying critical infrastructures amounts to finding a subset of edges whose removal from the graph causes the largest increase in the total weight. In the literature this problem is referred to as the k most vital edges problem. In this paper, we are interested in determining a subset of edges of the graph whose deletion causes the largest increase in the weight of a minimum spanning tree (MST). This problem is referred to as k MOST VITAL EDGES MST.

The problem of finding the k most vital edges of a graph has been investigated for various problems including shortest path [1, 9, 13], maximum flow [20, 16, 21], 1-median and 1-center [2]. For the minimum spanning tree problem defined on a graph G with n vertices and m edges, Frederickson and Solis-Oba [6] showed that, for general k, k MOST VITAL EDGES MST is NP-hard and proposed an $O(\log k)$-approximation algorithm. The problem remains NP-hard even for complete graphs with weights 0 or 1 and 3-approximable for graphs with weights 0 or 1 [3]. For a fixed k the problem is obviously polynomial. The case $k = 1$ has been largely studied in the literature [7, 8, 18]. Hsu *et al.* [7] gave two algorithms that run in $O(m \log m)$ and

$O(n^2)$. Iwano and Katoh [8] proposed an algorithm in $O(m\alpha(m,n))$ using Tarjan's result [19], where α is the inverse Ackermann function. Pettie [14] improved the results of Tarjan [19] and Dixon *et al.* [5], giving rise to the current best deterministic algorithm in $O(m \log \alpha(m,n))$. For general k, several exact algorithms based on an explicit enumeration of possible solutions have been proposed [10, 11, 17]. The best one [10] runs in time $O(n^k \alpha((k+1)(n-1),n))$ and was achieved by reducing G to a sparse graph. Using Pettie's result [14], the running time of the later algorithm becomes $O(n^k \log \alpha((k+1)(n-1),n))$.

In this paper we propose a new efficient algorithm also based on an explicit enumeration of all possible solutions for k Most Vital Edges MST. Its complexity $O(n^k \log \alpha(2(n-1),n))$ for fixed k is theoretically very slightly worse than the complexity of the algorithm proposed by Liang [10] using Pettie's result [14]. However, given the fact that $\alpha(m,n)$ is always less than 4 in practice, the complexity of these two algorithms can be deemed as equivalent. Moreover, the complexity of our algorithm is better than that of Liang's algorithm for general k. More interestingly, unlike any other algorithm, our algorithm has two specific useful features. First, it can also determine an optimal solution for i Most Vital Edges MST, for each $1 \leq i \leq k$, with the same time complexity. Second, it can be easily adapted to establish an implicit enumeration algorithm based on a branch and bound procedure. We also present in this paper a mixed integer programming formulation to solve k Most Vital Edges MST. We implement and test all these proposed algorithms using, for the implicit enumeration algorithm, different branching and evaluation strategies. The results show that the implicit enumeration algorithm is much faster than the explicit enumeration algorithm as well as the resolution of the mixed integer program. Moreover, the implicit enumeration algorithm can handle significantly larger instances due to a better use of memory space. Finally, we also propose an ε-approximate algorithm.

The rest of the paper is organized as follows. In section 2 we introduce notations and some results related to our problem. In section 3 we present a new explicit enumeration algorithm that solves k Most Vital Edges MST. In section 4 we propose another exact algorithm based on an implicit enumeration scheme. In section 5, we present a mixed integer programming formulation for k Most Vital Edges MST. Computational results are presented in section 6. We also include in these experiments an ϵ-approximate version of our implicit enumeration schema. Conclusions are provided in section 7.

2 Basic concepts and preliminary results

Let $G = (V,E)$ be a weighted undirected connected graph with $|V| = n$, $|E| = m$ where $w(e) \geq 0$ is the integer weight of each edge $e \in E$. We denote by $G - E'$ the graph obtained from G by removing the subset of edges $E' \subseteq E$. k Most Vital Edges MST consists of finding a subset of edges $S^* \subseteq E$ with $|S^*| = k$ that maximizes the weight of a MST in the graph $G - S^*$. We assume that G is at least $(k+1)$ edge-connected, since otherwise any selection of k edges including the edges of a minimum unweighted cut is a trivial solution. Therefore, we assume $k \leq \lambda(G) - 1$, where $\lambda(G)$ is the edge-connectivity of G. Also, without loss of generality, we suppose in the following that all weights are different (by introducing, if necessary, an arbitrary total order on edges with the same weight). This assumption implies the uniqueness of minimum spanning trees or forests. For a non necessarily connected graph, a minimum spanning forest (MSF) is the union of minimum spanning trees for each of its connected components. In this paper a tree or a forest is considered as a graph but also, for convenience, as a subset of edges. For a set of edges F, $w(F)$ represents the sum of the weights of the edges in F.

We denote by T_0 the MST of G. Remark that an optimal solution of k Most Vital Edges MST must contain at least one edge of T_0. For $i \geq 1$, let T_i be the MSF of the graph $G_i = G - \cup_{j=0}^{i-1} T_j$. We use in the following the graph $U_k^G = (V, \cup_{j=0}^{k} T_j)$ which has the following interesting property.

Lemma 1 *(Liang and Shen [11]) For any $S \subseteq E$, $|S| \leq k$, any edge of the MST of graph $G - S$ belongs to U_k^G.*

By Lemma 1, solving k Most Vital Edges MST on G reduces to solving the same problem on the sparser graph U_k^G whose number of edges is at most $(k+1)(n-1)$.

Considering T a MST of a graph, the replacement edge $r(e)$ for an edge $e \in T$ is defined as the edge $e' \neq e$ of minimum weight which connects the two disconnected components of $T \backslash \{e\}$. The sensitivity of a minimum spanning tree T, i.e. the allowable variation for each edge weight so that T remains a minimum spanning tree, can be computed in $O(m \log \alpha(m,n))$ [14]. In particular, for edges in T, this algorithm provides replacement edges. As a consequence, we get the following result.

Lemma 2 1 Most Vital Edges MST *defined on a graph with n vertices and m edges is solvable in $O(m \log \alpha(m,n))$.*

Proof: Let T^* be the minimum spanning tree in a given graph. We calculate the replacement edges $r(e)$ for all edges $e \in T^*$. The most vital edge is the edge e^* such that $w(r(e^*)) - w(e^*) = \max_{e \in T^*} w(r(e)) - w(e)$. □

Actually, replacement edges belong to a specific subset of edges as shown by the following result.

Lemma 3 *For each edge $e \in T_i$, we have $r(e) \in T_{i+1}$ for $i = 0, \ldots, k-1$.*

Proof: Given a graph G, Liang [10] shows that for each edge $e \in T_0$, $r(e) \in T_1$. Applying this to graph G_i, for which T_i is the MSF, we get the result. □

3 An explicit enumeration algorithm for finding the k most vital edges

We propose an algorithm that constructs a search tree of depth $k-1$ in a breadth-first mode. At the i^{th} level of this search tree, $i = 0, \ldots, k-1$, a node s is characterized by:

- $mv(s)$: a subset of i edges, corresponding to a tentative partial selection of the k most vital edges.
- $\widetilde{U}(s) = U_{k-|mv(s)|}^{G'(s)}$ where $G'(s) = (V, E \backslash mv(s))$. Hence, we have

  $\widetilde{U}(s) = (V, \cup_{i=0}^{k-|mv(s)|} T_i(s))$ where $T_i(s)$ is the MSF in $G'(s) - \cup_{j=0}^{i-1} T_j(s)$.
- $mst(s)$: a subset of edges forbidden to deletion. These edges belonging to $T_0(s)$, will necessary belong to any MST associated with any descendant of s. Depending on the position of s in the search tree, the cardinality of $mst(s)$ varies from 0 to $n-2$.

Denote by N_i, for $i = 0, \ldots, k-1$, the set of nodes of the search tree at the i^{th} level. We describe in the following the exact algorithm (section 3.1) and exemplify its use on an illustrative example (section 3.2).

3.1 Description of the algorithm

We first construct the graph U_k^G. Let a be the root of the search tree with $mv(a) = mst(a) = \emptyset$, $\widetilde{U}(a) = U_k^G$, $w(T_0(a)) = w(T_0)$, and $N_0 = \{a\}$.

For a level i, $0 \leq i \leq k-2$, we compute for each node $s \in N_i$ and each edge $e \in T_0(s)$, the replacement edges $r(e)$ in $T_1(s)$. Node s gives rise to $|T_0(s) \backslash mst(s)|$ children in N_{i+1}. Each such child d, corresponding to an edge e_j in $T_0(s) \backslash mst(s) = \{e_1, \ldots, e_{n-1-|mst(s)|}\}$, is characterized by:

- $mv(d) = mv(s) \cup \{e_j\}$.
- $mst(d) = mst(s) \cup (\cup_{\ell=1}^{j-1} \{e_\ell\})$.
- $\widetilde{U}(d)$ is updated from $\widetilde{U}(s)$ as follows (using Lemma 3):
    - $T_0(d) = T_0(s) \cup \{r(e_j)\} \setminus \{e_j\}$ and hence $w(T_0(d)) = w(T_0(s)) - w(e_j) + w(r(e_j))$.
    - For $j = 1, \ldots, k - |mv(d)|$, $T_j(d)$ is obtained from $T_j(s)$ by deleting the replacement edge e_{rep} of the edge deleted from $T_{j-1}(s)$ and replacing it by its replacement edge $r(e_{rep}) \in T_{j+1}(s)$.

      If for a level i and an edge e_{rep}, the replacement edge $r(e_{rep})$ does not exist, $T_j(d) = T_j(s) \setminus \{e_{rep}\}$ and $T_\ell(d) = T_\ell(s)$ for $\ell = j+1, \ldots, k - |mv(d)|$.

      If for a level i, $T_i(s) = \emptyset$ then $T_\ell(d) = \emptyset$ for $\ell = i, \ldots, k - |mv(d)|$.

At level $k-1$, for each node $s \in N_{k-1}$ and for all edges $e \in T_0(s) \setminus mst(s)$, we find $r(e)$ in $T_1(s)$ and we determine a node s^* that verifies $\max_{s \in N_{k-1}} \max_{e \in T_0(s) \backslash mst(s)} (w(T_0(s)) - w(e) + w(r(e)))$. An optimal solution is the subset $mv(s^*) \cup \{e^*\}$ where $e^* = \arg \max_{e \in T_0(s^*) \backslash mst(s^*)} w(T_0(s^*)) - w(e) + w(r(e))$. The largest weight of a MST in the partial graph obtained by deleting this subset is $w(T_0(s^*)) - w(e^*) + w(r(e^*))$.

Algorithm 1 describes this procedure. Its correctness and complexity are given in Theorem 1.

Theorem 1 *Algorithm 1 computes an optimal solution for an instance of k* Most Vital Edges MST *with n vertices and m edges in $O(km\alpha(m,n) + n^k \log \alpha(2(n-1), n))$ time.*

Proof: We first show that Algorithm 1 gives an optimal solution for k Most Vital Edges MST. Let S^* be the solution returned by Algorithm 1, and w^* the weight of the MST in $U_k^G - S^*$. Consider any solution S', with $|S'| = k$, and w' the weight of the MST in $U_k^G - S'$. Let r be a node of the search tree such that $mv(r) \subseteq S'$ and for any child d of r, $mv(d) \nsubseteq S'$. Clearly, r exists and corresponds at worst to root a when $S' \cap T_0 = \emptyset$. Since, by definition, r is such that no edge of $T_0(r)$ belongs to S', we have $w' = w(T_0(r))$. Moreover, since $w(T_0(r)) \leq w^*$, we have $w' \leq w^*$.

Algorithm 1: Explicit resolution of k MVE MST

/* Let a be the root of the search tree */
Construct U_k^G;
$mv(a) \leftarrow \emptyset; mst(a) \leftarrow \emptyset; w(T_0(a)) \leftarrow w(T_0); \widetilde{U}(a) \leftarrow U_k^G$;
$N_0 \leftarrow \{a\}; N_i \leftarrow \emptyset, i = 1, \ldots, k-1$;
for $i \leftarrow 0$ **to** $k-2$ **do**
    forall $s \in N_i$ **do**
        forall $e \in T_0(s)$ **do**
            find $r(e)$ in $T_1(s)$;
        /* $T_0(s) \backslash mst(s) = \{e_1, \ldots, e_{n-1-|mst(s)|}\}$ */
        forall $e_j \in T_0(s) \backslash mst(s)$ **do**
            /* create a new node d, a child of s */
            $mv(d) \leftarrow mv(s) \cup \{e_j\}$;
            $w(T_0(d)) \leftarrow w(T_0(s)) - w(e_j) + w(r(e_j))$;
            $mst(d) \leftarrow mst(s) \cup (\cup_{\ell=1}^{j-1}\{e_\ell\})$;
            determine $\widetilde{U}(d)$ by using Algorithm 2;
            $N_{i+1} \leftarrow N_{i+1} \cup \{d\}$;
$max \leftarrow 0$;
forall $s \in N_{k-1}$ **do**
    forall $e \in T_0(s)$ **do**
        find $r(e)$ in $T_1(s)$;
    forall $e \in T_0(s) \backslash mst(s)$ **do**
        if $w(T_0(s)) - w(e) + w(r(e)) > max$ **then**
            $max \leftarrow w(T_0(s)) - w(e) + w(r(e))$;
            $e^* \leftarrow e$;
            $s^* \leftarrow s$;
/* The largest weight of a MST in the partial obtained graph is $w(T_0(s^*)) - w(e^*) + w(r(e^*))$ */
return $S^* = mv(s^*) \cup \{e^*\}$;

Algorithm 2: Construction of $\widetilde{U}(d)$ from $\widetilde{U}(s)$ where d is the child of s in the search tree obtained from s by deleting e_j

$T_0(d) \leftarrow T_0(s) \cup \{r(e_j)\} \backslash \{e_j\}$;
$replace \leftarrow r(e_j)$;
$\ell \leftarrow 0$;
while $T_{\ell+1}(s) \neq \emptyset$ **do**
    if $replace$ $exists$ **then**
        Determine if there exists a replacement edge, called $replace1$, of $replace$ in $T_{\ell+1}(s)$
        if $replace1$ $exists$ **then**
            $T_\ell(d) \leftarrow T_\ell(s) \cup \{replace1\} \backslash \{replace\}$;
            $replace \leftarrow replace1$;
        else
            $T_\ell(d) \leftarrow T_\ell(s) \backslash \{replace\}$;
    else
        $T_\ell(d) \leftarrow T_\ell(s)$;
    $\ell \leftarrow \ell + 1$;
return $\widetilde{U}(d)$;

We determine now the complexity of Algorithm 1. Denote by t_u the time for constructing U_k^G, by $t_{edge-rep}$ the time for finding the replacement edges for all edges of a minimum spanning tree, and by t_{gen} the time for generating any node s of the search tree (that is determining $mv(s), mst(s)$ and $\widetilde{U}(s)$). Level 0 requires $|N_0| t_{edge-rep}$ time. Level i takes $|N_i| t_{edge-rep} + |N_i| t_{gen}$ time, for $1 \leq i \leq k-1$. At level k, we compute the k most vital edges. Thus, the total time of Algorithm 1 is given by

$$t_u + \sum_{i=0}^{k-1} |N_i| t_{edge-rep} + \sum_{i=1}^{k-1} |N_i| t_{gen} + |N_k|$$

For each node $s \in N_i$, subset $mv(s)$ consists of ℓ tree edges of $T_0(a)$ and $(i-\ell)$ edges belonging to the union set of the $(i-\ell)$ replacement edges of these ℓ edges, $1 \leq \ell \leq i$ (the p replacement edges of an edge $e \in T_0(a)$ are the p edges of minimum weight which connect the two disconnected components of $T_0(a) \backslash \{e\}$). This implies that $|N_i| = \sum_{\ell=1}^{i} \binom{n-1}{\ell} K_\ell^{i-\ell} = \sum_{\ell=1}^{i} \binom{n-1}{\ell}\binom{i-1}{i-\ell} = \binom{n+i-2}{i} = O(n^i)$, where $K_n^p = \binom{n+p-1}{p}$ is the number of combinations with repetition of p elements chosen from a set of n elements.

For a node $s \in N_i$, $1 \leq i \leq k-1$, $\widetilde{U}(s)$ contains at most $k-i+1$ forests. Then, t_{gen} is in $O((k-i+1)n)$ time. Since the replacement edges of a MST in a graph with n vertices and m edges can be computed in $O(m \log \alpha(m,n))$ [14], $t_{edge-rep}$ is in $O(n \log \alpha(2(n-1),n))$ time. The construction of U_k^G, corresponding to t_u, can be performed in $O(km\alpha(m,n))$ time, using k times the best current algorithms for MST [4, 15]. Therefore, the complexity of Algorithm 1 is in $O(km\alpha(m,n) + n^k \log \alpha(2(n-1),n))$ time. Note that the time needed to generate all the nodes of the search tree is dominated by the total time to find, for all nodes s of the search tree, the replacement edges $r(e)$ in $T_1(s)$ for all edges $e \in T_0(s)$. □

Remark For each node s of the search tree, we could use, instead of the graph $\widetilde{U}(s)$, the graph $U(s) = U_{k-|mv(s)|}^{G''(s)}$ where $G''(s)$ is the graph obtained from G by contracting the edges of $mst(s)$ and removing the edges of $mv(s)$. Thus, $U(s) = (V, \cup_{i=0}^{k-|mv(s)|} T_i(s))$ where $T_i(s)$ is the MSF of $G''(s) - \cup_{j=0}^{i-1} T_j(s)$. Unfortunately, given a child d of a node s of the search tree, updating efficiently $U(d)$ from $U(s)$ is not as straightforward as for $\tilde{U}$. However, even if updating U could be performed more efficiently than $\widetilde{U}$, we would get the same complexity since the time for generating all nodes of the search tree is dominated by the total time for finding the replacement edges for all nodes in the search tree.

We close this subsection by comparing our algorithm with the previously best known algorithm for k Most Vital Edges MST. For fixed k, by using the result of Dixon *et al.* [5], Liang [10] proposes an algorithm to solve k Most Vital Edges MST in $O(n^k \alpha((k+1)(n-1),n))$ time. Using Pettie's result [14] Liang's algorithm can be implemented in $O(t_u + n^k \log \alpha((k+1)(n-1),n))$ time, where t_u is the time for constructing U_k^G. Our algorithm has a complexity that is theoretically slightly worse than that of Liang. Nevertheless, since $\alpha(m,n)$ is always less than or equal to 4 in practice, the complexity of these two algorithms can be considered as equivalent. Moreover, a specific advantage of our algorithm is that it can also determine, with the same time complexity, an optimal solution for i Most Vital Edges MST, for $1 \leq i \leq k$. Indeed, at each level i, we can find among nodes of N_i, the node with the largest weight of a MST.

For general k, our bound is clearly better than that of Liang. Indeed, in Liang's algorithm, after the determination of U_k^G, Liang divides the problem into two cases: (i) $|T_0 \cap S^*| = i, 1 \leq i < k$ and (ii) $|T_0 \cap S^*| = k$ where S^* represents a subset of k most vital edges. In (i), for every possible combination of i edges among the $n-1$ edges of T_0, $1 \leq i < k$, the author constructs a specific graph $\mathcal{G}$ with a number of nodes and edges depending only on k, and determines the $k-i$ remaining edges in $\mathcal{G}$. In (ii), from every possible choice of $(k-1)$

edges among the $n-1$ edges of T_0, the author constructs a MST T' in the graph obtained by deleting these $(k-1)$ edges and finds the k^{th} edge to be removed by using the replacement edges of T'. Therefore, (i) and (ii) are performed respectively in $\sum_{i=1}^{k-1} \binom{n-1}{i} (t_{\mathcal{G}} + t_{k-i})$ and $\binom{n-1}{k-1} t_{last}$ time, where $t_{\mathcal{G}}$, t_{k-i} and t_{last} are respectively the time to construct $\mathcal{G}$, the time to determine the $k-i$ remaining edges to be removed from $\mathcal{G}$ and the time to find the k^{th} edge to be removed from $T' \cap T_0$. Note that Liang, who considers only the case where k is fixed, does not need to explicit the term involving t_{k-i}. However, for general k, even if expressing the complexity of his algorithm by $O(t_u + k^3 n^k + \sum_{i=1}^{k-1} \binom{n-1}{i} t_{k-i} + kn^k \log \alpha((k+1)(n-1), n))$, one can observe that it is relatively larger than the complexity of our proposed algorithm that remains in $O(t_u + n^k \log \alpha(2(n-1), n))$ time.

The other exact algorithms proposed in the literature [11, 17] have a worse complexity than our algorithm both for fixed and general k.

3.2 An illustrative example for the explicit enumeration algorithm

In this section, we apply Algorithm 1 to solve 3 Most Vital Edges MST on the graph G, illustrated in Figure 1. The bold edges represent the MST of G.

We start the construction of the search tree with the root a whose elements are $mv(a) = mst(a) = \emptyset$, $\tilde{U}(a)$ is the union of the following forests

$T_0(a) : (1, 2), (1, 3), (3, 5), (4, 5)$

$T_1(a) : (2, 3), (3, 4), (1, 4), (2, 5)$

$T_2(a) : (2, 4), (1, 5)$

and $w(T_0(a)) = 12$. We omitted $T_3(a)$ since it is an empty set.

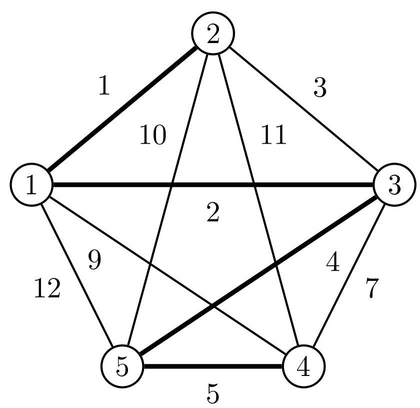

Figure 1: Graph G

The replacement edges of each edge in $T_0(a)$ are given in Table 1.

Level 1 contains four children of a and thus $N_1 = \{1, 2, 3, 4\}$. The elements of each node $s \in N_1$ are given in Table 2 where for example, $\tilde{U}(3)$ is constructed as follows: $T_0(3)$ is obtained by removing $(3, 5)$ from $T_0(a)$ and adding its replacement edge $(3, 4)$. For $T_1(3)$, we delete $(3, 4)$ from $T_1(a)$ and find its replacement edge among the edges in $T_2(a)$ which is $(2, 4)$. Finally, $T_2(3)$ is obtained by removing $(2, 4)$ from $T_2(a)$.

e	$(1,2)$	$(1,3)$	$(3,5)$	$(4,5)$
$r(e)$	$(2,3)$	$(2,3)$	$(3,4)$	$(3,4)$

Table 1: Replacement edges of $e \in T_0(a)$

s		1	2	3	4
$mv(s)$		$\{(1,2)\}$	$\{(1,3)\}$	$\{(3,5)\}$	$\{(4,5)\}$
$mst(s)$		$\emptyset$	$\{(1,2)\}$	$\{(1,2),(1,3)\}$	$\{(1,2),(1,3),(3,5)\}$
$w(T_0(s))$		14	13	15	14
$\widetilde{U}(s)$	T_0	$(1,3),(2,3),(3,5),(4,5)$	$(1,2),(2,3),(3,5),(4,5)$	$(1,2),(1,3),(4,5),(3,4)$	$(1,2),(1,3),(3,5),(3,4)$
	T_1	$(3,4),(1,4),(2,5),(2,4)$	$(3,4),(1,4),(2,5),(2,4)$	$(2,3),(1,4),(2,5),(2,4)$	$(2,3),(1,4),(2,5),(2,4)$
	T_2	$(1,5)$	$(1,5)$	$(1,5)$	$(1,5)$

Table 2: Elements of $s \in N_1$

We construct level 2 in the same way. The search tree is represented in Figure 2. The elements of each node $s \in N_2 = \{5, 6, \dots, 14\}$ are given in Table 3.

The construction of the search tree is completed. After determining the replacement edges of edges in T_0 for all leaves of the search tree, we find that s^* is node 5, a solution for 3 Most Vital Edges MST is $\{(1,2),(1,3),(2,3)\}{=}mv(5) \cup \{e^*\}$ and the weight of the MST in $G \setminus \{(1,2),(1,3),(2,3)\}$ is 28.

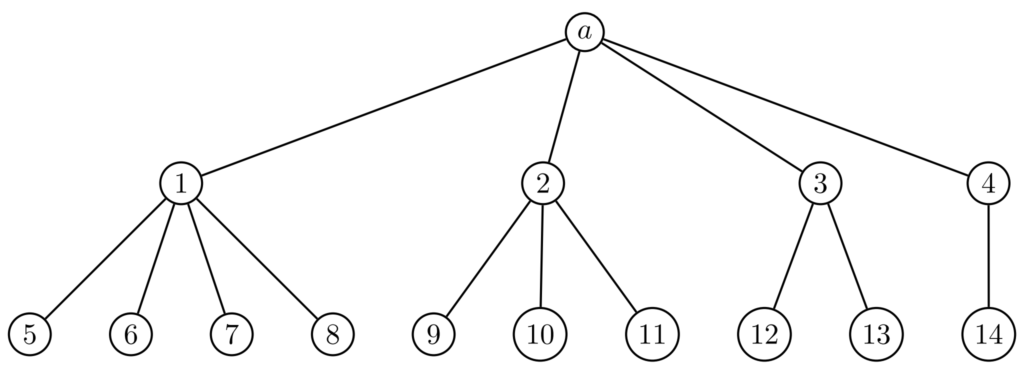

Figure 2: Search tree

Observe that the most vital edge is not necessarily the edge of smallest weight. Moreover, the most vital edge is not necessarily included in any optimal solution of k Most Vital Edges MST for $k \geq 2$.

4 An implicit enumeration algorithm for finding the k most vital edges

An interesting feature of our explicit enumeration algorithm is that, unlike the algorithms previously proposed, it can easily be adapted to design an implicit algorithm based on a branch and bound scheme. To do this, we use for each node s an upper bound $UB(s)$ based on successive replacements of edges. We also use lower bounds $LB(s)$ constructed by extending the forest, corresponding to s, to a particular minimum spanning tree.

s		5	6	7	8
$mv(s)$		$\{(1,2),(1,3)\}$	$\{(1,2),(2,3)\}$	$\{(1,2),(3,5)\}$	$\{(1,2),(4,5)\}$
$mst(s)$		$\emptyset$	$\{(1,3)\}$	$\{(1,3),(2,3)\}$	$\{(1,3),(2,3),(3,5)\}$
$w(T_0(s))$		21	21	17	16
$\widetilde{U}(s)$	T_0	$(2,3),(3,5),(4,5),(1,4)$	$(1,3),(3,5),(4,5),(2,5)$	$(1,3),(2,3),(4,5),(3,4)$	$(1,3),(2,3),(3,5),(3,4)$
	T_1	$(3,4),(2,5),(2,4),(1,5)$	$(3,4),(1,4),(2,4),(1,5)$	$(1,4),(2,5),(2,4)$	$(1,4),(2,5),(2,4)$
s		9	10	11	12
$mv(s)$		$\{(1,3),(2,3)\}$	$\{(1,3),(3,5)\}$	$\{(1,3),(4,5)\}$	$\{(3,5),(4,5)\}$
$mst(s)$		$\{(1,2)\}$	$\{(1,2),(2,3)\}$	$\{(1,2),(2,3),(3,5)\}$	$\{(1,2),(1,3)\}$
$w(T_0(s))$		19	16	15	17
$\widetilde{U}(s)$	T_0	$(1,2),(3,5),(4,5),(1,4)$	$(1,2),(2,3),(4,5),(3,4)$	$(1,2),(2,3),(3,5),(3,4)$	$(1,2),(1,3),(3,4),(2,5)$
	T_1	$(3,4),(2,5),(2,4),(1,5)$	$(1,4),(2,5),(2,4)$	$(1,4),(2,5),(2,4)$	$(2,3),(1,4),(2,4),(1,5)$
s		13	14		
$mv(s)$		$\{(3,5),(3,4)\}$	$\{(4,5),(3,4)\}$		
$mst(s)$		$\{(1,2),(2,3),(4,5)\}$	$\{(1,2),(2,3),(3,5)\}$		
$w(T_0(s))$		20	16		
$\widetilde{U}(s)$	T_0	$(1,2),(2,3),(4,5),(1,4)$	$(1,2),(2,3),(3,5),(1,4)$		
	T_1	$(2,3),(2,5),(2,4),(1,5)$	$(2,3),(2,5),(2,4),(1,5)$		

Table 3: Elements of $s \in N_2$

In order to obtain the best possible bounds, we construct $U(s)$ for each node s, instead of using $\widetilde{U}(s)$. For each child d of s, $U(d)$ is determined by constructing $T_i(d)$, for $0 \leq i \leq k - |mv(d)|$ from the edges of $U(s)$.

4.1 Lower bounds

For a fixed node s of the search tree, $k - |mv(s)|$ edges remain to be deleted from $U(s)$. We present different ways of determining these remaining edges giving rise to three possible lower bounds.

1. $LB_{greedy}(s)$: Given $T_0(s)$, we compute $r(e_j)$ for all $e_j \in T_0(s)$. We delete the edge e_j^* which attains $max_{e_j \in T_0(s) \setminus mst(s)}(w(r(e_j)) - w(e_j))$ and replace it by $r(e_j^*)$. We update $U(s)$ and repeat the process until $k - |mv(s)|$ edges are removed. The value of this bound is the weight of the last MST obtained.

2. $LB_{first}(s)$: We remove the $k - |mv(s)|$ edges of $T_0(s) \setminus mst(s)$ having the smallest weight, and we construct a MST from the remaining edges in $T_0(s)$. The value of this bound is the weight of the MST obtained.

3. $LB_{best}(s)$: Given $T_0(s)$, we compute $r(e_j)$ for all $e_j \in T_0(s)$. We remove the $k - |mv(s)|$ edges in $T_0(s) \setminus mst(s)$ whose difference between the weight of their replacement edge and their weight is the largest, and we construct a MST from the remaining edges in $T_0(s)$. The value of this bound is the weight of the MST obtained.

In order to test these bounds, we computed, for instances with different values of n and k, these three lower bounds at the root a of the search tree. The instances are generated as explained in section 6. The results are given in Tables 4 and 5 where we report, for each lower bound, its value, as well as the percent deviation from the optimal value $\frac{opt - LB}{opt}$, and the time to compute it. We note that there is no dominance between these three bounds. We also note that LB_{first} is the fastest in terms of running time but gives bad values. LB_{greedy}, which gives the best values in most cases, takes much more time than the other bounds. LB_{best}, which gives similar values as LB_{greedy}, takes only about twice as much time as LB_{first} and about 40 to 100 times less time than LB_{greedy}.

n	k	$LB_{greedy}(a)$			$LB_{k\ first}(a)$			$LB_{k\ best}(a)$			$w(T_0)$	$UB(a)$	
		value	%	time	value	%	time	value	%	time	in $G \setminus S^*$	value	%
20	9	265	6.0	0.873	255	9.6	0.016	250	11.3	0.047	282	719	155.0
		221	3.5	0.889	219	4.4	0.015	222	3.1	0.032	229	711	210.5
		178	15.6	0.982	179	15.2	0.032	180	14.7	0.031	211	669	217.1
		166	10.8	0.842	157	15.6	0.000	157	15.6	0.016	186	681	266.1
		276	0.7	0.624	268	3.6	0.015	267	4.0	0.016	278	726	161.2
		246	11.8	0.904	243	12.9	0.016	240	14.0	0.000	279	764	173.8
		236	1.3	0.764	232	2.9	0.031	235	1.7	0.047	239	682	185.4
		272	0.0	0.967	254	6.6	0.031	255	6.3	0.031	272	712	161.8
		205	1.0	1.060	193	6.8	0.016	203	1.9	0.000	207	668	222.7
		245	1.6	0.748	216	13.3	0.000	225	9.6	0.016	249	716	187.6
average			5.2	0.865		9.1	0.017		8.2	0.024			194.1
25	8	174	1.7	1.045	146	17.5	0.031	172	2.8	0.047	177	491	177.4
		191	1.5	0.936	173	10.8	0.000	191	1.5	0.016	194	474	144.3
		215	0.5	0.998	164	24.1	0.000	210	2.8	0.016	216	533	146.8
		240	4.4	0.951	219	12.7	0.031	232	7.6	0.047	251	523	108.4
		180	2.7	1.232	169	8.6	0.016	174	5.9	0.015	185	528	185.4
		202	2.4	0.967	200	3.4	0.031	202	2.4	0.047	207	491	137.2
		218	3.1	1.185	209	7.1	0.016	218	3.1	0.000	225	582	158.7
		183	5.7	0.904	179	7.7	0.032	180	7.2	0.046	194	498	156.7
		216	5.3	0.982	211	7.5	0.032	215	5.7	0.046	228	564	147.4
		235	4.9	0.982	243	1.6	0.000	232	6.1	0.016	247	562	127.5
average			3.2	1.018		10.1	0.019		4.5	0.030			149.0
30	7	153	0.6	0.920	126	18.2	0.016	147	4.5	0.015	154	339	120.1
		160	14.4	0.982	150	19.8	0.032	158	15.5	0.046	178	426	127.8
		187	1.6	1.232	164	13.7	0.031	181	4.7	0.063	190	385	102.6
		164	3.5	1.170	157	7.6	0.015	164	3.5	0.016	170	392	130.6
		198	4.8	0.982	187	10.1	0.016	197	5.3	0.016	208	399	91.8
		181	2.7	1.014	155	16.7	0.015	175	5.9	0.016	186	398	114.0
		194	1.5	0.889	180	8.6	0.015	193	2.0	0.016	197	402	104.1
		186	5.1	1.155	156	20.4	0.015	168	14.3	0.016	196	403	105.6
		243	0.8	1.170	214	12.7	0.016	228	6.9	0.015	245	455	85.7
		197	0.5	0.753	184	7.1	0.012	195	1.5	0.016	198	406	105.1
average			3.6	1.027		13.5	0.018		6.4	0.024			108.7
50	5	190	17.4	1.373	160	30.4	0.078	189	17.8	0.031	230	281	22.2
		230	0.0	1.263	215	6.5	0.016	230	0.0	0.031	230	288	25.2
		174	1.1	1.435	161	8.5	0.031	174	1.1	0.031	176	264	50.0
		161	1.8	1.357	155	5.5	0.015	161	1.8	0.032	164	231	40.9
		202	0.0	1.232	185	8.4	0.016	199	1.5	0.015	202	266	31.7
		177	0.6	1.311	165	7.3	0.031	177	0.6	0.015	178	254	42.7
		176	2.2	1.311	157	12.8	0.015	176	2.2	0.031	180	262	45.6
		191	2.1	1.372	191	2.1	0.016	191	2.1	0.016	195	262	34.4
		173	1.7	1.201	153	13.1	0.016	173	1.7	0.016	176	245	39.2
		167	1.2	1.248	151	10.7	0.000	166	1.8	0.031	169	232	37.3
average			2.8	1.310		10.5	0.023		3.1	0.025			36.9
50	7	164	1.2	2.012	148	10.8	0.016	166	0.0	0.031	169	291	72.2
		185	1.6	2.012	168	10.6	0.016	184	2.1	0.031	188	311	65.4
		156	8.2	2.231	146	14.1	0.016	156	8.2	0.031	170	304	78.8
		182	1.6	1.997	173	6.5	0.016	182	1.6	0.031	185	319	72.4
		222	0.0	1.903	207	6.8	0.015	219	1.4	0.016	222	355	59.9
		191	4.0	2.090	178	10.6	0.016	191	4.0	0.031	199	331	66.3
		180	0.6	2.044	163	9.9	0.015	177	2.2	0.063	181	299	65.2
		209	0.0	2.246	189	9.6	0.016	207	1.0	0.031	209	326	56.0
		205	1.4	2.184	193	7.2	0.046	205	1.4	0.032	208	343	64.9
		196	0.0	2.013	185	5.6	0.015	196	0.0	0.016	196	315	60.7
average			2.0	2.073		9.3	0.019		2.4	0.031			66.2
75	5	172	0.6	2.309	159	8.1	0.016	172	0.6	0.031	173	215	24.3
		159	0.6	2.496	149	6.9	0.016	159	0.6	0.031	160	225	40.6
		181	0.0	2.371	172	5.0	0.078	181	0.0	0.032	181	229	26.5
		181	0.0	2.637	168	7.2	0.031	181	0.0	0.031	181	228	26.0
		168	1.2	2.855	159	6.5	0.016	168	1.2	0.031	170	219	28.8
		197	0.5	2.481	189	4.5	0.015	197	0.5	0.031	198	248	25.3
		168	1.2	2.247	158	7.1	0.015	170	0.0	0.047	170	218	28.2
		200	0.0	2.450	183	8.5	0.015	200	0.0	0.031	200	258	29.0
		184	1.6	2.153	165	11.8	0.031	184	1.6	0.031	187	248	32.6
		186	0.5	2.418	176	5.9	0.032	186	0.5	0.031	187	236	26.2
average			0.6	2.442		7.1	0.027		0.5	0.33			28.8
75	7	227	0.0	3.572	214	5.7	0.016	226	0.4	0.046	227	329	44.9
		239	1.6	3.510	225	7.4	0.031	239	1.6	0.032	243	332	36.6
		189	0.0	3.666	162	14.3	0.031	189	0.0	0.032	189	272	43.9
		206	0.5	3.417	188	9.2	0.031	206	0.5	0.047	207	299	44.4
		187	1.6	3.573	170	10.5	0.015	187	1.6	0.047	190	275	44.7
		188	0.5	4.009	180	4.8	0.032	188	0.5	0.046	189	273	44.4
		206	2.4	3.260	189	10.4	0.016	206	2.4	0.047	211	300	42.2
		183	1.6	3.588	172	7.5	0.031	183	1.6	0.047	186	273	46.8
		183	2.1	3.759	168	10.2	0.032	183	2.1	0.046	187	270	44.4
		188	0.0	2.868	168	10.6	0.034	188	0.0	0.058	188	279	48.4
average			1.0	3.522		9.1	0.027		1.1	0.045			44.1

Table 4: Values of the three lower bounds and upper bound at the root of the search tree (instances of small size with $n < 100$)

n	k	$LB_{greedy}(a)$			$LB_{k\ first}(a)$			$LB_{k\ best}(a)$			$w(T_0)$	$UB(a)$	
		value	%	time	value	%	time	value	%	time	in $G \setminus S^*$	value	%
100	5	186	0.0	3.635	177	4.8	0.031	186	0.0	0.031	186	230	23.7
		209	0.9	3.947	201	4.7	0.031	209	0.9	0.047	211	244	15.6
		193	0.5	3.760	184	5.2	0.031	193	0.5	0.047	194	230	18.6
		187	0.5	3.572	175	6.9	0.032	187	0.5	0.031	188	216	14.9
		205	0.0	3.697	195	4.9	0.031	205	0.0	0.031	205	263	28.3
		187	1.1	3.916	176	6.9	0.031	187	1.1	0.047	189	233	23.3
		211	0.5	3.572	201	5.2	0.032	209	1.4	0.046	212	249	17.5
		179	0.0	4.009	167	6.7	0.031	179	0.0	0.047	179	211	17.9
		201	1.0	4.040	187	7.9	0.031	199	2.0	0.031	203	239	17.7
		182	4.2	3.463	169	11.1	0.031	182	4.2	0.032	190	233	22.6
average													
100	7	185	0.0	5.912	173	6.5	0.032	184	0.5	0.062	185	253	36.8
		192	3.5	5.554	186	6.5	0.031	192	3.5	0.062	199	264	32.7
		215	0.0	5.850	192	10.7	0.031	212	1.4	0.047	215	274	27.4
		211	0.5	5.585	193	9.0	0.031	211	0.5	0.062	212	278	31.1
		201	0.0	5.651	186	7.5	0.035	201	0.0	0.056	201	265	31.8
		215	0.0	5.446	194	9.8	0.035	215	0.0	0.052	215	279	29.8
		220	1.3	5.028	202	9.4	0.034	220	1.3	0.052	223	279	25.1
		218	0.9	5.048	201	8.6	0.031	218	0.9	0.051	220	284	29.1
		202	1.0	5.772	192	5.9	0.031	202	1.0	0.047	204	276	35.3
		207	1.4	5.616	191	9.0	0.031	205	2.4	0.047	210	274	30.5
average													
200	5	266	0.0	11.965	254	6.5	0.062	266	0.5	0.156	266	277	36.8
		243	3.5	12.480	237	6.5	0.062	243	3.5	0.094	243	262	32.7
		241	0.0	12.699	238	10.7	0.171	241	1.4	0.078	245	262	27.4
		251	0.5	12.963	243	9.0	0.063	251	0.5	0.078	251	267	31.1
		243	0.0	13.306	233	7.5	0.063	243	0.0	0.234	244	260	31.8
		236	0.0	12.886	227	9.8	0.047	236	0.0	0.078	236	250	29.8
		245	1.3	12.075	237	9.4	0.046	245	1.3	0.094	245	260	25.1
		247	0.9	12.355	237	8.6	0.156	246	0.9	0.094	247	277	29.1
		241	1.0	11.559	233	5.9	0.063	241	1.0	0.093	241	268	35.3
		257	1.4	11.283	248	9.0	0.058	257	2.4	0.109	257	259	30.5
average													
300	5	316	0.6	27.565	311	2.2	0.078	316	0.6	0.125	318	328	3.1
		333	0.0	27.051	323	3.0	0.078	333	0.0	0.234	333	346	3.9
		325	0.3	29.796	318	2.5	0.156	325	0.3	0.125	326	338	3.7
		324	0.3	28.002	318	2.2	0.140	324	0.3	0.140	325	335	3.1
		334	0.0	30.077	326	2.4	0.078	334	0.0	0.187	334	346	3.6
		328	0.6	30.046	322	2.4	0.078	328	0.6	0.187	330	344	4.2
		324	0.3	28.252	320	1.5	0.234	324	0.3	0.125	325	337	3.7
		334	0.3	28.283	327	2.4	0.078	334	0.3	0.203	335	346	3.3
		330	0.0	25.303	322	2.4	0.094	330	0.0	0.328	330	341	3.3
		316	0.0	27.779	311	1.6	0.141	316	0.0	0.125	316	327	3.5
average													
400	5	418	0.0	45.256	413	1.2	0.218	418	0.0	0.172	418	426	1.9
		410	0.0	44.398	405	1.2	0.109	410	0.0	0.250	410	421	2.7
		412	0.0	45.022	406	1.5	0.187	412	0.0	0.250	412	419	1.7
		409	0.0	43.617	406	0.7	0.203	409	0.0	0.141	409	419	2.4
		411	0.0	47.861	406	1.2	0.125	411	0.0	0.156	411	422	2.7
		418	0.0	45.552	410	1.9	0.109	418	0.0	0.187	418	426	1.9
		409	0.0	44.429	406	0.7	0.140	409	0.0	0.219	409	424	3.7
		411	0.0	45.692	406	1.2	0.187	411	0.0	0.172	411	422	2.7
		415	0.0	44.647	412	0.7	0.141	415	0.0	0.249	415	424	2.2
		414	0.0	38.207	410	1.0	0.109	414	0.0	0.172	414	423	2.2
average													

Table 5: Values of the three lower bounds and upper bound at the root of the search tree (instances of large size with $n \geq 100$)

4.2 Upper bound

Let s be a given node of the search tree. To compute $UB(s)$, we select the edge in $T_1(s)$ of largest weight and we replace the edge deleted from $T_j(s)$ by the edge with largest weight belonging to $T_{j+1}(s)$, for $j = 1, \ldots, k - |mv(s)| - 1$. We repeat this process $k - |mv(s)| - 1$ times.

Let F be the set of the $k - |mv(s)|$ edges selected from $T_1(s)$ in this process. Then, we must determine the $k - |mv(s)|$ edges to be removed. To obtain an upper bound for all feasible solutions obtained from s, we delete the $k - |mv(s)|$ edges of smallest weight among the edges of $F \cup T_0(s) \backslash mst(s)$. Denote by E_{min} the subset of these selected edges removed. Therefore, $UB(s) = w(T_0(s)) + w(F) - w(E_{min})$.

We computed, for instances with different values of n and k, this upper bound at the root

a of the search tree (see Tables 4 and 5). Besides the bound value, we report the percent deviation from the optimal value defined as $\frac{UB-opt}{opt}$. The main observation is that $UB(a)$ is rather close to the optimal value for small values of k and deteriorates as k increases.

4.3 Branching strategy

Let a be the root of the search tree. The branching strategy is the same as for the explicit enumeration algorithm. We start with a feasible solution value corresponding to $\max\{LB_{greedy}(a), LB_{first}(a), LB_{best}(a)\}$. We tested two different best first search strategies. The first one is the standard strategy (Branching: best upper bound) where the node with the largest upper bound is selected first. No lower bound is computed and the fathoming test is performed only when we update the current best feasible solution value, which can occur only at level $k-1$ of the search tree. In the second strategy (Branching: best lower bound), the node with the largest lower bound is selected first. Lower and upper bounds are computed at every node. Since LB_{best} gives values close to the best ones and takes less time, we use this bound for computing a lower bound. Here, the fathoming test is performed at each node by comparing each lower bound value with the current best feasible solution value.

4.4 Illustrative example

Reconsider the graph G, illustrated in Figure 1, as input for 3 Most Vital Edges MST. We show how the strategy "Branching: best upper bound" proceeds.

We start, as in the explicit algorithm, by constructing the root a of the search tree, whose elements are:

- $mv(a) = mst(a) = \emptyset$
- $U(a)$ is the union of the following trees

  $T_0(a) : (1,2), (1,3), (3,5), (4,5)$

  $T_1(a) : (2,3), (3,4), (1,4), (2,5)$

  $T_2(a) : (2,4), (1,5)$
- and $w(T_0(a)) = 12$

Let $S = \{a\}$, $bestvalue = \max\{LB_{Greedy}(a), LB_{k\ first}(a), LB_{k\ best}(a)\} = \max\{22, 24, 22\} = 24$ and $bestset = \{(1,2), (1,3), (3,5)\}$.

Iteration 1: We select node a.

Since $|mv(a)| < k-1$, we test the four children of node a whose elements are given in Table 6.

Since $UB(4) \leq bestvalue$, we generate only nodes $1, 2$ and 3. Then, $S = \{1, 2, 3\}$.

Iteration 2: We select node 1, which has the current best upper bound.

Since $|mv(1)| < k-1$, we test the four children of node a whose elements are summarized in Table 7.

Since $UB(7)$ and $UB(8)$ are less than $bestvalue$, $S = \{5, 6, 2, 3\}$.

Iteration 3: We select node 5.

s		1	2	3	4
$mv(s)$		$\{(1,2)\}$	$\{(1,3)\}$	$\{(3,5)\}$	$\{(4,5)\}$
$mst(s)$		$\emptyset$	$\{(1,2)\}$	$\{(1,2),(1,3)\}$	$\{(1,2),(1,3),(3,5)\}$
$w(T_0(s))$		14	13	15	14
$U(s)$	T_0	$(1,3),(2,3),(3,5),(4,5)$	$(1,2),(2,3),(3,5),(4,5)$	$(1,2),(1,3),(4,5),(3,4)$	$(1,2),(1,3),(3,5),(3,4)$
	T_1	$(3,4),(1,4),(2,5),(2,4)$	$(3,4),(1,4),(2,5)$	$(1,4),(2,5)$	$(1,4)$
	T_2	$(1,5)$	$(2,4),(1,5)$	$(2,4),(1,5)$	$(2,4)$
$UB(s)$		32	28	25	18

Table 6: Children of node a

s		5	6	7	8
$mv(s)$		$\{(1,2),(1,3)\}$	$\{(1,2),(2,3)\}$	$\{(1,2),(3,5)\}$	$\{(1,2),(4,5)\}$
$mst(s)$		$\emptyset$	$\{(1,3)\}$	$\{(1,3),(2,3)\}$	$\{(1,3),(2,3),(3,5)\}$
$w(T_0(s))$		21	21	17	16
$\widetilde{U}(s)$	T_0	$(2,3),(3,5),(4,5),(1,4)$	$(1,3),(3,5),(4,5),(2,5)$	$(1,3),(2,3),(4,5),(3,4)$	$(1,3),(2,3),(3,5),(3,4)$
	T_1	$(3,4),(2,5),(2,4),(1,5)$	$(3,4),(2,4),(1,5)$	$(1,4),(2,5)$	$(1,4)$
$UB(s)$		30	29	22	18

Table 7: Children of node 1

As $|mv(5)| = k-1$, we compute the replacement edge for all edges in $T_0(5)$ and find that $w(T_0(5)) + max_{e_j \in T_0(5) \setminus mst(5)}(w(r(e_j)) - w(e_j)) = 28 > bestvalue$. Then, $bestvalue = 28$, $bestset = \{(1,2),(1,3),(2,3)\}$ and $S = \{6\}$.

Iteration 4: We select node 6.

As $|mv(6)| = k-1$, we compute the replacement edge for all edges in $T_0(5)$ and find that $w(T_0(5)) + max_{e_j \in T_0(5) \setminus mst(5)}(w(r(e_j)) - w(e_j)) = 24 < bestvalue$. We discard node 6 from S.

Since $S = \emptyset$, the algorithm terminates. Figure 3 gives the search tree generated during the algorithm. A solution for 3 Most Vital Edges MST is $bestset = \{(1,2),(1,3),(2,3)\}$ and the weight of MST in $G \setminus \{(1,2),(1,3),(2,3)\}$ is $bestvalue = 28$.

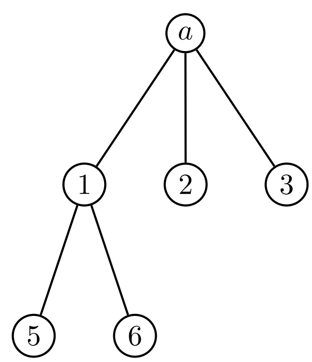

Figure 3: Search tree of implicite algorithm

5 A mixed integer programming formulation for finding the k most vital edges

Several linear programming formulations have been proposed to model the determination of a minimum spanning tree on a graph [12]. In one of these formulations, the minimum spanning tree is considered as a special version of a network design problem. Modeling the network by a connected graph, the problem consists of sending flow between all nodes of the graph. Thus, a variable x_e associated to an edge e indicates whether or not we install the edge e to be available to carry any flow. One model proposed by Magnanti and Wolsey is the *directed multicommodity flow model.* Let $D = (V, A)$ be the digraph formed by replacing each edge (i, j) in E by two arcs (i, j) and (j, i) in A. In this model, one of the nodes, say node 1, is considered as a root and each node $\ell \neq 1$ defines a commodity. Node 1 must send to each node $\ell \neq 1$ one unit of commodity ℓ. Denote by f^{ℓ}_{ij} the flow of commodity ℓ on the arc (i, j). For an edge $e = (i, j)$, we set $w_{ij} = w(e)$ in the following formulations. The linear program associated to this model is:

$$
\begin{cases}
\min\limits_{(i,j)\in E} \sum w_{ij}(y_{ij} + y_{ji}) & & \\
\text{s.t.} & & \\
\sum\limits_{(j,1)\in A} f^{\ell}_{j1} - \sum\limits_{(1,j)\in A} f^{\ell}_{1j} = -1 & \forall \ell \in V\backslash\{1\} & (1) \\
\sum\limits_{(j,i)\in A} f^{\ell}_{ji} - \sum\limits_{(i,j)\in A} f^{\ell}_{ij} = 0 & \forall i, \ell \in V\backslash\{1\},\ i \neq \ell & (2) \\
\sum\limits_{(j,\ell)\in A} f^{\ell}_{j\ell} - \sum\limits_{(\ell,j)\in A} f^{\ell}_{\ell j} = 1 & \forall \ell \in V\backslash\{1\} & (3) \\
f^{\ell}_{ij} \leq y_{ij} & \forall (i, j) \in A,\ \forall \ell \in V\backslash\{1\} & (4) \\
\sum\limits_{(i,j)\in A} y_{ij} = n - 1 & & (5) \\
f_{ij} \geq 0, \qquad y_{ij} \geq 0 & \forall (i, j) \in A &
\end{cases}
$$

In this model, constraints (1) - (3) correspond to flow balances at the nodes. Constraints (4) state that the flows on (i, j) for all commodities are zero if $y_{ij} = 0$. Thus, these four groups of constraints impose that the graph defined by a solution given by edges (i, j) such that $y_{ij} = 1$ is connected. Constraints (5) indicate that any solution must contain $n-1$ edges, thus any possible solution has to be a tree. Therefore, this formulation models the problem of finding a minimum spanning tree.

Remark that in this formulation, integrity constraints on variables y_{ij} are omitted. Indeed, Magnanti and Wolsey [12] show that the extreme points of the set of feasible solutions corresponding to this model are integers. The dual corresponding to this linear program is given by:

$$
\begin{cases}
\max \sum\limits_{\ell \in V,\, \ell \neq 1} (\alpha_\ell^\ell - \alpha_1^\ell) \ + (n-1)\mu \\
\text{s.t.} \\
\sigma_{ij}^\ell \geq \alpha_j^\ell - \alpha_i^\ell & \forall (i,j) \in A,\ \forall \ell \in V \backslash \{1\} \\
\sum\limits_{\ell \neq 1} \sigma_{ij}^\ell + \mu \leq w_{ij} & \forall (i,j) \in E \\
\sum\limits_{\ell \neq 1} \sigma_{ji}^\ell + \mu \leq w_{ij} & \forall (i,j) \in E \\
\sigma_{ij}^\ell \geq 0 & \forall (i,j) \in A,\ \forall \ell \in V \backslash \{1\} \\
\alpha_i^\ell \geq 0 & \forall i \in V, \ell \in V \backslash \{1\} \\
\mu \text{ unrestricted}
\end{cases}
$$

Using the previous MST formulation, one can model k Most Vital Edges MST defined on the graph $U_k^G = (V, E_u)$ with $E_u = \cup_{j=0}^k T_j$ as follows:

$$
\begin{cases}
\max\limits_{z \in Z} \ \ \min \sum\limits_{(i,j) \in E_u} (w_{ij} + M_{ij}\ z_{ij})(y_{ij} + y_{ji}) \\
\text{s.t.} \\
\qquad \sum\limits_{(j,1) \in A_u} f_{j1}^\ell - \sum\limits_{(1,j) \in A_u} f_{1j}^\ell = -1 & \forall \ell \in V \backslash \{1\} \\
\qquad \sum\limits_{(j,i) \in A_u} f_{ji}^\ell - \sum\limits_{(i,j) \in A_u} f_{ij}^\ell = 0 & \forall i, \ell \in V \backslash \{1\},\ i \neq \ell \\
\qquad \sum\limits_{(j,\ell) \in A_u} f_{j\ell}^\ell - \sum\limits_{(\ell,j) \in A_u} f_{\ell j}^\ell = 1 & \forall \ell \in V \backslash \{1\} \\
\qquad f_{ij}^\ell \leq y_{ij} & \forall (i,j) \in A_u,\ \forall \ell \in V \backslash \{1\} \\
\qquad \sum\limits_{(i,j) \in A_u} y_{ij} = n-1 \\
\qquad f_{ij} \geq 0, y_{ij} \geq 0 & \forall (i,j) \in A_u \\
\text{where } Z = \{z_{ij} \in \{0,1\},\ \forall (i,j) \in E_u : \sum\limits_{(i,j) \in E_u} z_{ij} = k\}
\end{cases}
$$

In this formulation, variable z_{ij} is equal to 1 if edge (i,j) is deleted and 0 otherwise. In order to discard this edge from any MST, we assign it the weight $w_{ij} + M_{ij}$ where M_{ij} is a large enough constant, e.g. $M_{ij} = \max_{(i,j) \in E} w_{ij} + 1 - w_{ij}$.

Using the dual of the inner program, we obtain the following mixed integer programming formulation for k Most Vital Edges MST.

$$
\begin{cases}
\max \sum\limits_{\ell \in V,\, \ell \neq 1} (\alpha_\ell^\ell - \alpha_1^\ell) \ + (n-1)\mu \\
\text{s.t.} \\
\sigma_{ij}^\ell \geq \alpha_j^\ell - \alpha_i^\ell & \forall (i,j) \in A_u,\ \forall \ell \in V \backslash \{1\} \\
\sum\limits_{\ell \neq 1} \sigma_{ij}^\ell + \mu \leq w_{ij} + M_{ij}\ z_{ij} & \forall (i,j) \in E_u \\
\sum\limits_{\ell \neq 1} \sigma_{ji}^\ell + \mu \leq w_{ij} + M_{ij}\ z_{ij} & \forall (i,j) \in E_u \\
\sum\limits_{(i,j) \in E_u} z_{ij} = k \\
z_{ij} \in \{0,1\} & \forall (i,j) \in E_u \\
\sigma_{ij}^\ell \geq 0 & \forall (i,j) \in A_u,\ \forall \ell \in V \backslash \{1\} \\
\alpha_i^\ell \geq 0 & \forall i \in V, \ell \in V \backslash \{1\} \\
\mu \text{ unrestricted}
\end{cases}
$$

6 Computational results

All experiments presented here were performed on a 3.4GHz computer with 3Gb RAM. All proposed algorithms are implemented in C. All instances are complete graphs defined on n vertices. Weights $w(e)$ for all $e \in E$ are generated randomly, uniformly distributed in $[1, 100]$. For each value of n and k presented in this study, 10 different instances were generated and tested. The results are reported in Table 8 where each given value is the average over 10 instances. For the implicit enumeration algorithm, *computed* and *generated* nodes represent respectively nodes for which we have determined mv, mst, U and UB and nodes for which $UB > bestvalue$ and must be stored. Column $\sharp$opt corresponds to the number of instances solved optimally.

We first compare the explicit and implicit enumeration algorithms. The results show that implicit enumeration algorithms are much faster than the explicit enumeration algorithm and can handle instances of considerably larger size. Observe that, for the explicit enumeration algorithm, the search tree size is identical for any instance of the same (n, k) type. As a consequence, either all or none of the instances of a same (n, k) type can be solved. Moreover, for the same reason, computation times show a low variance for all instances of a same (n, k) type. Regarding the implicit enumeration algorithm, the "Branching: best upper bound" strategy yields slightly better running times than the "Branching: best lower bound" strategy. However, the "Branching: best upper bound" strategy, for which fathoming tests are performed less frequently, generates more nodes. Thus, owing to the limited memory capacity, the "Branching: best lower bound" strategy can handle instances of larger size.

We compare now the results obtained by the mixed integer program with those of the implicit enumeration algorithm. For this, we implemented the mixed integer program using the solver CPLEX 12.1 and we run it on the same generated instances. We limited the running time to 1 hour for the instances with 20, 25, 30 and 50 vertices, and to 2 hours for the other instances. The results are also reported in Table 8 where

- *Time*, given in seconds, is the average running time on the 10 instances. For any instance which is not solved optimally within the time limit, the running time is set to this limit;
- *Generated nodes* represents the average number of nodes created in the search tree corresponding to instances returning feasible solutions;
- *Gap*, expressed as a percentage, represents the average over ratios $\frac{UB-BS}{UB}$ computed on all instances returning at least one feasible solution, where UB is the final best upper bound and BS is the best solution value found;
- *Opt/Feas* represents the number of instances solved optimally /for which at least one feasible solution was found within the time limit.

We note that the mixed integer program reaches the optimal value for very small instances only. Actually, for $n < 100$, we only obtain in most cases feasible solutions with rather large gaps which indicates that optimality is far from being reached. Finally, for instances with $n \geq 100$, no feasible solutions are returned within the time limit. Moreover, for $n = 300$ and 400, the execution of the program exceeds the memory capacity after a few seconds (297.437 and 0.56 seconds in average respectively).

From all these remarks, we can conclude that our proposed implicit enumeration algorithm gives better results than the explicit enumeration algorithm as well as the resolution of the mixed integer program and this both in terms of running time and memory use.

We propose in the following an ε-approximate algorithm based on our implicit algorithm. The aim being to obtain an ε-approximate solution of the optimum, the condition to generate a node s in the search tree is now $(1-\varepsilon)UB(s) > bestvalue$. Indeed, the value v returned by the approximate algorithm must verify $opt(G)(1-\varepsilon) \leq v \leq opt(G)$. Since v is equal to $bestvalue$, any node for which $UB(s)(1-\varepsilon) \leq bestvalue$ is fathomed.

The algorithm is tested on the same instances generated before and this for $\varepsilon = 0.01, 0.05$, and 0.1. Thus, we compare the ε-approximate algorithm with the implicit algorithm. The results are summarized in Table 9. The meaning of computed and generated nodes is the same as above and each given value in the table represents the average over the 10 generated instances for each value of n and k.

n	k	Explicit enumeration		Implicit enumeration							Mixed Integer Program			
				Branching: best lower bound			Branching: best upper bound							
		Time (s)	Nodes	Time (s)	Nodes		Time (s)	Nodes		♯opt	Time (s)	Generated nodes	Gap (%)	Opt/Feas
					Computed	Generated		Computed	Generated					
20	3	**0.000**	210	**0.000**	165.1	33.5	0.001	165.1	34.3	10	35.750	1 638.2	0	10 / 10
	5	0.135	8 855	**0.032**	3 280.6	422.3	**0.032**	3 230.9	463.2	10	692.984	21 792.4	0	10 / 10
	7	2.732	177 100	0.419	35 714.0	4 792.0	**0.380**	35 659.2	5 918.2	10	3 600.000	61 386.5	23.91	0 / 10
	9	36.020	220 075	3.322	258 321.8	35 639.1	**3.047**	257 776.0	44 037.4	10	3 600.000	36 908.1	46.49	0 / 10
25	3	**0.000**	325	**0.000**	245.0	29.8	0.003	245.0	31.4	10	141.270	2 066.5	0	10 / 10
	5	0.318	20 475	0.095	7 146.4	705.1	**0.089**	7 047.2	866.7	10	2 984.021	29 300.4	8.69	5 / 10
	7	8.783	593 775	1.772	128 802.5	15 143.4	**1.617**	128 742.2	16 926.0	10	3 600.000	14 218.1	46.05	0 / 10
	8	52.068	2 629 575	3.765	247 900.6	26 076.8	**3.566**	247 822.8	31 938.3	10	3 600.000	10 733.5	66.43	0 / 10
30	3	0.007	465	**0.000**	345.1	47.7	0.005	345.1	49.7	10	424.171	3 831.9	0	10 / 10
	5	0.812	40 920	0.260	16 756.3	1 373.7	**0.231**	16 625.9	1 588.7	10	3 458.330	13 156.2	26.03	1 / 10
	7	40.461	1 623 160	3.899	231 523.5	20 779.0	**3.553**	231 210.2	25 737.4	10	3 600.000	4 855.9	63.65	0 / 10
50	3	0.880	1 275	0.028	949.1	64.9	**0.026**	949.1	85.3	10	3 600.000	1 285.8	17.28	0 / 10
	5	15.390	292 825	2.043	76 840.3	4 649.3	**1.856**	74 550.2	5 138.1	10	3 600.000	503.0	43.59	0 / 10
	7	-	-	88.886	3 156 471.8	168 127.4	**81.707**	3 156 170.1	218 830.4	10	3 600.000	21.33	80.47	0 / 9
75	3	0.376	2 850	0.101	2 296.8	114.8	**0.096**	2 296.8	117.7	10	7 200.000	430.2	17.83	0 / 10
	5	-	-	11.248	259 738.0	8 130.7	**10.459**	259 737.6	10 519.6	10	6 490.238	0.3	39.22	1 / 10
	7	-	-	**650.008**	13 330 591.9	474 912.7	*463.385*	*9 608 531.7*	*379 179.2*	7	7 200.000	0	55.75	0 / 3
100	3	1.083	5 050	0.224	3 617.1	83.3	**0.210**	3 617.1	89.9	10	7 200.000	0		0 / 0
	5	-	-	54.148	904 662.4	19 383.8	**49.895**	904 662.4	23 800.1	10	7 200.000	0		0 / 0
	7	-	-	**2 016.410**	26 835 600.6	721 120.4	*935.777*	*11 986 049.2*	*368 180.0*	4	7 200.000	0		0 / 0
200	5	-	-	**572.557**	2 933 547.2	46 236.3	670.340	2 933 296.1	49 073.6	10	7 200.000	0		0 / 0
300	5	-	-	**1 793.460**	3 996 192.1	43 671.2	2 163.350	3 980 311.0	56 924.5	10	7 200.000	0		0 / 0
400	5	-	-	**7 265.850**	10 956 321.8	106 433.4	*6 195.182*	*5 927 376.8*	*56 424.5*	7	-	-	-	0 / 0

italics: average over instances solved optimally
-: memory overflow

Table 8: Comparison of explicit enumeration, implicit enumeration and MIP-based algorithms

n	k	ε-approximate algorithm											
		$\varepsilon = 0.01$				$\varepsilon = 0.05$				$\varepsilon = 0.1$			
		Time (s)	Nodes		ε'	Time (s)	Nodes		ε'	Time (s)	Nodes		ε'
			Computed	Generated			Computed	Generated			Computed	Generated	
20	3	0.000	162.9	30.6	0.00000	0.000	136.9	14.0	0.00000	0.000	100.6	7.4	0.00198
	5	0.035	3 108.2	384.6	0.00000	0.024	2 068.8	211.1	0.00043	0.012	1 258.4	113.1	0.00267
	7	0.393	33 258.5	4 356.0	0.00000	0.273	21 820.0	2 575.7	0.00323	0.174	13 209.9	1 451.0	0.00922
	9	3.044	237 267.0	32 085.4	0.00000	2.180	160 036.0	20 093.2	0.00421	1.376	93 275.6	10 888.2	0.00735
25	3	0.000	230.8	26.7	0.00000	0.000	189.8	13.1	0.00263	0.000	98.2	5.7	0.00263
	5	0.093	6 691.6	637.2	0.00060	0.061	4 235.5	345.2	0.00213	0.031	2 002.0	146.1	0.00779
	7	1.648	119 033.8	13 603.0	0.00000	1.066	72 193.8	7 178.9	0.00148	0.606	37 683.2	3 379.8	0.00416
	8	3.513	226 536.1	23 389.9	0.00000	2.255	135 623.2	12 792.9	0.00142	1.242	68 426.4	5 900.1	0.00319
30	3	0.000	338.1	38.8	0.00000	0.000	280.1	17.3	0.00453	0.000	161.6	7.2	0.00452
	5	0.233	15 137.4	1 171.7	0.00000	0.123	7 302.6	470.5	0.00307	0.059	3 146.2	181.9	0.00363
	7	3.523	209 289.3	18 256.8	0.00000	2.183	119 797.9	9 363.5	0.00470	1.155	57 665.1	4 062.5	0.00721
50	3	0.025	899.4	48.8	0.00000	0.011	381.6	14.1	0.00000	0.000	76.5	2.4	0.00646
	5	1.790	67 052.0	3 757.3	0.00000	0.635	20 586.6	866.5	0.00178	0.255	7 213.3	241.8	0.00279
	7	74.688	2 534 780.6	130 685.3	0.00000	28.324	820 954.5	36 722.1	0.00053	7.958	193 201.5	7 827.2	0.00316
75	3	0.092	2 121.1	75.7	0.00000	0.0016	325.4	5.3	0.00241	0.003	75.0	1.0	0.00355
	5	8.334	187 230.6	5 444.6	0.00000	1.679	27 753.6	616.2	0.00168	0.232	2 860.8	50.4	0.00387
	7	510.768	9 838 080.8	336 993.8	0.00000	109.664	1 734 007.8	51 514.5	0.00195	20.661	260 410.7	6 584.0	0.00536
100	3	0.208	3 341.4	57.4	0.00000	0.013	121.6	1.4	0.00051	0.010	100.0	1.0	0.00308
	5	34.779	561 343.8	10 619.5	0.00000	3.875	41 860.1	611.1	0.00143	0.396	3 307.4	41.6	0.00297
	7	1 214.43	14 901 505.8	377 861.2	0.00000	179.771	1 703 572.1	34 196.8	0.00143	13.940	95 045.1	1 492.2	0.00371
200	5	165.904	682 703.2	10 147.9	0.00000	0.731	1 693.0	11.5	0.00163	0.131	200.0	1.0	0.00163
300	5	87.600	164 368.6	1 129.4	0.00030	0.380	300.0	1.0	0.00245	0.379	300.0	1.0	0.00241
400	5	89.564	80 786.1	257.3	0.00000	0.846	400.0	1.0	0.00000	0.842	400.0	1.0	0.00000

Table 9: Results of the ε-approximate algorithm

We note that the running times of the ε-approximate algorithm are significantly lower than those of the implicit enumeration algorithm. Running times do not exceed 21 seconds for $\varepsilon = 0.1$, 180 seconds for $\varepsilon = 0.05$ and 1 215 seconds for $\varepsilon = 0.01$. We also note that for large instances with $n = 300$ and 400 nodes, the ε-approximate algorithm solves the problem for $\varepsilon = 0.05$ and 0.1 at the root in a time less than 1 second, and for $\varepsilon = 0.1$ in a time less than 90 seconds while the implicit enumeration algorithm requires 1 793.460 and 7 265.850 seconds respectively.

Moreover, the approximate solutions a posteriori are within ε' to the optimum, with $\varepsilon' \leq 0.0006$ for $\varepsilon = 0.01$, $\varepsilon' \leq 0.0047$ for $\varepsilon = 0.05$ and $\varepsilon' \leq 0.00922$ for $\varepsilon = 0.1$.

For $\varepsilon = 0.01$, we note that the problem is nearly solved to optimality ($\varepsilon' = 0$).

All these remarks show that the proposed lower bounds and upper bound are of very good quality and that the running time of the implicit enumeration algorithm is the time needed to verify the optimality of the solution. Indeed, this optimal solution is either found in a few seconds or determined at the root of the search tree corresponding then to the maximum value of the three lower bounds associated to the root.

7 Conclusions

In this paper we presented and compared different algorithms for solving k Most Vital Edges MST. We first proposed an explicit enumeration algorithm that gives the best time complexity for general k. Using upper and lower bounds, we adapted the previous algorithm into an efficient implicit enumeration algorithm. We also proposed a mixed integer programming formulation of k Most Vital Edges MST which was solved using CPLEX. Our experiments showed a large superiority of the implicit enumeration algorithm. An ε-approximate version of this algorithm substantially improves running times while providing very good quality results (with an a posteriori approximation ratio usually much less than one tenth of the guaranteed ratio ε).

All the previous algorithms can be easily adapted to solve some variants of the k Most Vital Edges MST problem. In a first variant, a removing cost is associated to each edge. The problem consists of finding a subset of edges with total cost bounded by a budget limit whose deletion causes the largest increase in the weight of a minimum spanning tree. In a second variant, we have to determine a minimum number of edges to be removed such that the weight of a minimum spanning tree in the resulting graph is at least a fixed value.

References

[1] A. Bar-Noy, S. Khuller, and B. Schieber. The complexity of finding most vital arcs and nodes. Technical Report CS-TR-3539, University of Maryland, 1995.

[2] C. Bazgan, S. Toubaline, and D. Vanderpooten. Complexity of determining the most vital elements for the 1-median and 1-center location problems. In *Proceeding of the 4^{th} Annual International Conference on Combinatorial Optimization and Applications (COCOA 2010)*, LNCS 6508, Part I, pages 237–251, 2010.

[3] C. Bazgan, S. Toubaline, and D. Vanderpooten. Critical edges/nodes for the minimum spanning tree problem: complexity and approximation. to appear in *Journal of Combinatorial Optimization.*

[4] B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000.

[5] B. Dixon, M. Rauch, and R.E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992.

[6] G. N. Frederickson and R. Solis-Oba. Increasing the weight of minimum spanning trees. *Proceedings of the* 7^{th} *ACM-SIAM Symposium on Discrete Algorithms (SODA 1996)*, pages 539–546, 1996. Also appeared in *Journal of Algorithms*, 33(2): 244-266, 1999.

[7] L. Hsu, R. Jan, Y. Lee, C. Hung, and M. Chern. Finding the most vital edge with respect to minimum spanning tree in a weighted graph. *Information Processing Letters*, 39(5):277–281, 1991.

[8] K. Iwano and N. Katoh. Efficient algorithms for finding the most vital edge of a minimum spanning tree. *Information Processing Letters*, 48(5):211–213, 1993.

[9] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, V. Gurvich, G. Rudolf, and J. Zhao. On short paths interdiction problems : total and node-wise limited interdiction. *Theory of Computing Systems*, 43(2):204–233, 2008.

[10] W. Liang. Finding the k most vital edges with respect to minimum spanning trees for fixed k. *Discrete Applied Mathematics*, 113(2-3):319–327, 2001.

[11] W. Liang and X. Shen. Finding the k most vital edges in the minimum spanning tree problem. *Parallel Computing*, 23(13):1889–1907, 1997.

[12] T. L. Magnanti and L. Wolsey. Optimal trees. In *M. O. Ball, et al. (Eds.), Network Models, Handbook in Operations Research and Management Science, Vol 7, North-Holland, Amsterdam*, pages 503–615, 1995.

[13] E. Nardelli, G. Proietti, and P. Widmayer. A faster computation of the most vital edge of a shortest path. *Information Processing Letters*, 79(2):81–85, 2001.

[14] S. Pettie. Sensitivity analysis of minimum spanning tree in sub-inverse-ackermann time. In *Proceedings of* 16^{th} *International Symposium on Algorithms and Computation (ISAAC 2005)*, LNCS 3827, pages 964–973, 2005.

[15] S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM*, 49(1):16–34, 2002.

[16] H. D. Ratliff, G. T. Sicilia, and S. H. Lubore. Finding the n most vital links in flow networks. *Management Science*, 21(5):531–539, 1975.

[17] H. Shen. Finding the k most vital edges with respect to minimum spanning tree. *Acta Informatica*, 36(5):405–424, 1999.

[18] F. Suraweera, P. Maheshwari, and P. Bhattacharya. Optimal algorithms to find the most vital edge of a minimum spanning tree. Technical Report CIT-95-21, School of Computing and Information Technology, Griffith University, 1995.

[19] R. E. Tarjan. Applications of path compression on balanced trees. *Journal of the ACM*, 26(4):690–715, 1979.

[20] R. Wollmer. Removing arcs from a network. *Operations Research*, 12(6):934–940, 1964.

[21] R. K. Wood. Deterministic network interdiction. *Mathematical and Computer Modeling*, 17(2):1–18, 1993.