

## AN ENUMERATIVE ALGORITHM FOR THE MINIMUM WEIGHT $t$ -SPANNER TREE PROBLEM\*

**Gabriel H. de Sousa**

ParGO, Departamento de Computação, Universidade Federal do Ceará  
Av. Humberto Monte, s/n, Campus do Pici, Bloco 910. CEP 60440-900. Fortaleza, Ceará, Brasil  
gabrielsousa.9957@gmail.com

**Manoel Campêlo**

ParGO, Departamento de Estatística e Matemática Aplicada, Universidade Federal do Ceará  
Av. Humberto Monte, s/n, Campus do Pici, Bloco 910. CEP 60440-900. Fortaleza, Ceará, Brasil  
mcampelo@lia.ufc.br

### ABSTRACT

In the Minimum Weight  $t$ -Spanner Tree Problem, whose input is a triplet  $(G, w, t)$ , where  $G = (V, E)$  is a simple graph,  $w : E \rightarrow \mathbb{R}^+$  is an edge weighting function and  $t \geq 1$  is a parameter, called the dilatation factor, the objective consists in determining a spanning tree  $T$  of  $G$  of minimum weight among those where the path between each pair of vertices  $i$  and  $j$  in tree  $T$  has weight at most  $t$  times the minimum weight of a path between  $i$  and  $j$  in  $G$ . It is known that determining a Minimum Weight  $t$ -Spanner Tree is an NP-hard problem even on the unit costs case. In this work, we propose an enumerative algorithm to solve this problem. It is the first one that does not depend on solving mathematical programs. We implemented the algorithm and compared the computational performance with the most efficient formulation from the literature.

**KEYWORDS.** Minimum Weight  $t$ -Spanner Tree. Enumerative Algorithm. Spanning Tree.

**Topics :** Combinatorial Optimization, Theory and Algorithms in Graphs, Mathematical Programming.

### RESUMO

No problema da árvore  $t$ -expansora de peso mínimo, cuja entrada é uma tripla  $(G, w, t)$ , onde  $G = (V, E)$  é um grafo simples,  $w : E \rightarrow \mathbb{R}^+$  é uma função de ponderação das arestas e  $t \geq 1$  é um parâmetro, chamado fator de dilatação, o objetivo consiste em determinar uma árvore geradora  $T$  de peso mínimo entre aquelas onde o peso do caminho entre cada par de vértices  $i$  e  $j$  em  $T$  é no máximo  $t$  vezes o peso de um caminho mínimo entre  $i$  e  $j$  em  $G$ . Sabe-se que determinar uma árvore  $t$ -expansora de peso mínimo é um problema NP-Difícil mesmo no caso de pesos unitários. Neste trabalho, propomos um algoritmo enumerativo para resolver esse problema. Ele é o primeiro a não depender da resolução de formulação de programação matemática. Implementamos o algoritmo e comparamos seu desempenho computacional com a formulação mais eficiente da literatura.

**PALAVRAS CHAVE.** Árvore  $t$ -Expansora de Peso Mínimo, Algoritmo Enumerativo, Árvore Geradora.

**Tópicos:** Otimização Combinatória, Teoria e Algoritmos em Grafos, Programação Matemática.

\*This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Partially supported by CNPq 309315/2019-0, FUNCAP PNE-0112-00061.01.00/16.

## 1. Introduction

Let  $G = (V, E)$  be a simple graph, where  $V$  is the vertex set, and  $E$  is the edge set. A subgraph  $H = (V', E')$  of  $G$  is another graph whose vertices and edges are subsets of the vertices and edges of  $G$ . The subgraph  $H$  is called a spanning graph if  $V' = V$ , and, if connected and acyclic,  $H$  is called a spanning tree of  $G$ . When we have a weighting function  $w : E \rightarrow \mathbb{R}$ , that assigns a real value to each edge  $e \in E(G)$ , we say that  $G$  is a weighted graph. In this case, the weight (or cost) of the graph is the sum of the weights of its edges. The distance  $dist_G(i, j)$  between two vertices  $i, j \in V(G)$  is the minimum weight of a path in  $G$  between  $i$  and  $j$ .

Given a triple  $(G, w, t)$ , where  $(G, w)$  is a graph with non-negative edge weights, and  $t \geq 1$  is a real number, called dilatation factor, a  $t$ -spanner of  $G$  is a spanning subgraph  $H$  such that

$$dist_H(i, j) \leq t * dist_G(i, j), \quad \forall i, j \in V(G) \quad (1)$$

It can be shown that it is enough to require condition (1) for  $\{i, j\} \in E(G)$  [Cai and Coneil, 1995].

Spanner subgraphs problems with predetermined features appear in various scenarios such as distributed computing, [Awerbuch, 1985; Peleg and Ullman, 1989], communication network [Peleg and Upfal, 1988; Peleg and Reshef, 1999; Oliveira and Pardalos, 2005], location and distribution [Goldschmidt et al., 1996], robotic [Marble and Bekris, 2011b; Arikati et al., 1996; Marble and Bekris, 2011a] and bioinformatics [Nguyen et al., 2008]. In this work, we focus on one of the problems about spanner subgraphs: the Minimum Weight  $t$ -Spanner Tree Problem (MWtSTP).

The MWtSTP is the composition of two optimization problems: the minimum spanning tree problem (MST) and minimum  $t$ -spanner problem (MtST). Given the triple  $(G, w, t)$ , it consists in determining a  $t$ -spanner of  $G$  that is a tree, if any, with minimum weight. Figures 1, 2(a) and 2(b) illustrate, respectively, an input graph  $G$ , a minimum  $t$ -spanner tree for  $t = 3$  and a minimum spanning tree. The number close to each edge represents its weight. When  $w_e = 1$ , for every  $e \in E(G)$ , any spanning tree will have a weight of  $|V(G)| - 1$ . So the optimization problem becomes a decision problem, in which we want to determine whether  $G$  admits a  $t$ -spanner tree or not. In this case, we will denote the problem as  $t$ -spanner tree problem (tSTP).

Figure 1: Graph  $G$ ,  $t = 3$

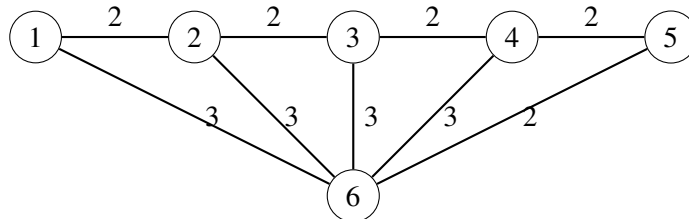
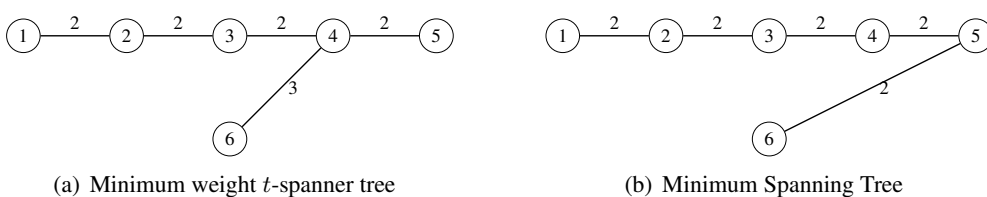


Figure 2: Spanning Trees



In the unit weights case, tSTP is  $NP$ -complete for  $t \geq 4$  [Brandstadt et al., 2004; Dragan et al., 2011]. When  $t = 1$  the problem becomes trivial, since  $G$  admits a 1-spanner tree, if, and only

if,  $G$  is a tree [Cai and Coneil, 1995]. For  $t = 2$ , the problem of checking whether  $G$  has a  $t$ -spanner tree is also polynomial [Cai, 1992; Cai and Coneil, 1995]. For  $t = 3$ , the complexity is still open.

When the graph weights are arbitrary, we speak of MWtSTP. Since tSTP instances are a subset of MWtSTP instances,  $NP$ -complete cases for the former translate into  $NP$ -complete cases for the latter. Furthermore, for  $t > 1$  fixed, checking whether there is a  $t$ -spanner tree in a weighted graph is an  $NP$ -complete problem [Cai and Coneil, 1995].

The Minimum Weight  $t$ -Spanner Tree Problem has been extensively explored for the case with unit weights (tSTP). For instance, polynomial algorithms for tSTP on some classes of graphs, for a fixed dilatation factor, are known [e.g. Cai, 1992; Madanlal et al., 1996; Fekete and Kremer, 2001; Narayanaswamy and Ramakrishna, 2015; Venkatesan et al., 1997; Papoutsakis, 2018; Brandstadt et al., 1999, 2004; Papoutsakis, 2014]. On the other hand, there are still few studies for the case with arbitrary weights, particularly when it comes to mathematical programming. In the literature of MWtSTP, we found only one exact algorithm and two mathematical formulations [Braga, 2019].

In this work, we introduce an algorithm that determines a minimum weight  $t$ -spanner tree of  $G$  or shows that there is no such tree. The algorithm is based on an enumerative *branch-and-prune* procedure, in which each subproblem is solved as a minimum spanning tree problem, through an adaptation of the Kruskal Algorithm [Kruskal, 1956]. We will denote by MWtST\* and MST\* an optimal solution of MWtSTP and MST, respectively. In the next section we formally describe the enumerative algorithm.

## 2. Enumerative Algorithm

Let  $(G, w, t)$  be an instance of MWtSTP. Note that the weight of an MST\* is a lower bound for an MWtST\*, since any MWtST\* is a feasible solution for MST. Besides, if an MST\* of  $G$  is a  $t$ -spanner, then it will itself be an MWtST\*. Otherwise, it means that some constraint (1) is violated, that is, there is a path in this MST\*, between some vertices  $i$  and  $j$ , whose weight is greater than  $t * dist_G(i, j)$ . In general, a path  $P$  is said to be  $t$ -infeasible if  $dist_P(i, j) > t * dist_G(i, j)$ .

Following this observation, the key idea of our algorithm is to enumerate spanning trees of  $G$  and, given a spanning tree  $T$  that is not a  $t$ -spanner, to prevent an  $i, j$ -path of  $T$  that is  $t$ -infeasible. For example, considering the MWtSTP instance shown in Figure 3(c), the path  $P = \langle 1, 3, 4, 2 \rangle$  in Figure 3(d) is 3-infeasible, since  $dist_P(1, 2) = 9 > 6 = 3 * dist_G(1, 2)$ .

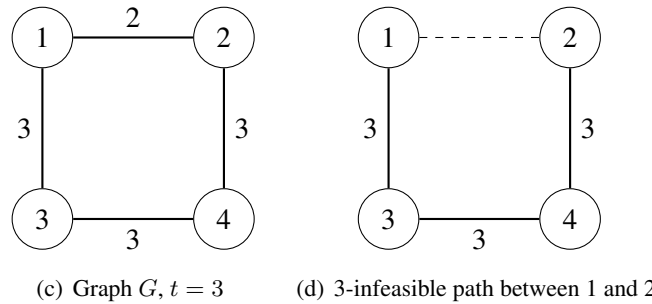


Figure 3: Illustration of  $t$ -infeasible path.

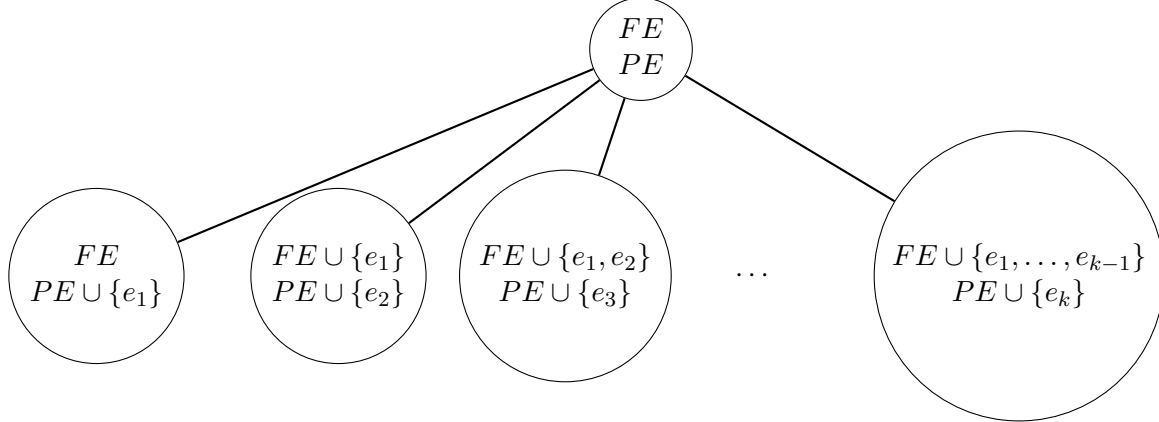
The algorithm starts by finding an MST\*. If it is a  $t$ -spanner, the procedure stops returning  $MStST^* = MST^*$ . Otherwise, we select a  $t$ -infeasible path in  $MST^*$  and partition the problem into new subproblems. In each subproblem, we prohibit an edge of the  $t$ -infeasible path. In the following section, we will see in detail how the branching process occurs.

## 2.1. Branching Procedure

Let  $P = \langle v_1, v_2, \dots, v_k, v_{k+1} \rangle$ ,  $k \geq 1$ , be an  $t$ -infeasible path. Therefore, the edges  $e_i = \{v_i, v_{i+1}\}$ , for  $i = 1, \dots, k$ , cannot be selected simultaneously to compose a  $t$ -spanner tree, that is, at most  $k - 1$  of these edges belong to a  $t$ -spanner tree. We are going to partition the feasible set into  $k$  disjoint subsets. In the  $i$ -th subset, edge  $e_i$  is prohibited, and edges  $e_1, \dots, e_{i-1}$  are mandatory. The best solution, if any, of the subproblems defined by these subsets is a solution to the original problem. This partitioning can be applied recursively, thus generating a tree. It is worth mentioning that this branching strategy has already been used in the literature, e.g. for the asymmetric capacitated vehicle routing problem [Batsyn et al., 2013] and the degree constrained minimum spanning tree problem [Andrade and Freitas, 2013].

Formally, let  $S = (G, FE, PE)$  be a node in the search tree, where  $FE \subseteq E(G)$  and  $PE \subseteq E(G)$  are the subsets of fixed and prohibited edges on that node, respectively. We denote by  $MST(S)$  the problem of determining a minimum spanning tree of  $G \setminus PE$  containing  $FE$ . If it exists, let  $T$  be a solution to this problem. If it satisfies (1),  $T$  is an optimal  $t$ -spanner tree for MWtSTP restricted to  $S$ . Otherwise, let  $P = \langle v_1, v_2, \dots, v_k, v_{k+1} \rangle$  be a  $t$ -infeasible path in  $T$ , and  $e_i = \{v_i, v_{i+1}\}$ , for  $i = 1, 2, \dots, k$ . In this case,  $S$  will be branched into  $k$  subproblems,  $S_i = (G, FE_i, PE_i)$ , for  $i \in \{1, 2, \dots, k\}$ , where  $FE_1 = FE$ ,  $PE_1 = PE \cup \{e_1\}$ , and  $FE_i = FE \cup \{e_1, \dots, e_{i-1}\}$ ,  $PE_i = PE \cup \{e_i\}$ , for  $2 \leq i \leq k$ . See Figure 4.

Figure 4: Branching on Path  $P = \langle v_1, v_2, \dots, v_k, v_{k+1} \rangle$ , with  $e_i = \{v_i, v_{i+1}\}$



Note that any spanning tree containing  $P$  (in particular  $T$ ) is not feasible for each  $MST(S_i)$  subproblem, as it prohibits edge  $e_i \in E(P)$ . In addition, any spanning tree of  $G \setminus AP$  containing  $AF$  is feasible for one of these subproblems, except for those containing  $P$ . In fact, consider a tree  $T^*$  that is feasible for  $MST(S)$  and does not contain all of the edges of  $P$ . Let  $e_c$  be the first edge of  $P$  that is not in  $T^*$ , and therefore  $\{e, \dots, e_{c-1}\} \subset E(T^*)$ . Since  $T^*$  is a feasible solution of  $MST(S)$ , we have  $FE \subset E(T^*)$  and  $PE \cap E(T^*) = \emptyset$ . So  $T^*$  is feasible for  $S_c$ , since all edges of  $FE_c = FE \cup \{e_1, \dots, e_{c-1}\}$  are in  $E(T^*)$ , and every edge of  $PE_c = AP \cup \{e_c\}$  is missing. In other words, at each branch we only prohibit the presence of infeasible paths in the solution, which implies that the solution to the original problem is contained in a of these subproblems. As each of them is a more restricted version of the original problem, with a set of fixed or prohibited edges, eventually, we will find, if it exists, a solution that does not contain  $t$ -infeasible paths, that is, a  $t$ -spanner tree.

Given a solution to a subproblem  $MST(S)$ , we can have several  $t$ -infeasible paths, all of them are candidates for branching. We will select, among them, the one that has the lowest cost. This leads us to have fewer branches at a node. For example, consider a graph  $G$  where  $dist_G(1, 5) = 4$  and  $dist_G(3, 5) = 2$ , and a dilatation factor  $t = 3$ . Suppose that the solution tree for a given subproblem contains the path shown in Figure 5. Note that this path as well as the subpath between 3 and 5 are 3-infeasible. In Figures 6 and 7, we present a possible branching at this solution corresponding to each of these two paths. Note that the branching relative to the path  $\langle 3, 4, 5 \rangle$  is prohibiting, in addition to it, the path  $\langle 1, 2, 3, 4, 5 \rangle$ , since the edges of the former are a subset of the edges of latter. Besides, the number of generated subproblems is smaller, due to the fact that the number of edges of the path  $\langle 3, 4, 5 \rangle$  is strictly smaller. So, we will select one of the  $t$ -infeasible paths with the least weight to generate the branching.

Figure 5:  $t$ -infeasible paths to branch

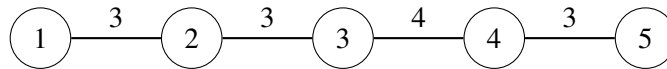


Figure 6: Branching on path 1-5

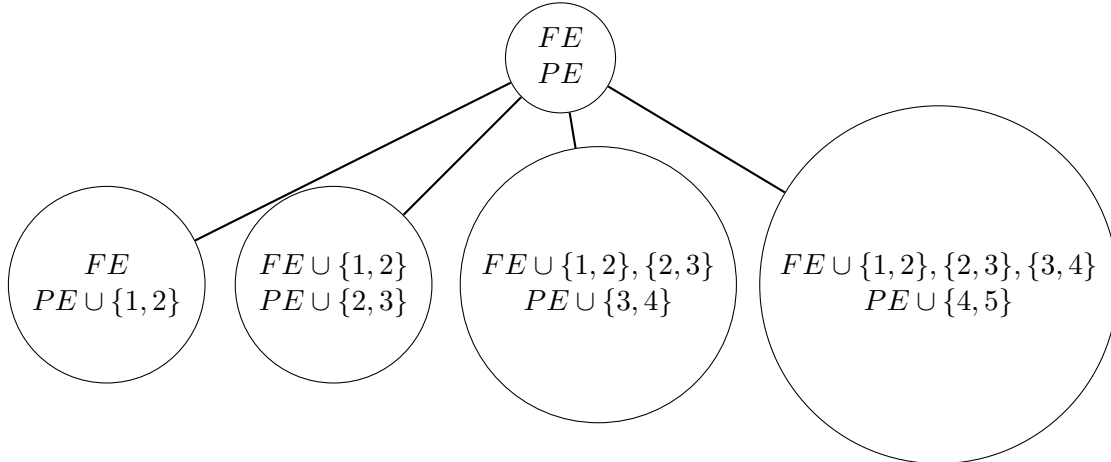
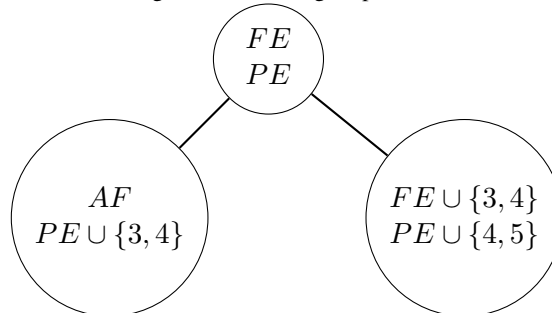


Figure 7: Branching on path 3-5



A pseudocode of the branching procedure can be found in Algorithm 1. The *branch-and-bound* tree is stored in a stack,  $B\&BTree$ , so the enumeration follows a *backtracking* process. Each node in the tree stores, in addition to  $S = (G, FE, PE)$ , the value of a lower bound for the subproblem restricted to  $S$ , to be given by the optimum of  $MST(S)$ , that is, by the minimum weight

of a spanning tree of  $G \setminus PE$  containing  $FE$ . When creating the node, this lower bound is initialized with that of the parent node.

---

**Algorithm 1** Branch ( $B\&BTree, T^*$ )

---

```

1:  $\{e_1, e_2, \dots, e_k\} \leftarrow \arg \min_{i,j \in V(G), P_{ij} \in T^*} \{w(P_{ij}) : w(P_{ij}) > t * dist_G(i, j)\}$ 
2:  $(G, FE, PE, w) \leftarrow B\&BTree[Size(B\&BTree)]$ 
3: Pop Back  $B\&BTree$ 
4:  $FE^* \leftarrow FE$ 
5: for  $i \leftarrow 1$  to  $k$  do
6:    $PE^* \leftarrow PE \cup \{e_i\}$ 
7:   for  $j \leftarrow 1$  to  $i - 1$  do
8:      $FE^* \leftarrow FE^* \cup \{e_j\}$ 
9:   end for
10:  for  $e' \in E(G) \setminus FE^*$  do
11:    if  $e' \cup FE^*$  is cyclic then
12:       $PE^* \leftarrow PE^* \cup e'$ 
13:    end if
14:  end for
15:  Push Back  $(G, FE^*, PE^*, w(T^*))$  in  $B\&BTree$ 
16: end for
17: Return

```

---

## 2.2. Subproblems solving

To solve each  $MST(S)$  subproblem, we will adapt the Kruskal Algorithm [Kruskal, 1956], a classical algorithm to determine a minimum spanning tree of a graph. Note that  $MST(G, \emptyset, \emptyset)$  corresponds to  $MST$  in  $G$ . In general, Kruskal's algorithm receives a weighted connected graph  $G = (V, E, w)$  and an ordering  $E^* = (e_1^*, e_2^*, \dots, e_{|E(G)|}^*)$  of its edge set, where  $w(e_i^*) \leq w(e_{i+1}^*)$ , for  $i \in \{1, \dots, |E(G)| - 1\}$ . It starts with a spanning subgraph  $T = (V(G), \emptyset)$ , with no edges. At each iteration  $i$ , from 1 to  $|E(G)|$ , if adding edge  $e_i^*$  does not generate a cycle in  $T$ , then it will be indeed included in  $E(T)$ ; in other words, if  $(V(G), E(T) \cup \{e_i^*\})$  is acyclic, then  $E(T) \leftarrow E(T) \cup \{e_i^*\}$ . It stops when  $|E(T)| = |V| - 1$ . We will adapt the algorithm to solve a given subproblem  $MST(S)$ , with  $S = (G, FE, PE)$ . See Algorithm 2. As for set  $PE$ , let's just assume that its edges do not "exist" in the algorithm's input graph, that is, let's ignore each edge in  $PE$  at step 4 of the algorithm. On the other hand, the edges in  $FE$  will be pre-selected for  $T$ , as shows step 1, and then ignored in step 4, just as done with  $PE$ .

The weight  $w(T)$  of the spanning tree  $T$  returned by Algorithm 2 is a lower bound of the current node and should be stored. However, as we are going to branch this node immediately, this will not be necessary.

## 2.3. Pruning Procedure

After solving a subproblem, we extract some information from the solution. The lowest of all (relaxed) solution costs for the subproblems solved so far gives a lower bound for the cost of  $MWtST^*$ . On the other hand, the lowest cost of these solutions that are  $t$ -spanner trees gives an upper bound for the optimum of the problem. Based on this information, we can ignore subproblems such that the best solution it can generate for  $MWtSTP$  is greater than a feasible solution found so far. Next, we will present the pruning criteria for a given subproblem  $S = (G, FE, PE)$  of the enumerative algorithm. The pseudocode of the pruning procedure is showed in Algorithm 3.

---

**Algorithm 2** Modified Kruskal ( $S=(G, FE, PE), E^*$ )

---

```

1:  $T \leftarrow (V(G), FE)$ 
2:  $i \leftarrow 1$ 
3: while  $|E(T)| < |V(G)| - 1$  and  $i \leq |E^*|$  do
4:   if  $(V(G), E(T) \cup \{e_i^*\})$  is acyclic,  $e_i^* \notin FE$  and  $e_i^* \notin PE$  then
5:      $E(T) \leftarrow E(T) \cup \{e_i^*\}$ 
6:   end if
7:    $i \leftarrow i + 1$ 
8: end while
9: Return  $T$ 

```

---



---

**Algorithm 3** Prune ( $B\&BTree, T^*, PE, UB, T_{OPT}$ )

---

```

1: if  $T^*$  is  $t$ -spanner tree then
2:   if  $w(T^*) < UB$  then
3:      $UB \leftarrow w(T^*)$ 
4:      $T_{OPT} \leftarrow T^*$ 
5:   end if
6:   Pop back  $B\&BTree$ 
7:   Return 1
8: end if
9: if  $w(T^*) \geq UB$  then
10:  Pop back  $B\&BTree$ 
11:  Return 1
12: end if
13: for all  $i, j \in V(G)$  do
14:  if  $\text{dist}_{(V, E \setminus PE)}(i, j) > t * \text{dist}_G(i, j)$  then
15:    Pop back  $B\&BTree$ 
16:    Return 1
17:  end if
18: end for
19: Return 0

```

---

**Optimality Pruning** If the current MST( $S$ ) subproblem solution  $T^*$  is a  $t$ -spanner tree, i.e.  $\text{dist}_{T^*}(u, v) \leq t * \text{dist}_G(u, v)$ , for every  $\{u, v\} \in E(G)$ , then  $T^*$  is optimum for MWtSTP restricted to  $S$  and feasible for MWtSTP on  $G$ . In this case, we prune the subproblem by *optimality*. In addition, if the cost of  $T^*$  is the best ever found for a feasible solution, we update the upper bound  $UB$  for the optimal solution to  $w(T^*)$ . See steps 1–8 of Algorithm 3.

**Bound Pruning** If the solution cost of MST( $S$ ) is greater than or equal to the upper bound  $UB$  of the optimal solution, we can prune the subproblem by *bound*, as the best solution that it could generate would have cost at least  $UB$ , the value of a feasible solution already known. See steps 9–12 of Algorithm 3.

**Infeasibility Pruning** If there are two vertices  $i$  and  $j$  such that  $\text{dist}_{(V, E \setminus PE)}(i, j) > t * \text{dist}_G(i, j)$ , then we can prune the current subproblem by *infeasibility*, since the minimum path between  $i$  and  $j$  in the subproblem's input graph already violates constraint (1). This pruning corresponds to steps 13–18 of Algorithm 3.

### 3. Pseudocode of EA

Now we will combine the procedures seen above to define our enumerative algorithm, to be denoted EA. A pseudocode is presented in Algorithm 4. As anticipated, we will store search tree in a stack, which we will call  $B\&BTree$ . This implies that the last stacked subproblem will be the first to be solved. Each  $i$  element in the  $B\&BTree$  stack will store the fixed ( $FE$ ) and prohibited ( $PE$ ) edges in that node, as well as a lower bound ( $w$ ) for the solution of this subproblem.  $LB$  and  $UB$  will store the lower and upper bounds for MWtST\*, respectively. Note that  $LB = \min_{1 \leq i \leq \text{Size}(B\&BTree)} w(i)$ . The algorithm stops when  $B\&BTree$  becomes empty, which means that the solution space was fully explored, unless for pruned subproblems, or by an eventual equality between the lower and upper bounds, which guarantees optimality.

---

#### Algorithm 4 Enumerative Algorithm ( $G, t$ )

---

```

1: Stack  $B\&BTree \leftarrow \emptyset$ 
2:  $E^* \leftarrow$  Sorting  $E(G)$  by increasing weights
3:  $T \leftarrow$  Modified Kruskal ( $G, \emptyset, \emptyset, E^*$ )
4: if  $T$  is  $t$ -spanner then
5:   Return  $T$ 
6: end if
7: Push Back ( $G, \emptyset, \emptyset, w(T)$ ) in  $B\&BTree$ 
8:  $UB \leftarrow |V(G)| * \max_{(i,j) \in E(G)} \{w_{ij}\}$ 
9:  $T^* \leftarrow \emptyset$ 
10:  $LB \leftarrow 0$ 
11: while  $\text{Size}(B\&BTree) > 0$  e  $LB < UB$  do
12:   Branch ( $B\&BTree, T$ )
13:    $j \leftarrow \text{Size}(B\&BTree)$ 
14:    $LB \leftarrow \min_{1 \leq i \leq j} w(B\&BTree[i])$ 
15:    $T \leftarrow$  Modified Kruskal ( $G, FE(B\&BTree[j]), PE(B\&BTree[j]), E^*$ )
16:   while Prune ( $B\&BTree, T, PE(B\&BTree[j]), UB, T^*) = 1$  and  $j > 0$  do
17:      $LB \leftarrow \min_{1 \leq i \leq j} w(B\&BTree[i])$ 
18:      $j \leftarrow j - 1$ 
19:      $T \leftarrow$  Modified Kruskal ( $G, FE(B\&BTree[j]), PE(B\&BTree[j]), E^*$ )
20:   end while
21: end while
22: Return  $T^*$ 

```

---

## 4. Computational Results

### 4.1. Instances

A unique set of (randomly generated) instances has been used for computational experiments with MWtSTP solving methods [Braga, 2019]. Unfortunately, it is no longer accessible. So, we generated new instances using the procedure described in Braga [2019]. To generate each graph, we use two input parameters: the number of vertices  $n \in \mathbb{N}^*$ ; and a probability  $p \in [0, 1]$  for the edge occurrence. As for the weighting of the edges, we considered two cases: unit weights (tSTP) and random weights (MWtSTP). Note that the problem is  $NP$ -Hard in both cases. The ranges used each parameter are:

- Number of vertices:  $n \in \{15, 30, 45, 60\}$ .

- Edge occurrence probability:  $p \in \{0.2, 0.5, 0.8, 1\}$ .
- Dilatation factor:  $t \in \{1.1, 2, 3, 4\}$ .
- Edge Weighting:
  - Unit weights ( $w_e = 1$ , for all  $e \in E(G)$ ).
  - Arbitrary weights (for all  $e \in E(G)$ ,  $w_e \in \{1.2, 4, 8, 16\}$  with equal probability)

Let  $n$ ,  $p$  and  $t$  be fixed. To ensure connectivity, we start the instance graph as a tree with  $n$  vertices. The edges of this tree are sequentially selected at random with equal probability for each remaining pair of vertices that does not form a cycle with the edges already chosen. Then, for every pair  $\{i, j\} \in V(G)$  that does not form a cycle with graph created so far, we generate a random number  $p' \in [0, 1]$ . If  $p' \leq p$ , we add the edge  $\{i, j\}$  to the graph. Note that the higher this factor  $p$ , the greater the number of edges of the graph, which is directly reflected in its density. At this point, we have already generated an instance for the decision version of the problem (tSTP). If the instance is weighted, for each edge  $\{i, j\} \in E(G)$ , we randomly select a weight from the elements of the set  $\{1, 2, 4, 8, 16\}$ , each with the same choice probability.

## 4.2. Experiments

The enumerative algorithm (EA) was implemented in programming language C++. The experiments were carried out on a machine with an Intel® Core™ i7-7700 CPU with 3.60 GHz, 32 GB RAM and operational system Ubuntu. The time limit set for resolving each instance was 1200 seconds (20 minutes).

We have run EA and the MTZ Formulation (CR) proposed in Braga [2019] on the generated instances. Actually, we have implemented a strengthened version of CR with smaller big-M values (see Sousa [2021] for details). Table 1 compares the obtained results. Columns ‘Id’, ‘n’, ‘p’ and ‘t’ present the instance parameters: identification number, number of vertices of the graph, probability of edge occurrence, dilatation factor. For each combination of solving method (EA or CR) and problem version (MWtSTP or tSTP), we show two columns named ‘Time’ and ‘Status’ that respectively presents the computational time (in seconds) and the final method result (the optimum value, ‘inf’ to indicates that the instance is infeasible, or ‘?’ to mark that the algorithm could not prove optimality or infeasibility within the time limit). In the last row (‘solved’), we compute the percentage of solved instances and the average time for these instances.

We can observe that the algorithm demanded more computational time, on average, for instances with unit weights in comparison to those with arbitrary weights. In the former, the time limit was reached already for a graph with 15 vertices. In the latter, this occurrence starts for graphs with 30 vertices. Besides, in the second group, a higher percentage of instances was solved ( $70.32\% \times 65.68\%$ ) in a much smaller time ( $2.73s \times 26.36s$ ). It is worth remarking, however, that there were instances defined by the same parameters that EA could solve for unit weights but not for arbitrary weights, e.g. the instances with  $n = 60$ ,  $p = 1$  and  $t = \{2, 3, 4\}$ . Regarding the dilatation factor, its variation also influenced the performance of the algorithm. For both the weighted and unweighted cases, we observed that most of the instances where the algorithm timed out had  $t \in \{3, 4\}$ . For  $t = 2$  and, mainly,  $t = 1.1$ , it runs faster. Actually, for  $t \leq 2$  the problem is polynomial in the unweighted case (although EA does not explore this fact).

When compared to formulation CR, the algorithm solved a smaller number of instances in the time limit –  $65.68\% \times 76.57\%$  for unit weights, and  $70.72\% \times 92.19\%$  for arbitrary weights. On the other hand, the average time demanded by EA for the solved instances was considerably smaller than the formulation time –  $26.36s \times 74.92s$  for unit weights, and  $2.73s \times 22.14s$  for arbitrary

Table 1: Computational results for the studied instances

| Id           | n          | p   | t   | Unit Weights |           |         |           | Arbitrary Weights |           |         |           |
|--------------|------------|-----|-----|--------------|-----------|---------|-----------|-------------------|-----------|---------|-----------|
|              |            |     |     | Time CR      | Status CR | Time EA | Status EA | Time CR           | Status CR | Time EA | Status EA |
| 1            | 15         | 0,2 | 1,1 | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 2            | 15         | 0,2 | 2   | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 3            | 15         | 0,2 | 3   | 1            | inf       | 0       | inf       | 0                 | 45        | 0       | 45        |
| 4            | 15         | 0,2 | 4   | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 5            | 15         | 0,5 | 1,1 | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 6            | 15         | 0,5 | 2   | 0            | inf       | 1       | inf       | 0                 | inf       | 0       | inf       |
| 7            | 15         | 0,5 | 3   | 0            | inf       | 1105    | inf       | 0                 | 25        | 0       | 25        |
| 8            | 15         | 0,5 | 4   | 0            | 14        | 0       | 14        | 0                 | 19        | 0       | 19        |
| 9            | 15         | 0,8 | 1,1 | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 10           | 15         | 0,8 | 2   | 0            | 14        | 1200    | ?         | 0                 | inf       | 0       | inf       |
| 11           | 15         | 0,8 | 3   | 3            | 14        | 0       | 14        | 1                 | inf       | 0       | inf       |
| 12           | 15         | 0,8 | 4   | 0            | 14        | 0       | 14        | 0                 | 16        | 0       | 16        |
| 13           | 15         | 1   | 1,1 | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 14           | 15         | 1   | 2   | 1            | 14        | 0       | 14        | 0                 | inf       | 0       | inf       |
| 15           | 15         | 1   | 3   | 1            | 14        | 0       | 14        | 0                 | inf       | 3       | inf       |
| 16           | 15         | 1   | 4   | 0            | 14        | 0       | 14        | 0                 | 14        | 0       | 14        |
| 17           | 30         | 0,2 | 1,1 | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 18           | 30         | 0,2 | 2   | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 19           | 30         | 0,2 | 3   | 0            | inf       | 1200    | ?         | 0                 | inf       | 0       | inf       |
| 20           | 30         | 0,2 | 4   | 742          | inf       | 1200    | ?         | 0                 | 64        | 0       | 64        |
| 21           | 30         | 0,5 | 1,1 | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 22           | 30         | 0,5 | 2   | 1            | inf       | 1200    | ?         | 0                 | inf       | 0       | inf       |
| 23           | 30         | 0,5 | 3   | 872          | inf       | 1200    | ?         | 0                 | inf       | 1       | inf       |
| 24           | 30         | 0,5 | 4   | 58           | 29        | 0       | 29        | 2                 | inf       | 1200    | ?         |
| 25           | 30         | 0,8 | 1,1 | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 26           | 30         | 0,8 | 2   | 0            | inf       | 1200    | ?         | 0                 | inf       | 0       | inf       |
| 27           | 30         | 0,8 | 3   | 127          | 29        | 1200    | ?         | 1                 | inf       | 1200    | ?         |
| 28           | 30         | 0,8 | 4   | 84           | 29        | 0       | 29        | 82                | 31        | 1200    | ?         |
| 29           | 30         | 1   | 1,1 | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 30           | 30         | 1   | 2   | 8            | 29        | 0       | 29        | 0                 | inf       | 0       | inf       |
| 31           | 30         | 1   | 3   | 601          | 29        | 0       | 29        | 1                 | inf       | 1200    | ?         |
| 32           | 30         | 1   | 4   | 563          | 29        | 0       | 29        | 754               | 29        | 1200    | ?         |
| 33           | 45         | 0,2 | 1,1 | 0            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 34           | 45         | 0,2 | 2   | 0            | inf       | 1       | inf       | 0                 | inf       | 0       | inf       |
| 35           | 45         | 0,2 | 3   | 1            | inf       | 1200    | ?         | 0                 | inf       | 0       | inf       |
| 36           | 45         | 0,2 | 4   | 1200         | ?         | 1200    | ?         | 3                 | inf       | 5       | inf       |
| 37           | 45         | 0,5 | 1,1 | 0            | inf       | 0       | inf       | 0                 | inf       | 1       | inf       |
| 38           | 45         | 0,5 | 2   | 1            | inf       | 1200    | ?         | 0                 | inf       | 0       | inf       |
| 39           | 45         | 0,5 | 3   | 1200         | ?         | 1200    | ?         | 1                 | inf       | 1200    | ?         |
| 40           | 45         | 0,5 | 4   | 566          | 44        | 1200    | ?         | 2                 | inf       | 1200    | ?         |
| 41           | 45         | 0,8 | 1,1 | 1            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 42           | 45         | 0,8 | 2   | 1            | inf       | 1200    | ?         | 0                 | inf       | 11      | inf       |
| 43           | 45         | 0,8 | 3   | 1200         | ?         | 1200    | ?         | 3                 | inf       | 1200    | ?         |
| 44           | 45         | 0,8 | 4   | 1200         | ?         | 0       | 44        | 1201              | ?         | 1200    | ?         |
| 45           | 45         | 1   | 1,1 | 1            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 46           | 45         | 1   | 2   | 1200         | ?         | 0       | 44        | 0                 | inf       | 82      | inf       |
| 47           | 45         | 1   | 3   | 1200         | ?         | 0       | 44        | 29                | inf       | 1200    | ?         |
| 48           | 45         | 1   | 4   | 1201         | ?         | 0       | 44        | 1201              | ?         | 1200    | ?         |
| 49           | 60         | 0,2 | 1,1 | 1            | inf       | 0       | inf       | 0                 | inf       | 0       | inf       |
| 50           | 60         | 0,2 | 2   | 0            | inf       | 1200    | ?         | 1                 | inf       | 0       | inf       |
| 51           | 60         | 0,2 | 3   | 8            | inf       | 1200    | ?         | 0                 | inf       | 3       | inf       |
| 52           | 60         | 0,2 | 4   | 1200         | ?         | 1200    | ?         | 1                 | inf       | 1200    | ?         |
| 53           | 60         | 0,5 | 1,1 | 1            | inf       | 0       | inf       | 1                 | inf       | 0       | inf       |
| 54           | 60         | 0,5 | 2   | 2            | inf       | 1200    | ?         | 0                 | inf       | 0       | inf       |
| 55           | 60         | 0,5 | 3   | 1200         | ?         | 1200    | ?         | 2                 | inf       | 1200    | ?         |
| 56           | 60         | 0,5 | 4   | 1200         | ?         | 1200    | ?         | 1200              | ?         | 1200    | ?         |
| 57           | 60         | 0,8 | 1,1 | 1            | inf       | 0       | inf       | 1                 | inf       | 0       | inf       |
| 58           | 60         | 0,8 | 2   | 24           | inf       | 1200    | ?         | 0                 | inf       | 17      | inf       |
| 59           | 60         | 0,8 | 3   | 1201         | ?         | 1200    | ?         | 21                | inf       | 1200    | ?         |
| 60           | 60         | 0,8 | 4   | 1200         | ?         | 0       | 59        | 1200              | ?         | 1200    | ?         |
| 61           | 60         | 1   | 1,1 | 1            | inf       | 0       | inf       | 1                 | inf       | 0       | inf       |
| 62           | 60         | 1   | 2   | 1201         | ?         | 0       | 59        | 0                 | inf       | 1200    | ?         |
| 63           | 60         | 1   | 3   | 1201         | ?         | 0       | 59        | 399               | inf       | 1200    | ?         |
| 64           | 60         | 1   | 4   | 1201         | ?         | 0       | 59        | 1200              | ?         | 1200    | ?         |
| Solved       | Percentage |     |     | 76,57%       |           |         |           | 92,19%            |           |         |           |
| Average Time |            |     |     | 74,92        |           |         |           | 22,14             |           |         |           |
|              |            |     |     | 26,36        |           |         |           | 2,73              |           |         |           |

weights. When considering only the instances solved by both methods, the averages for CR drop to 39s and 0.18s, and EA still outperforms CR on unit weights. Also, for unit weights, some instances that CR did not solve were solved by EA (see instances {44, 46, 47, 48, 60, 62, 63, 64}).

## 5. Conclusion

In this work, we propose an enumerative algorithm for MWtSTP, the first one that does not depend on the resolution of mathematical programs. We observed that, on average, it performed better for instances with arbitrary weights. Besides, even for the cases with  $t = 2$  and unit weights, where the problem is polynomial, the algorithm reached the time limit for some instances.

Most of the tested instances where the algorithm timed out had  $t \in \{3, 4\}$ . For  $t \in \{1.1, 2\}$ , it runs faster. In part, it is due to the fact that, for  $t = 1.1$ , the detection of subproblem infeasibility occurs earlier, which makes the pruning procedure more effective. With the increase of  $t$ , the imminence of feasibility translates into an enlargement of the search space, and so more time for its exploration. When compared to a method in the literature, it proved to be more competitive for unweighted instances, regarding both the number of resolved instances and computational time.

As future work, there are at least two lines for research. The first one is to refine some steps of the algorithm. For the pruning procedure, the objective would be to detect infeasibility more quickly. For the branch, we could try another way of branching, if possible, by exploring a specific MWtSTP and tSTP property. The other line is to adapt the algorithm to help solving mathematical formulations in the literature [Braga, 2019; Sousa, 2021]. By improving the performance of the formulations, we would obtain a faster exact method.

## References

- Andrade, R. C. and Freitas, A. T. (2013). Disjunctive combinatorial branch in a subgradient tree algorithm for the dcmst problem with vns-lagrangian bounds. *Electronic Notes in Discrete Mathematics, Elsevier, [s.l.]*, 41:5 – 12.
- Arikati, S. R., Chen, D. Z., Chew, L. P., Das, G., Smid, M. H. M., and Zaroliagis, C. D. (1996). Planar spanners and approximate shortest path queries among obstacles in the plane. *Proceedings of the Fourth Annual European Symposium on Algorithms, ESA '96*, p. 514 – 528.
- Awerbuch, B. (1985). Complexity of network synchronization. *J. ACM*, 32, p. 802 – 823.
- Batsyn, M., Goldengorin, B., Kocheturov, A., and Pardalos, P. M. (2013). Tolerance-based vs. cost-based branching for the asymmetric capacitated vehicle routing problem. *Models, Algorithms, and Technologies for Network Analysis. [s.l.]: Springer*, p. 1–10.
- Braga, H. V. V. (2019). *Algoritmos exatos para problemas de spanner em grafos*. PhD thesis, Universidade de São Paulo.
- Brandstadt, A., Chepoi, V., and Dragan, F. (1999). Distance approximating trees for chordal and dually chordal graphs. *J. Algorithms*, 30(1), p. 166 – 184.
- Brandstadt, A., Dragan, F. F., Le, H.-O., and Le, V. B. (2004). Tree spanners on chordal graphs: complexity and algorithms. *Theor. Comput. Sci.*, 310(1-3), p. 329 – 354.
- Cai, L. and Coneil, D. G. (1995). Tree spanners. *Journal on Discrete Mathematics*, p. 359 – 387.
- Cai, L. (1992). *Tree spanners: spanning trees that approximate distances*. PhD thesis, University of Toronto.

- Dragan, F. F., Fomin, F. V., and Golovach, P. A. (2011). Spanners in sparse graphs. *J. Comput. Syst. Sci.*, 77(6), p. 1108 – 1119.
- Fekete, S. P. and Kremer, J. (2001). Tree spanners in planar graphs. *Discrete Applied Mathematics*, 108(1–2), p. 85 – 103.
- Goldschmidt, O., Takvorian, A., and Yu, G. (1996). On finding a biconnected spanning planar subgraph with applications to the facilities layout problem. *European Journal of Operational Research*, 94(1), p. 97 – 105.
- Kruskal, J. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, p. 48 – 51.
- Madanlal, M., Venkatesan, G., and Rangan, C. (1996). Tree 3-spanners on interval, permutation and regular bipartite graphs. *Information Processing Letters*, 59(2), p. 97 – 102.
- Marble, J. D. and Bekris, K. E. (2011a). Asymptotically near-optimal is good enough for motion planning. *Proc. of the 15th International Symposium on Robotics Research (ISRR-11)*.
- Marble, J. D. and Bekris, K. E. (2011b). Computing spanners of asymptotically optimal probabilistic roadmaps. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 4292 – 4298.
- Narayanaswamy, N. and Ramakrishna, G. (2015). Tree t-spanners in outerplanar graphs via supply demand partition. *Discrete Applied Mathematics*, 195, p. 104 – 109.
- Nguyen, C., Shen, J., Hou, M., Sheng, L., Miller, W., and Zhang, L. (2008). Approximating the spanning star forest problem and its application to genomic sequence alignment. *SIAM Journal on Computing*, p. 946 – 962.
- Oliveira, C. A. S. and Pardalos, P. M. (2005). A survey of combinatorial optimization problems in multicast routing. *Comput. Oper. Res.*, 32(8), p. 1953 – 1981.
- Papoutsakis, I. (2014). Tree 3-spanners of diameter at most 5. *CoRR*, abs/1402.3573.
- Papoutsakis, I. (2018). Tree spanners of bounded degree graphs. *Discrete Appl. Math.*, 236(C), p. 395 – 407.
- Peleg, D. and Reshef, E. (1999). A variant of the arrow distributed directory with low average complexity. *Automata, Languages and Programming*, p. 615 – 624.
- Peleg, D. and Ullman, J. D. (1989). An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4), p. 740 – 747.
- Peleg, D. and Upfal, E. (1988). A tradeoff between space and efficiency for routing tables. *Proceedings of the twentieth annual ACM symposium on Theory of computing*, p. 43 – 52.
- Sousa, G. H. (2021). O problema da árvore t-spanner de custo mínimo. Master's thesis, Universidade Federal do Ceará.
- Venkatesan, G., Rotics, U., Madanlal, M., Makowsky, J., and Rangan, C. P. (1997). Restrictions of minimum spanner problems. *Information and Computation*, 136(2), p. 143 – 164.