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

Cristina Bazgan, Sónia Toubaline, Daniel Vanderpooten

Efficient algorithms for finding the k most vital edges for the minimum spanning tree problem

Cristina Bazgan, Sonia Toubaline, and Daniel Vanderpooten

Université Paris-Dauphine, LAMSADE,
Place du Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, 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 exploration 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 by a weighted graph, identifying critical infrastructures amounts to finding a subset of edges whose removal from the graph causes the largest increase in the cost. 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 studied for various problems including shortest path [1,7,11] and maximum flow [18,14,19]. For the minimum spanning tree problem defined on a graph G with n vertices and m edges, Frederickson *et al.* [4] showed that, for general k, k Most Vital Edges MST is *NP*-hard and proposed an $O(\log k)$-approximation algorithm. For a fixed k the problem is obviously polynomial. The case $k = 1$ has been largely studied in the literature [5,6,16]. Hsu *et al.* [5] gave two algorithms in $O(m \log m)$ and $O(n^2)$. Iwano and Katoh [6] proposed an algorithm in $O(m\alpha(m, n))$ using Tarjan's result [17], where α is the inverse-Ackermann function. Pettie [12] improved

the results of Tarjan[17] and Dixon *et al.* [3], and therefore the current best deterministic algorithm for solving the case $k = 1$ is in $O(m \log \alpha(m, n))$. Several exact algorithms based on an explicit enumeration of possible solutions have been proposed [8,9,15]. The best one [8] 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 [12], 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 [8] using Pettie's result [12]. 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 formulation by a mixed integer program 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 and its use of memory space can handle instances of significantly larger size. Moreover, we 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. In section 7, we present an ϵ-approximate algorithm and compare it with the exact one. Conclusions are provided in section 8.

2 Basic concepts and preliminary results

Let $G = (V, E)$ be a weighted undirected connected graph with $|V| = n$, $|E| = m$ and $w(e) \geq 0$ 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 [9]) 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 $T \setminus \{e\}$. 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 $O(m \log \alpha(m,n))$ [12]. 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 a 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 [8] 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 tree search of depth $k-1$ in a breadth-first mode. At the i^{th} level of this tree search, $i = 0, \ldots, k-1$, 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, $\widetilde{U}(s) = (V, \cup_{i=0}^{k-|mv(s)|} T_i(s))$ 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 tree search, 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 tree search at the i^{th} level. We describe in the following the exact 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, $T_j(d) = T_j(s) \backslash \{e_{rep}\}$ 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.*

Algorithm 1: Explicit resolution of k MVE MST

```
   /* Let a be the root of the tree search                                   */
 1 Construct U_k^G;
 2 mv(a) ← ∅; mst(a) ← ∅; w(T_0(a)) ← w(T_0); Ũ(a) ← U_k^G;
 3 N_0 ← {a}; N_i ← ∅, i = 1, ..., k − 1;
 4 for i ← 0 to k − 2 do
 5     forall s ∈ N_i do
 6         forall e ∈ T_0(s) do
 7             find r(e) in T_1(s);
           /* T_0(s)\mst(s) = {e_1, ..., e_{n−1−|mst(s)|}}                   */
 8         forall e_j ∈ T_0(s)\mst(s) do
               /* create a new node d, a child of s                         */
 9             mv(d) ← mv(s) ∪ {e_j};
10             w(T_0(d)) ← w(T_0(s)) − w(e_j) + w(r(e_j));
11             mst(d) ← mst(s) ∪ (∪_{ℓ=1}^{j−1} {e_ℓ});
12             determine Ũ(d);
13             N_{i+1} ← N_{i+1} ∪ {d};
14 max ← 0;
15 forall s ∈ N_{k−1} do
16     forall e ∈ T_0(s) do
17         find r(e) in T_1(s);
18     forall e ∈ T_0(s)\mst(s) do
19         if w(T_0(s)) − w(e) + w(r(e)) > max then
20             max ← w(T_0(s)) − w(e) + w(r(e));
21             e* ← e;
22             s* ← s;
   /* The largest weight of a MST in the partial obtained graph is
      w(T_0(s*)) − w(e*) + w(r(e*))                                          */
23 return S* = mv(s*) ∪ {e*};
```

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 tree search such that $mv(r) \subseteq S'$ and for any child d of r, $mv(d) \not\subseteq S'$. 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 $w' \leq w^*$.

We compute now the complexity of Algorithm 1. The construction of U_k^G requires $O(km\alpha(m,n))$ using k times the best current algorithms for MST [2,13]. 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 tree search (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 $O(m \log \alpha(m,n))$ [12], $t_{edge-rep}$ is in $O(n \log \alpha(2(n-1),n))$ time. 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 tree search is dominated by the total time to find, for all nodes s of the tree search, the replacement edges $r(e)$ in $T_1(s)$ for all edges $e \in T_0(s)$. □

Remark 1. For each node s of the tree search, 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 tree search, 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 $\tilde{U}$, we would get the same complexity since the time for generating all nodes of the tree search is dominated by the total time for finding the replacement edges for all nodes in the tree search.

Discussion For fixed k, by using the result of Dixon *et al.* [3], Liang [8] 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 [12] 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, the advantage of our algorithm is to 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 as in $O(t_u + k^3 n^k + \sum_{i=1}^{k-1} \binom{n-1}{i} t_{k-i} + kn^k \log \alpha((k+1)(n-1), n))$, 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 [9,15] have a worse complexity than our algorithm both for fixed ad general k.

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.

In order to obtain the best possible bounds, we construct $U(s)$ for each node s, instead of using $\tilde{U}(s)$. 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 tree search, $k - |mv(s)|$ 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 realizes $max_{e_j \in T_0(s) \setminus mst(s)}(w(r(e_j)) - w(e_j))$ and replace it by $r(e_j^*)$. We update $U(s)$ and repeat the process until we remove $k - |mv(s)|$ edges. 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 weight of the MST obtained is the value of this bound.
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 tree search. The instances are generated as explained in section 6. Due to space limitation, we give in Table 1, results for two types of instances. 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	$LB_{greedy}(a)$		$LB_{first}(a)$		$LB_{best}(a)$		$w(T_0)$	$UB(a)$
		value	time(s)	value	time(s)	value	time(s)	in $G \setminus S^*$	
20	9	265	0.873	255	0.016	250	0.047	282	719
		221	0.889	219	0.015	222	0.032	229	711
		178	0.982	179	0.032	180	0.031	211	669
		166	0.842	157	0.000	157	0.016	186	681
		276	0.624	268	0.015	267	0.016	278	726
		246	0.904	243	0.016	240	0.000	279	764
		236	0.764	232	0.031	235	0.047	239	682
		272	0.967	254	0.031	255	0.031	272	712
		205	1.060	193	0.016	203	0.000	207	668
		245	0.748	216	0.000	225	0.016	249	716
100	7	185	5.912	173	0.032	184	0.062	185	253
		192	5.554	186	0.031	192	0.062	199	264
		215	5.850	192	0.031	212	0.047	215	274
		211	5.585	193	0.031	211	0.062	212	278
		201	5.651	186	0.035	201	0.056	201	265
		215	5.446	194	0.035	215	0.052	215	279
		220	5.028	202	0.034	220	0.052	223	279
		218	5.048	201	0.031	218	0.051	220	284
		202	5.772	192	0.031	202	0.047	204	276
		207	5.616	191	0.031	205	0.047	210	274

Table 1. Values of the lower and upper bounds at the root of the tree search

4.2 Upper bound

Let s be a given node of the tree search. 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 remove. 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 $F \cup T_0(s) \setminus mst(s)$. 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 tree search (see Table 1). 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 tree search. 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 tree search. 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.

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

Consider the graph $U_k^G = (V, E_u)$ with $E_u = \cup_{j=0}^{k} T_j$. Let $D = (V, A_u)$ be the digraph obtained by replacing each edge (i,j) in E_u by two arcs (i,j) and (j,i) in A_u and let $w_{ij} = w(e)$ for each edge $e \in E_u$. In [10], Magnanti and Wolsey present a formulation of the minimum spanning tree problem, called the directed multicommodity flow model. Using this model, we propose the following formulation for k Most Vital Edges MST:

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

In this formulation, we consider node 1 as the root of a MST and every node $\ell \neq 1$ defines a commodity. Denote by f^{ℓ}_{ij} the flow of ℓ passing through (i,j). 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.

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

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 2 where each given value is the average over 10 instances. For the implicit enumeration algorithm, treated and generated nodes represent respectively nodes for which we have computed mv, mst, and U and nodes satisfying the condition of not fathoming ($UB > bestvalue$). Column ♯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 tree search 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 2 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 tree search corresponding to instances giving a 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 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 using memory capacity.

7 ε-approximate algorithm

The proposed algorithm is based on the previous implicit algorithm. The aim being to obtain an ε-approximate solution of the optimum, the condition to generate a node s in the tree search 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 implemented in C and tested on the same instances generated in Section 6 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 3. The meaning of treated and generated nodes is the same as in Section 6 and each given value in the table represents the average over the 10 generated instances for each value of n and k.

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.

n	k	Explicit enumeration		Implicit enumeration								Mixed Integer Program			
				Branching: best lower bound			Branching: best upper bound								
		Time (s)	Nodes	Time (s)	Nodes Treated	Nodes Generated	Time (s)	Nodes Treated	Nodes Generated	♯opt		Time (s)	Generated nodes	Gap (%)	Opt/Feas
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 2. 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	Nodes		ε'	Time	Nodes		ε'	Time	Nodes		ε'
		(s)	Treated	Generated		(s)	Treated	Generated		(s)	Treated	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 3. Results of the ε-approximate algorithm

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 tree search corresponding then to the maximum value of the three lower bounds associated to the root.

8 Conclusions

Algorithms proposed in this paper 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. B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. W. Liang and X. Shen. Finding the k most vital edges in the minimum spanning tree problem. *Parallel Computer*, 23(3):1889–1907, 1997.

10. 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.
11. 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.
12. 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.
13. S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM*, 49(1):16–34, 2002.
14. 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.
15. H. Shen. Finding the k most vital edges with respect to minimum spanning tree. *Acta Informatica*, 36(5):405–424, 1999.
16. 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.
17. R. E. Tarjan. Applications of path compression on balanced trees. *Journal of the ACM*, 26(4):690–715, 1979.
18. R. Wollmer. Removing arcs from a network. *Operations Research*, 12(6):934–940, 1964.
19. R. K. Wood. Deterministic network interdiction. *Mathematical and Computer Modeling*, 17(2):1–18, 1993.