On the enumeration of non-dominated spanning trees with imprecise weights

Tom Davot, Sébastien Destercke, David Savourey

On the enumeration of non-dominated spanning trees with imprecise weights

Tom Davot, Sébastien Destercke, David Savourey

Université de Technologie de Compiègne, CNRS, Heudiasyc (Heuristics and Diagnosis of Complex Systems), CS 60319 - 60203 Compiègne Cedex,
surname.name@hds.utc.fr

Abstract. Many works within robust combinatorial optimisation consider interval-valued costs or constraints. While most of these works focus on finding unique solutions such as minimax ones, a few consider the problem of characterising a set of non-dominated optimal solutions. This paper is situated within this line of work, and consider the problem of exactly enumerating the set of non-dominated spanning trees under interval-valued costs. We show in particular that each tree in this set can be obtained through a polynomial procedure, and provide an efficient algorithm to achieve the enumeration.

1 Introduction

Combinatorial optimisation problems under interval-valued costs have attracted some attention in the past (one can check, for instance, the book [6] for a good reference on the topic). While the greatest majority of works in this setting look for robust unique solutions to this problem, some of them look at the problem of enumerating, or at least characterising sets of possible solutions.

In this paper, we are interested in the specific yet practically important case of minimum spanning trees, the problem or its generalisations being routinely used in many applications [10].

Given its importance as a basic combinatorial optimisation problem, it is not a surprise that many authors have considered interval-valued edges in the minimum spanning tree problem. A number of works have focused on finding a robust solution to the problem, such as Yaman et al. [13] that provides a mixed integer programming (MIP) to compute a minimax solution, or [1,9,5,2] that consider other notions of robust yet unique solution of the problem.

In this paper, our interest is not in providing one unique robust solution, but rather to consider the set of all non-dominated solutions, and to enumerate efficiently such solutions. Such a problem may be important if, e.g., one wants to browse the Pareto front of optimal solutions. Note that we are not the first one to explore such a problem, as for example [13] investigate the concept of weak (possible) and strong (necessary) edges, that is, edges that belong to at least one non-dominated solution and to every non-dominated solution, respectively. In [7], the authors defined a relation order on the set of feasible solutions and generated a Pareto set using bi-objective optimisation, yet this relation order is different from the one we consider here, and will in general not include all non-dominated solutions.

Our paper is structured as follows[1]: next section presents some notation and introduces the problem. In Section 3, we develop some structural preliminary results. Our main result is described in Section 4: we develop an algorithm that enumerates every non-dominated spanning tree. Finally, Section 5 is devoted to the presentation of some numerical experiments.

2 Notations and problem description

We present here the main notations used in the paper for graphs and set up our problem. The most important notions are illustrated in Figures 1 and 2.

2.1 Graph

Spanning tree. Let G be an undirected graph. We denote $V(G)$ the set of vertices of G and $E(G)$ the set of edges. A subgraph H of G is a graph such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. In the following, we let n and m denote the number of vertices and edges in a graph, respectively. We denote $G - H$ the subgraph of G for which we delete every vertex of H in G, that is, $V(G-H) = V(G) \setminus V(H)$ and $E(G-H) = \{uv \mid uv \in E(G) \wedge uv \cap V(H) = \varnothing\}$. Let X be a set of edges of G, we denote $G - X$ the subgraph of G obtained by deleting every edge of X in G, that is $V(G-X) = V(G)$ and $E(G-X) = E(G) \setminus X$. A *path* between two vertices u and v is a sequence of distinct vertices $(x = v_1, \ldots, v_k = v)$ such that there is an edge between v_i and v_{i+1} for each $1 \leq i < k$. A *cycle* is a path $(v_1, \ldots, v_k)$ for which there is also an edge between v_1 and v_k. A graph is *connected* if there is a path between each pair of vertices. A *connected component* H of G is a maximal connected subgraph of G, that is there is no vertex $v \in V(G) \setminus V(H)$ such that there is a path between v and a vertex $u \in V(H)$. Notice that G is connected if and only if G contains exactly one connected component. A *tree* is a connected graph without cycle. A *spanning tree* T of G is tree such that $V(T) = V(G)$ and $E(T) \subseteq E(G)$. We denote $ST(G)$ the set of spanning trees of G.

Cut. A *cut* $P = (V_1, V_2)$ of a graph G is a partition of its vertices into two disjoint subsets V_1 and V_2, *i.e.* $V(G) = V_1 \cup V_2$ and $V_1 \cap V_2 = \varnothing$. To each cut $P = (V_1, V_2)$, we associate a set of edges $X = \{uv \in E(G) \mid u \in V_1, v \in V_2\}$ called *cut-set* of P (or simply cut-set if P is not known). Notice that the deletion of X in G disconnects the graph, that is, $G - X$ contains at least one more connected component than G. The cut-set X is *minimal* if there is no $X' \subset X$ such that X' is also a cut-set. If G is connected, X is minimal if and only if $G - V_1$ and $G - V_2$ are connected. Let T be a spanning tree of G, notice that $E(T) \cap X \neq \varnothing$ since otherwise, T would not be connected. Let X be a cut-set and let uv be an edge of X that does not belong to $E(T)$. Let p be the path between u and v

[1] We have provided proofs in the appendix for review purposes, as including them would exceed page limits. Appendices will not be part of the final version.

in T and let e be an edge in $X \cap E(p)$. Notice that e exists since otherwise X would not be a cut-set. We say that e is X-*blocking* for uv in T. Note that it is possible to construct another spanning tree T' by adding uv and removing e in T, that is $E(T') = E(T) \cup \{uv\} \setminus \{e\}$. In the following, we call such operation *swapping* uv *and* e *in* T. It is possible to define a cut-set with a spanning tree and an edge as follows.

Definition 1 (Figure 1). *Let G be a graph, let T be a spanning tree of G and let e be an edge of T. Let H_1 and H_2 be the two connected components of $T - e$. We say that the cut-set of the cut $(V(H_1), V(H_2))$ is the* cut-set induced by T and e.

Note that a cut-set can be induced by different spanning trees and different edges, as depicted by Figure 1. Also, a cut-set X is induced by T and e if and only if $E(T) \cap X = e$. Moreover, a cut-set induced by a spanning tree and an edge is always minimal.

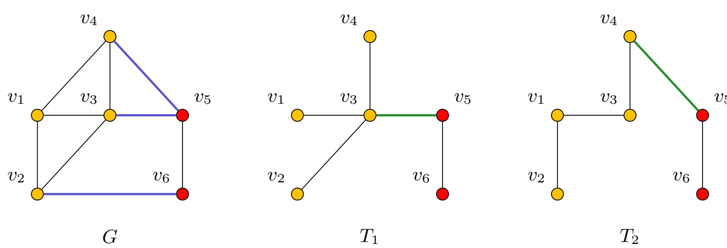

Fig. 1. Left. Example of a cut $P = (V_1 = \{v_1, \dots, v_4\}, V_2 = \{v_5, v_6\})$ for a graph G. The vertices of V_1 and V_2 are depicted in yellow and red, respectively. The edges that belong to the cut-set X of P are depicted in blue. **Center and Right.** The cut-set X is induced by the spanning tree T_1 (resp. T_2) and the edge v_3v_5 (resp. v_4v_5), depicted in green. The edge v_3v_5 is X-blocking for v_2v_6 and v_4v_5 in T_1. The edge v_4v_5 is X-blocking for v_2v_6 and v_3v_5 in T_2.

2.2 Imprecise weights and problem description

An *imprecise weight* $[\underline{\omega}, \overline{\omega}]$ is an interval of numbers. An *imprecise weighted graph* (G, Ω) is a graph with a function Ω that associates with each edge e an imprecise weight $[\underline{\omega}_e, \overline{\omega}_e]$. A *realization* $R : E(G) \mapsto \mathbb{R}$ of Ω is a function that associates with each edge e a weight $w \in [\underline{\omega}_e, \overline{\omega}_e]$. We denote $\mathcal{R}_\Omega$ the set of realizations of Ω.

Let H be a subgraph of G. Given a weight realization R, the weight of H, denoted $R(H)$ is the sum of the weights of its edges, that is, $R(H) = \Sigma_{e \in E(H)} R(e)$. Given two subgraphs H_1 and H_2, we say that H_1 *dominates* H_2, denoted by $H_1 \succ H_2$ if,

$$\forall R \in \mathcal{R}_\Omega, R(H_1) < R(H_2).$$

Given two edges e_1 and e_2, we say that e_1 *dominates* e_2 if $\overline{\omega}_{e_1} < \underline{\omega}_{e_2}$. In the following, we are interested in the set of non-dominated spanning trees

$$\mathcal{T}(G, \Omega) := \{T \in ST(G) \mid \nexists T' \in ST(G), T' \succ T\}. \tag{1}$$

In this article, we address the problem of enumerating every spanning tree of $\mathcal{T}(G, \Omega)$. We recall that computing a minimum spanning tree T for some realization R (*i.e.* such that $R(T)$ is minimum) can be done in polynomial time using a greedy algorithm. For example, Kruskal's algorithm computes a minimum spanning tree in $\mathcal{O}(m \log n)$ [8].

An edge e is *possible* if there is a tree $T \in \mathcal{T}(G, \Omega)$ such that $e \in E(T)$. An edge e is *necessary* if for every tree $T \in \mathcal{T}(G, \Omega)$, we have $e \in E(T)$. Yaman et al. shown that it is possible to determine if an edge is possible or necessary in polynomial time [13].

Theorem 1 ([13]). *Let (G, Ω) be an imprecise weighted graph and let $e \in E(G)$ be an edge. Let $\epsilon > 0$ be an infinitely small positive value.*

(a) Let $R_p \in \mathcal{R}_\Omega$ such that $R_p(e) = \underline{\omega}_e - \epsilon$ and $\forall e' \in E(G-e), R_p(e') = \overline{\omega}_{e'}$. Let T be a minimum spanning tree under R_p, computed with a greedy algorithm. The edge e is possible if and only if $e \in E(T)$.
(b) Let $R_n \in \mathcal{R}_\Omega$ such that $R_n(e) = \overline{\omega}_e + \epsilon$ and $\forall e' \in E(G-e), R_p(e') = \underline{\omega}_{e'}$. Let T be a minimum spanning tree under R_n, computed with a greedy algorithm. The edge e is necessary if and only if $e \in E(T)$.

In other words, an edge e is possible (resp. necessary) if e belongs to a minimum spanning tree under the best (resp. worst) realization for e. The addition (resp. subtraction) of ϵ is needed so that in case of a tie between e and another edge in the greedy algorithm, e is considered first (resp. last). Notice that R_p and R_p are not feasible realizations for (G, Ω). However, any minimum spanning tree under R_p or R_p belongs to $\mathcal{T}(G, \Omega)$.

2.3 Partial solution

Let G be a graph for which we want to enumerate every non-dominated spanning trees. A *partial solution* S is a pair of sets of edges $in(S)$ and $out(S)$ such that there is a tree T in $\mathcal{T}(G, \Omega)$ with $in(S) \subseteq E(T)$ and $out(S) \cap E(T) = \varnothing$ and in that case, we say that T *is associated* to S. We denote $\mathcal{T}_S(G, \Omega)$ the set of trees of $\mathcal{T}(G, \Omega)$ associated to S. We denote $S_\varnothing$ the *empty partial solution* for which $in(S_\varnothing) = out(S_\varnothing) = \varnothing$. Notice that $\mathcal{T}(G, \Omega) = \mathcal{T}_{S_\varnothing}(G, \Omega)$. An example of partial solution is depicted in Figure 2.

Let S be a partial solution. We extend the notion of possible and necessary edges for partial solutions as follows. An edge $e \notin in(S) \cup out(S)$ is *possible* with respect to S if there is a tree $T \in \mathcal{T}_S(G, \Omega)$ such that $e \in E(T)$. Similary, e is *necessary* with respect to S if for all $T \in \mathcal{T}_S(G, \Omega)$, we have $e \in E(T)$. Notice that an edge e is possible (resp. necessary) if and only if e is possible (resp. necessary) with respect to $S_\varnothing$.

An important remark is that we cannot reuse Theorem 1 to determine if an edge is necessary with respect to some partial solution S. For example, consider the partial solution S_2 given by Figure 2: the edge v_2v_5 is necessary with respect to S_2. However, if we consider the realization R_n for which $R(v_2v_5) = 6 + \epsilon$ and $R(e) = \underline{\omega}_e$ for any other edge, then the greedy algorithm returns $T = G - \{v_1v_4, v_2v_5\}$ as a minimum spanning tree of $G - out(S_2)$ which does not belong to $\mathcal{T}_{S_2}(G, \Omega)$. However, it is possible to reuse the same idea than in Theorem 1 to determine if an edge is possible with respect to a partial solution, as we do in this article.

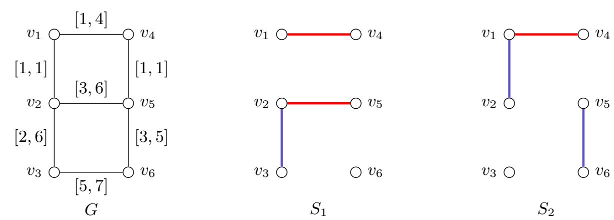

Fig. 2. **Left:** An imprecise weighted graph (G, Ω). **Center:** The pair of edges sets $in(S_1)$ and $out(S_1)$, depicted in blue and red respectively, is not a partial solution. The tree spanning tree $T = G - out(S_1)$ is the only spanning tree such that $in(S_1) \subseteq E(T)$ and $E(T) \cap out(S_1) = \varnothing$. We can observe that T is dominated by $G - \{v_2v_5, v_3v_6\}$. **Right:** Example of a partial solution S_2 with edges of $in(S_2)$ depicted in blue and edges of $out(S_2)$ depicted in red. There are two associated trees $T_1 = G - \{v_1v_4, v_2v_3\}$ and $T_2 = G - \{v_1v_4, v_3v_6\}$ in $\mathcal{T}_{S_2}(G, \Omega)$. The edges v_2v_5 and v_4v_5 are necessary with respect to S_2 and the edges v_2v_3 and v_3v_6 are possible with respect to S_2.

3 Preliminary results

In this section, we present some structural results on partial solutions and cut-set. We first introduce the key concept of core of a cut-set.

Definition 2. *Let (G, Ω) be an imprecise weighted graph and let X be a cut-set in G. An edge $e \in X$ belongs to the* core *of X if there is no edge $e' \in X$ such that e' dominates e. We denote C_X the core of X. Formally,*

$$C_X = \{e \in X \mid \nexists e' \in X, \overline{\omega}_{e'} < \underline{w}_e\}.$$

Let X be a cut-set, we denote e_X an edge such that $e_X = \arg\min\{\overline{w}_e \mid e \in X\}$. Notice that e_X dominates every edge e in $X \setminus C_X$.

We now introduce several structural properties regarding the cores of cut-sets and the non-dominated spanning trees. First, we show that every non-dominated spanning tree intersects the core of each cut-set.

Lemma 1. *Let X be a minimal cut-set. For all tree $T \in \mathcal{T}(G, \Omega)$, we have $C_X \cap E(T) \neq \varnothing$.*

Proof. Toward a contradiction, suppose there is a tree $T \in \mathcal{T}(G, \Omega)$ such that $C_X \cap E(T) = \varnothing$ and consider the edge e_X. Let e be an edge that is X-blocking for e_X in T. By hypothesis, $e \notin C_X$ and so, e is dominated by e_X. Thus, the tree obtained by swapping e_X and e in T dominates T contradicting T belonging to $\mathcal{T}(G, \Omega)$. □

Corollary 1. *Let X be the cut-set induced by a non-dominated spanning tree T and an edge $e \in E(T)$. We have $e \in C_X$.*

Proof. By definition of a cut-set X induced from T and e, we have $E(T) \cap X = \{e\}$. By Lemma 1, $E(T) \cap C_X \neq \varnothing$ which implies $e \in C_X$. □

We now show that it is possible to construct a non-dominated spanning tree from another by swapping two edges that belong to the same core. This allows one, among other things, to simply build a new solution in $\mathcal{T}(G, \Omega)$ from an existing, fully specified one.

Lemma 2. *Let T_1 be a tree of $\mathcal{T}(G, \Omega)$ and let $e_2 \notin E(T_1)$ be an edge that belongs to some core C_X of a cut-set. Let e_1 be a X-blocking edge for e_2 in T_1. The spanning tree T_2 obtained by swapping e_2 and e_1 in T_1 belongs to $\mathcal{T}(G, \Omega)$.*

Proof. Toward a contradiction, suppose there is a tree $T_3 \in \mathcal{T}(G, \Omega)$ that dominates T_2. Let e_3 be an edge of $X \cap E(T_3)$ such that $e_3 = e_1$ if $e_1 \in E(T_3)$ or, e_3 is an edge such that e_1 is X-blocking for e_3 in T_1 otherwise. The edge e_1 does not dominate e_3 since otherwise, the tree obtained by swapping e_3 and e_1 in T_3 dominates T_3, contradicting that T_3 belongs to $\mathcal{T}(G, \Omega)$. Hence, e_1 does not dominate e_3. Further, since $e_2 \in C_X$, e_3 does not dominate e_2. So, $T_3 - e_3$ dominates $T_2 - e_2 = T_1 - e_1$. But then, since e_1 does not dominate e_3, then T_3 dominates T_1, contradicting that T_1 belongs to $\mathcal{T}(G, \Omega)$. Hence, T_2 is not dominated and belongs to $\mathcal{T}(G, \Omega)$. □

Previous lemmas can be used to show some properties on possible/necessary edges with respect to a partial solution. Those properties will be essential in building our enumerating algorithms, as they allow to iteratively complete a current partial solution by adding possible edges to it.

Lemma 3. *Let S be a partial solution and let $e \notin in(S) \cup out(S)$ be an edge.*
(a) e is necessary with respect to S if and only if there is a minimal cut-set X such that $C_X \setminus out(S) = \{e\}$.
(b) e is possible with respect to S if and only if there is a minimal cut-set X such that $e \in C_X$ and $X \cap in(S) = \varnothing$.

Proof. (a) Let X be a minimal cut-set such that $C_X \setminus out(S) = \{e\}$. Then for any tree $T \in \mathcal{T}_S(G, \Omega)$, we have $e \in E(T)$ since otherwise it contradicts Lemma 1. Thus, e is necessary with respect to S. We now show the reciprocity. Let e_1 *not* $\in in(S) \cup out(S)$ be an edge that is necessary with respect to S and let assume by contradiction there is no cut-set X such that $e_1 \in C_X \setminus out(S)$. Let T_1 be a tree of $\mathcal{T}_S(G, \Omega)$ and let X be the cut-set induced by T_1 and e_1.

By hypothesis, there is an edge $e_2 \neq e_1$ in C_X. Let T_2 be a tree such that $E(T_2) = (E(T_1) \setminus \{e_1\}) \cup \{e_2\}$. By Lemma 2, T_2 belongs to $\mathcal{T}_S(G, \Omega)$ and since T_2 does not contain e_1, we obtain a contradiction.

(b) Let X be a cut-set such that $X \cap in(S) = \varnothing$ and let $e \in C_X$. Let T be a tree of $\mathcal{T}_S(G, \Omega)$. If $e \in E(T)$, then e is possible with respect to S. If $e \notin E(T)$, then let T' be a tree obtained by swapping e and a X-blocking edge for e in T. By Lemma 2, T' belongs to $\mathcal{T}(G, \Omega)$, and thus e is possible with respect to S. We now show the reciprocity. Let $e \notin in(S) \cup out(S)$ be an edge that is possible with respect to S and let T be a tree in $\mathcal{T}_S(G, \Omega)$. Let X be the cut-set induced by T and e. By Corollary 1, $e \in C_X$. Moreover, since $X \cap E(T) = \{e\}$ and $e \notin in(S)$, we have $in(S) \cap X = \varnothing$. Hence, there is cut-set X such that $e \in C_X$ and $X \cap in(S) \neq \varnothing$.

□

4 Enumerating algorithm

Having stated our formal results, we are now ready to provide our enumerating algorithms relying on them.

4.1 Possible and necessary edges of partial solutions

In this section, we use Lemma 3 to develop two algorithms that determine if an edge e is possible/necessary with respect to a given partial solution. Informally, the principle of the algorithms is to observe if e closes a cycle in some specific subgraphs (see Figure 3).

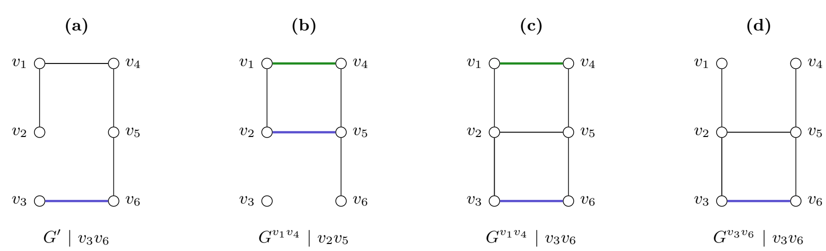

Fig. 3. Subgraphs considered by Algorithms 1 and 2 when the graph (G, Ω) and partial solution S_2 of Figure 2 is given. The edges of the subgraphs are depicted in black and the edge on which the algorithm is called is depicted in blue. **(a)** v_3v_6 is possible with respect to S_2, since v_3 and v_6 are in two different connected components in G'. **(b)** v_2v_5 is necessary with respect to S_2 since v_2v_5 and v_1v_4 lie between the two same connected components $\{v_1, v_2\}$ and $\{v_4, v_5, v_6\}$. **(c)** and **(d)** v_3v_6 is not necessary with respect to S_2 since v_3 and v_6 belong to the same connected component in $G^{v_1v_4}$ and in $G^{v_3v_6}$.

Algorithm 1: is_possible

Data: An imprecise weighted graph (G, Ω), a partial solution S and an edge uv.
Result: `true` if uv is possible with respect to S, `false` otherwise.

Let G' such that $E(G') = \{e \in E(G) \mid e$ `dominates` $uv\} \cup in(S)$;
Let H_1 be the connected component of G' containing u;
Let H_2 be the connected component of G' containing v;
return $H_1 \neq H_2$;

Lemma 4. *Algorithm 1 is correct. Hence, we can determine if an edge is possible with respect to a partial solution in $\mathcal{O}(m+n)$.*

Proof. We show that Algorithm 1 returns `true` if and only if uv is possible with respect to S. First, suppose that the algorithm returns `true`, that is H_1 and H_2 are two different connected components in G'. Let X be the cut-set between $V(H_1)$ and $V(G-H_1)$ in G. Since G' contains more than one connected component, $V(H_1) \neq V(G)$ and so, X is not empty. No edge $e \in X$ dominates uv since otherwise, e would belong to G' and H_1, contradicting the maximality of H_1. Hence, X contains a minimal cut-set X' such that $uv \in C_{X'}$ and $in(S) \cap X' = \varnothing$. By Lemma 3(b), uv is possible with respect to S.

Now suppose that uv is possible with respect to S. By Lemma 3(b), there is a cut-set X such that $uv \in C_X$ and $in(S) \cap X = \varnothing$. That is, no edge of X dominates uv. Hence, by construction of G', no edge of X belongs to G' which implies that u and v belong to two different connected components in G'. Thus, the algorithm returns `true`.

Concerning the running complexity of the algorithm: G' is constructed in $\mathcal{O}(m)$ and determining the connected components of a graph can be done in $\mathcal{O}(m+n)$, so we obtain a complexity in $\mathcal{O}(m+n)$. □

Notice that, since there is no need to sort the edges by increasing order of weight, Algorithm 1 has a better time complexity than the one developed by Yaman et al. [13] to determine if an edge is possible (*i.e.* if we run Algorithm 1 with $S := S_\varnothing$).

Lemma 5. *Algorithm 2 is correct. Hence, we can determine if an edge is necessary with respect to a partial solution S in $\mathcal{O}((|out(S)|+1) \cdot (n+m))$.*

Proof. We show that Algorithm 2 returns `true` if and only if uv is necessary with respect to S. First, suppose that the algorithm returns `true`. That is, there is an edge $xy \in out(S) \cup \{uv\}$ that does not dominate uv and such that:

- H_1^{xy} and H_2^{xy} are two different connected components in G^{xy}, and
- $x \in V(H_1^{xy})$ and $y \in V(H_2^{xy})$, or $x \in V(H_2^{xy})$ and $y \in V(H_1^{xy})$.

Suppose by symmetry that $x \in V(H_1^{xy})$ and $y \in V(H_2^{xy})$. Let X be the cut-set between $V(H_1^{xy})$ and $V(G - H_1^{xy})$. Since G^{xy} contains more than one connected

Algorithm 2: is_necessary

Data: An imprecise weighted graph (G, Ω), a partial solution S and an edge uv.

Result: `true` if uv is necessary with respect to S, `false` otherwise.

forall $xy \in out(S) \cup \{uv\}$ *such that* xy *does not dominate* uv **do**
  Let G^{xy} such that $E(G^{xy}) = \{e \in E(G) \mid xy$ `does not dominate` $e\} \setminus out(S)$;
  Let H_1^{xy} be the connected component of $G^{xy} - uv$ containing u ;
  Let H_2^{xy} be the connected component of $G^{xy} - uv$ containing v ;
  if $H_1^{xy} \neq H_2^{xy}$ **then**
    if $x \in V(H_1^{xy})$ ***and*** $y \in V(H_2^{xy})$ **then**
      return `true`;
    if $x \in V(H_2^{xy})$ ***and*** $y \in V(H_1^{xy})$ **then**
      return `true`;
return `false`;

component, $V(H_1^{xy}) \neq V(G)$ and so, X is not empty. Moreover, since $x \in V(H_1^{xy})$ and $y \notin V(H_1^{xy})$, xy belongs to X. By construction of G^{xy}, any edge $e \in X \setminus (out(S) \cup \{uv\})$ is dominated by xy, since otherwise, e would belong to G^{xy} and H_1, contradicting the maximality of H_1. So, since $xy \in X$, e does not belong to C_X. Hence, X contains a minimal cut-set X' such that $\{uv\} = C_{X'} \setminus out(S)$. So, by Lemma 3(a), uv is necessary with respect to S.

Now suppose that uv is necessary with respect to S. By Lemma 3(a), there is a minimal set-cut X such that $\{uv\} = C_X \setminus out(S)$. If $X \cap out(S) = \varnothing$ or $uv = e_X$, then Algorithm 2 returns `true` when $xy = uv$ in the `forall` loop. Otherwise, there is an edge $e_X \neq uv$ that belongs to $X \cap out(S)$. Consider the step of the `forall` loop for which xy is equal to e_X. Toward a contradiction, suppose xy does not link H_1^{xy} and H_2^{xy} in G^{xy}. Hence, either x or y belongs to a connected component H_3 in G^{xy}, different from H_1^{xy} and H_2^{xy} in $G^{xy} - uv$. Let X' be the cut-set between $V(H_3)$ and $V(G - H_3)$ in G. By construction of G^{xy}, every edge $e \in X' \setminus out(S)$ is dominated by xy, since otherwise e would belong to G^{xy} and H_3, contradicting the maximality of H_3. That is, $e \notin C_{X'}$. Thus, $C_{X'} \setminus out(S) = \varnothing$. It follows that it is not possible to construct a tree T associated to S that respects the property of Lemma 1. Hence, S is not a partial solution which is a contradiction. So, xy links H_1^{xy} and H_2^{xy} in $G^{xy} - uv$. Further, every edge in $X \setminus C_X$ is dominated by xy and thus, does not belong to G^{xy}. It follows that $X \cap E(G^{xy}) = \{uv\}$ and thus, $H_1^{xy} \neq H_2^{xy}$. Hence, the algorithm returns `true`.

Finally, concerning the time complexity of the algorithm: for each edge xy in the `forall` loop, G^{xy} is constructed in $\mathcal{O}(m)$ and determining the connected components of a graph can be done in $\mathcal{O}(m+n)$. Since this process is repeated is at most $|out(S)|+1$ times, we obtain a complexity of $\mathcal{O}((|out(S)|+1) \cdot (m+n))$. □

Notice that, once again, since there is no need to sort the edges by increasing order of weight, Algorithm 2 has a better time complexity than the one developed by Yaman et al. [13] to determine if an edge is necessary. Indeed, if we run Algorithm 2 with $S := S_\varnothing$, then the time complexity is $\mathcal{O}(m+n)$.

4.2 The enumerating algorithm

Now that we developed two polynomial-time algorithms to determine if an edge is possible/necessary with respect to some partial solution, we can enumerate every spanning trees of $\mathcal{T}(G,\Omega)$ with an exhaustive search as depicted by Algorithm 3. Note that, for some partial solution S, an addition of an edge in $out(S)$ or in $in(S)$ does not change the set of possible or necessary edges with respect to S since it does not change $\mathcal{T}_S(G,\Omega)$.

Corollary 2 (Lemma 4 and Lemma 5). *Algorithm 3 is correct. Hence, $\mathcal{T}(G,\Omega)$ can be enumerated in $\mathcal{O}(t(m^3n+m^2n^2))$, where $t=|\mathcal{T}(G,\Omega)|$.*

Proof. We show that the time complexity is correct. At each call of enumeration, the two functions *is_possible* and *is_necessary* are called on each edge in $E(G)\setminus(out(S)\cup in(S))$. So, each call, without taking in account the recursive call, has a complexity of $\mathcal{O}(m(|out(S)|(m+n)))=\mathcal{O}(m^3+m^2n)$. Since the number of edges in a spanning tree is $n-1$, we need $n-1$ recursive calls to display one non-dominated spanning tree, that is, a complexity of $\mathcal{O}(m^3n+m^2n^2)$. Finally, since there are t spanning trees to enumerate, we obtain a time complexity of $\mathcal{O}(t(m^3n+m^2n^2))$. □

Algorithm 3: enumeration

Data: An imprecise weighted graph (G,Ω) and a partial solution S ($S=S_\varnothing$ by default).
Result: Enumeration of $\mathcal{T}(G,\Omega)$

forall $e\in E(G)$ **do**
    if *is_necessary*($(G,\Omega),e,S$) **then**
        $in(S)\leftarrow in(S)\cup\{e\}$;
forall $e\in E(G)$ **do**
    if ***not*** *is_possible*($(G,\Omega),e,S$) **then**
        $out(S)\leftarrow out(S)\cup\{e\}$;
    if $in(S)$ *is a tree* **then**
        Display $in(S)$;
    else
        Let $e\in E(G)\setminus(in(S)\cup out(S))$;
        $S'\leftarrow S$;
        $in(S')\leftarrow in(S')\cup\{e\}$;
        enumeration((G,Ω),S');
        $S'\leftarrow S$;
        $out(S')\leftarrow out(S')\cup\{e\}$;
        enumeration((G,Ω),S');

---

5 Numerical Experiments

In this section, we present some tests on random generated instances. The source code and the instances are available at `https://gitlab.utc.fr/davottom/enum-imst`. We compare Algorithm 3 with the two following methods.

- **Outer approximation.** This method first compute a subgraph G' constituted by the possible and necessary edges in the initial graph. Then, it enumerates every spanning trees of G' that contains all necessary edges. Let t' be the number of (not necessarily non-dominated) spanning trees of G'. The complexity of the outer approximation is $\mathcal{O}(|ST(G)|)$. Note that the size of $ST(G)$ is not bounded by some polynomial function in the size of $\mathcal{T}(G, \Omega)$.
- **Reduce.** This method uses same algorithm than the outer approximation plus check for each spanning tree T of G' if T is non-dominated. To check if a tree T is non-dominated, we use the same idea as the one described in Theorem 1: we compute a minimum spanning tree in the realization R where $R(e) = \underline{\omega}_e - \epsilon$ if $e \in E(T)$ and, $R(e) = \overline{\omega}_e$, otherwise. The complexity of the reduce algorithm is $\mathcal{O}(|ST(G)| \cdot m \log n)$.

In the following, we refer to Algorithm 3 as the exact method.

5.1 Instances

We generated imprecise weighted graphs with 10 vertices by varying the density of the graph and the weight function. We chose to generate the instances according three graph densities and three scenarios for the weight function. The three possible densities *sparse*, *middle*, *dense* for which the graph contains $15, 25$ and 35 edges, respectively. The graph is generated using the random generator of the library boost in C++. If the graph is not connected, we add a random edge between two connected components until the graph is connected. For the generation of weight functions, given a scenario i for each edge e, we pick two random numbers $\ell \in [1, 10]$ $s \in [a_i, b_i]$, where a_i and b_i depend on the selected scenario. Then, we set $\Omega(e) = [\ell, \ell + s]$. For scenario 1, we have $a_i = 1$ and $b_i = 10$, for scenario 2, we have $a_i = 7$ and $b_i = 9$ and, for scenario 3, we have $a_i = 2$ and $b_i = 3$. Note that scenario 1 generates intervals with quite varying sizes, while scenario 2 generates intervals that will very often overlap. For each scenario and each density, we generate 10 instances.

5.2 Results

The tests were run on a personal laptop with 16Go of RAM and with an Intel Core 7 processor 2.5Ghz. The results are depicted in Tables 1 and 2. Not surprisingly, the outer approximation is the fastest method. Although the theoretical time complexity of the exact method is better than the reduce method, the latter is faster on the generated dataset (except in Scenario 3). In particular, the worst case for the exact method occurs in the set of dense graphs with the scenario 2 where the maximum computation time for the exact method takes more than 1 minute whereas the reduce method uses only 18 seconds. Regarding the statistics on the number of trees enumerated, the denser the graph, the bigger the cardinality of the enumerated sets for both methods. Samewise, the larger the intervals (*i.e.* in scenario 1), the bigger the cardinality of the enumerated sets. We can also observe than when the graph is not dense, the outer approximation seems reasonably close to the exact method.

Table 1. Time statistics. A set contains every graphs generated with the same density and scenario. For each set and each method, average, minimum and maximum times are depicted.

Set		Exact			Approx			Reduce		
density	scenario	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
dense	1	173ms	93ms	22s	120ms	82ms	8s	160ms	109ms	11s
middle	1	40ms	7ms	401ms	23ms	4ms	411ms	31ms	5ms	530ms
sparse	1	<1ms	<1ms	2ms	<1ms	<1ms	<1ms	<1ms	<1ms	2ms
dense	2	6s	13s	1m1s	1s	10s	14s	2s	13s	18s
middle	2	89ms	211ms	1s	35ms	181ms	400ms	45ms	229ms	510ms
sparse	2	<1ms	<1ms	2ms	<1ms	<1ms	1ms	<1ms	<1ms	1ms
dense	3	3ms	1ms	69ms	39ms	<1ms	386ms	48ms	<1ms	483ms
middle	3	<1ms	<1ms	9ms	<1ms	<1ms	29ms	<1ms	<1ms	36ms
sparse	3	<1ms	<1ms	<1ms	<1ms	<1ms	<1ms	<1ms	<1ms	<1ms

Table 2. Result statistics on the number of enumerated trees. A set contains every graph generated with the same density and scenario. Exact and Approx: number of enumerated trees for the corresponding method. The Diff column is the difference of cardinality between the exact method and the outer approximation.

Set		Exact			Approx			Diff		
dens.	scen.	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
dense	1	708,107	12,984	3.3M	1.3M	53,956	5M	656,372	18,576	1.9M
middle	1	23,548	1,476	84,936	56,837	3,012	29,6340	33,289	872	216,852
sparse	1	201	29	445	287	29	763	86	0	18
dense	2	5M	1.6M	8.2M	7.7M	6.3M	8.6M	2.7M	241,424	5,5M
middle	2	151,517	36,426	227,902	231,516	135,185	296,340	80,000	0	157,855
sparse	2	581	264	944	682	354	944	100	0	224
dense	3	4,533	222	9,857	41,279	304	261,134	36,746	82	257,214
middle	3	464	24	2,445	3,024	48	23,135	2,560	22	20,690
sparse	3	46	8	175	82	11	286	36	2	111

6 Conclusions

In this paper, we have considered the problem of enumerating non-dominated spanning trees in the case of interval-valued weights, and have provided an efficient algorithm to do so.

There are at least two directions in which we would like to extend the results presented in this paper: a first one is to consider more general combinatorial optimisation problems such as matroids, as those mostly remain tractable when considering intervals [6]. A second one would be to consider more general uncertainty models, such as possibility distributions [4], belief functions [12] or credal sets [11,3].

References

1. Ionut D. Aron and Pascal Van Hentenryck. On the complexity of the robust spanning tree problem with interval data. *Operation Research Letter*, 32(1):36–40, 2004.
2. Nawal Benabbou and Patrice Perny. On possibly optimal tradeoffs in multicriteria spanning tree problems. In *Algorithmic Decision Theory - 4th International Conference, ADT 2015, Lexington, KY, USA, September 27-30, 2015, Proceedings*, pages 322–337, 2015.
3. Sébastien Destercke and Romain Guillaume. Necessary and possibly optimal items in selecting problems. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems: 19th International Conference, IPMU 2022, Milan, Italy, July 11–15, 2022, Proceedings, Part I*, pages 494–503. Springer, 2022.
4. Romain Guillaume, Adam Kasperski, and Paweł Zieliński. Distributionally robust possibilistic optimization problems. *Fuzzy Sets and Systems*, 454:56–73, 2023.
5. Mikita Hradovich, Adam Kasperski, and Pawel Zielinski. The recoverable robust spanning tree problem with interval costs is polynomially solvable. *Optimization Letters*, 11(1):17–30, 2017.
6. Adam Kasperski. *Discrete optimization with interval data*. Springer, 2008.
7. Galina L. Kozina and Vitaly A. Perepelitsa. Interval spanning trees problem: solvability and computational complexity. *Interval Computations*, 1(1):42–50, 1994.
8. Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.
9. Roberto Montemanni and Luca M. Gambardella. A branch and bound algorithm for the robust spanning tree problem with interval data. *European Journal of Operational Research*, 161(3):771–779, 2005.
10. Petrică C Pop. The generalized minimum spanning tree problem: An overview of formulations, solution procedures and latest advances. *European Journal of Operational Research*, 283(1):1–15, 2020.
11. Erik Quaeghebeur, Keivan Shariatmadar, and Gert De Cooman. Constrained optimization problems under uncertainty with coherent lower previsions. *Fuzzy Sets and Systems*, 206:74–88, 2012.
12. Tuan-Anh Vu, Sohaib Afifi, Éric Lefèvre, and Frédéric Pichon. On modelling and solving the shortest path problem with evidential weights. In *Belief Functions: Theory and Applications: 7th International Conference, BELIEF 2022, Paris, France, October 26–28, 2022, Proceedings*, pages 139–149. Springer, 2022.
13. Hande Yaman, Oya Ekin Karaşan, and Mustafa Ç. Pınar. The robust spanning tree problem with interval data. *Operations Research Letters*, 29(1):31–40, 2001.