Efficient lower and upper bounds for the Weight-constrained Minimum Spanning Tree Problem using simple Lagrangian based algorithms *

Cristina Requejo

Department of Mathematics and CIDMA, University of Aveiro, Portugal.
crequejo@ua.pt

Eulália Santos

CIDMA and ISLA-Higher Institute of Leiria and Santaréam, Portugal.
eulalia.santos@sapo.pt

Abstract

The Weight-constrained Minimum Spanning Tree problem (WMST) is a combinatorial optimization problem for which simple but effective Lagrangian based algorithms have been used to compute lower and upper bounds. In this work we present several Lagrangian based algorithms for the WMST and propose two new algorithms, one incorporates cover inequalities. A uniform framework for deriving approximate solutions to the WMST is presented. We undertake an extensive computational experience comparing these Lagrangian based algorithms and show that these algorithms are fast and present small integrality gap values. The two proposed algorithms obtain good upper bounds and one of the proposed algorithms obtains the best lower bounds to the WMST.

***Keywords*:** Weighted Minimum spanning tree, Minimum spanning tree, Lagrangian based algorithms.

1 Introduction

Consider an undirected complete graph $G = (V, E)$, with node set $V = \{0, 1, \ldots, n-1\}$ and edge set $E = \{\{i, j\},\ i, j \in V, i \neq j\}$. Associated with each edge $e = \{i, j\} \in E$ consider positive integer costs c_e and weights w_e. The Weight-constrained Minimum Spanning Tree problem (WMST) is to find a spanning tree $T = (V, E_T)$ in G, $E_T \subset E$, of minimum cost $C(T) = \sum_{e \in E_T} c_e$ and with total weight $W(T) = \sum_{e \in E_T} w_e$ not exceeding a given limit W. An additional constraint to the Minimum Spanning Tree problem (MST) such as the weight constraint (the total tree weight $W(T)$ can not exceed a given limit W) turns this constrained MST into a NP-hard problem. The WMST is a NP-hard combinatorial optimization problem [1, 33].

The WMST appears in several real applications where the weight restrictions are mainly concerned with a limited budget on installation/upgrading costs. In this case, weights w_e represent

*This is a preprint of an article published in *Operational Research: An International Journal*. The final authenticated version is available online at: https://doi.org/10.1007/s12351-018-0426-x.

the installation/upgrading cost of the link $e = \{i, j\} \in E$ and c_e represent the link nominal cost/length. A classical application arises in the areas of communication networks and network design, in which information is broadcasted over a minimum spanning tree, and is related with the upgrade and/or design of the physical systems when there is a pre-specified budget restriction [16].

The WMST has received several different designations. It was first mentioned in Aggarwal, Aneja and Nair [1] as the *MST problem subject to a side constraint*, where MST stands for Minimal Spanning Tree. Besides the common designation as a WMST, the most common designation is as a *knapsack-constrained MST*. However there are some authors that refer to it as a *resource-constrained MST*.

Exact and approximation algorithms have already been proposed to the problem. Exact algorithms that use a Lagrangian relaxation to approximate a solution combined with a Branch and Bound strategy were proposed by Aggarwal, Aneja and Nair [1] and by Shogan [26]. Jörnsten and Migdalas [18] propose a Lagrangian Decomposition scheme in which, through duplication of variables, two subproblems, a MST and a Knapsack problem, have to be solved. Approximation schemes were proposed, by Ravi and Goemans [24] a polynomial-time approximation scheme, by Xue [31] a primal-dual algorithm, by Hong, Chung and Park [17] a bicriteria scheme and by Hassin and Levin [14] an improvement of the algorithm proposed by Ravi and Goemans [24]. A compilation of some results and existing algorithms to solve the problem can be found in Henn [16].

Requejo et al. [25] describe and compare, from the computational point of view, several Integer Linear Programming formulations for the WMST. Recently Agra et al. [2, 3] present valid inequalities for the WMST, the family of implicit cover inequalities is introduced and a lifting algorithm is discussed.

A common related approach is to include the weight of the tree as a second objective instead of a hard constraint. The resulting problem is the bicriteria/biobjective spanning tree problem (see [6, 11, 17, 23, 28, 29] among many others). In Aggarwal, Aneja and Nair [1] certain properties of an optimal solution are established considering a bicriteria spanning tree.

In this work we describe and compare, from the computational point of view, several Lagrangian based algorithms for the WMST. Similar Lagrangian based algorithms have been referred in several works of constrained shortest path problems [12, 19, 30] or of general combinatorial optimization problems [7, 21]. To the best of our knowledge, a computational study on the Lagrangian based algorithms for the WMST has never been published. Xue [31] describes a primal-dual algorithm to find approximate solutions for the WMST but no computational results are reported.

We present the following Lagrangian based algorithm approaches for deriving approximate solutions for the WMST: (i) algorithms based on approaches for the constrained shortest path problems and for general combinatorial optimization problems; (ii) the classical subgradient setting; and (iii) algorithms that use information about the shape of the dual Lagrangian function, two of these algorithms are new and one incorporates cover inequalities which is a novelty.

All the Lagrangian based algorithms considered solve only a MST as subproblem. This contrasts with other Lagrangian decomposition approaches (e.g. [18]) where both a MST and a Knapsack subproblems are solved.

We derive a uniform framework that standardises the several versions of the algorithms and further we undertake an extensive computational experience comparing the performance of the algorithms. This experience shows that the two proposed algorithms obtain good lower and upper bounds. Moreover, the new algorithm that uses cover inequalities obtains the best lower bounds.

We describe a general formulation for the problem in Section 2, discuss some properties of the problem in Section 3 and present the Lagrangian relaxation for the WMST in Section 4. In Section 5 we present a general framework, that uses different settings including the classical subgradient method to obtain approximate trees and propose two new settings. Computational results to assess the quality of the discussed procedures will be shown in Section 6. Finally, in Section 7 we present the conclusions.

2 A formulation for the WMST

To obtain formulations for the WMST one can adapt a MST formulation. For the MST several formulations are well known (see Magnanti and Wolsey [20]) and in Requejo et al. [25] natural and extended formulations for the WMST are discussed. For instance the two classical formulations for the MST, namely the formulation using cut-set inequalities and the formulation using circuit elimination inequalities, and the well-known compact extended multicommodity flow formulation using additional flow variables for the MST are easily adapted for the WMST through the inclusion of a weight constraint [25]. Other extended formulations, using e.g. Miller-Tucker-Zemlin (MTZ) inequalities to prevent the existence of circuits in the feasible solutions, can be derived [25].

Consider the canonical binary variables x_e (for all $e = \{i, j\} \in E$) indicating whether edge e is in the MST solution. A formulation for the WMST is as follows.

$$
\begin{aligned}
(WMST): \qquad \min \quad & \sum_{e \in E} c_e x_e \\
s.t. \quad & x \in (MST) \\
& \sum_{e \in E} w_e x_e \leq W \qquad (2.1)
\end{aligned}
$$

Where $x = (x_e) \in \mathbb{R}^{|E|}$ and (MST) represents a set of inequalities describing the convex hull of the (integer) solutions of the MST and can use one of the sets of inequalities referred previously (the circuit elimination inequalities, the cut-set inequalities, the multicommodity flow conservation constraints together with the connecting constraints) plus the following two sets of constraints: constraints $\sum_{e \in E} x_e = n - 1$, guaranteeing that the solution has $|V| - 1$ edges, and the variables integrality constraints $x_e \in \{0, 1\}$, for all $e \in E$. Constraint (2.1) is the weight constraint and we emphasize that the above formulation without the weight constraint is a formulation for the MST [20]. Let $\vartheta(WMST)$ be the optimal value of the WMST.

3 Some properties of the WMST

The well known Minimum Spanning Tree problem (MST) is to find a spanning tree $T_c = (V, E_{T_c})$, with $E_{T_c} \subset E$, of minimum cost $C(T_c) = \sum_{e \in E_{T_c}} c_e$. For this combinatorial optimization problem there are several polynomial algorithms such as Kruskal and Prim's algorithms

(see Ahuja et al. [4] for descriptions of these algorithms). Consider a companion problem, the Minimum-weight Spanning Tree problem that is to find a spanning tree $T_w = (V, E_{T_w})$, with $E_{T_w} \subset E$, of minimum weight $W(T_w) = \sum_{e \in E_{T_w}} w_e$. The trees T_c and T_w are two spanning trees of G, tree T_c is of minimum cost $C(T_c) = \sum_{e \in E_{T_c}} c_e$ having weight $W(T_c) = \sum_{e \in E_{T_c}} w_e$ and tree T_w is of minimum weight $W(T_w) = \sum_{e \in E_{T_w}} w_e$ having cost $C(T_w) = \sum_{e \in E_{T_w}} c_e$. The costs of these trees give lower and upper bounds to the optimal value of the problem

$$C(T_c) \leq \vartheta(WMST) \leq C(T_w).$$

The tree T_w corresponds to a feasible solution for the WMST and the tree T_c corresponds to an unfeasible solution when $W(T_c) > W$. We have the following propositions.

Proposition 1. *If $W \geq W(T_c)$ the WMST reduces to the MST and the tree T_c corresponds to the optimal solution.*

Proposition 2. *When $W < W(T_c)$, there exists an optimal solution for the WMST if and only if $W(T_w) \leq W$.*

If $W(T_w) > W$, then the WMST has no solution. If $W(T_w) \leq W$ and $C(T_w) = C(T_c)$, then the tree T_w corresponds to an optimal solution for the WMST.

In the case of $W(T_w) \leq W < W(T_c)$ and neither the tree T_c nor the tree T_w are optimal solutions for the WMST, we need to find another tree that is an optimal solution to the problem. To search for another tree, different from T_c and from T_w, we may use in the objective function different positive coefficients associated to each variable x_e corresponding to edge $e \in E$. Denote by p_e these new coefficients that are defined as a linear combination of the cost c_e and of the weight w_e associated to each edge $e \in E$. Thus $p_e = a\ w_e + b\ c_e$, where a and b are real non-negative scalars. With these coefficients in the objective function, find a new spanning tree $T_p = (V, E_{T_p})$ with $E_{T_p} \subset E$, of minimum value $P(T_p) = \sum_{e \in E_{T_p}} p_e$. The spanning tree T_p of G has cost $C(T_p) = \sum_{e \in E_{T_p}} c_e$ and weight $W(T_p) = \sum_{e \in E_{T_p}} w_e$. Notice that, the Minimum-cost Spanning Tree problem and the Minimum-weight Spanning Tree problem are particular cases of this Minimum Spanning Tree problem. If $a = 0$ and $b = 1$, then $T_p \equiv T_c$. If $a = 1$ and $b = 0$, then $T_p \equiv T_w$.

The tree T_p obtained with the coefficients p_e may be feasible or unfeasible. It holds $C(T_p) \geq C(T_c)$ and the following result.

Proposition 3. *Consider a tree T_p such that $C(T_c) \leq C(T_p) \leq C(T_w)$.*
If tree T_p is feasible, $W(T_p) \leq W$, then $C(T_p)$ is an upper bound to $\vartheta(WMST)$.
If tree T_p is unfeasible, $W(T_p) > W$, then $C(T_p)$ is a lower bound to $\vartheta(WMST)$.

Therefore, with tree T_p we may obtain a better upper bound or a better lower bound to $\vartheta(WMST)$, depending on the feasibility or unfeasibility of the tree T_p.

4 Lagrangian relaxation

In order to derive a Lagrangian relaxation, assign a non-negative Lagrangian multiplier λ to the weight constraint (2.1) and dualize the constraint in the usual Lagrangian way. This leads

to the following relaxed problem.

$$(WMST_\lambda): \quad -\lambda W \quad + \quad \min \quad \sum_{e\in E}(c_e+\lambda w_e)x_e$$
$$s.t. \quad x \in (MST)$$

For every $\lambda \geq 0$, the solutions to this relaxed problem give lower bounds on the optimum value, i.e.

$$\vartheta(WMST_\lambda) \leq \vartheta(WMST).$$

For a given non-negative value of λ, the relaxed problem $WMST_\lambda$ can be solved using any well known polynomial algorithm to solve the MST [4]. For each λ define positive coefficients $p_e^\lambda = \lambda w_e + c_e$ associated to each edge $e = \{i, j\} \in E$, this is, take $b = 1$ and $a = \lambda$. Let T_{p^λ} be the tree that corresponds to the solution for the problem $WMST_\lambda$, which is the minimum spanning tree obtained with the objective function coefficients defined by p_e^λ, therefore

$$\vartheta(WMST_\lambda) = -\lambda W + P(T_{p^\lambda}).$$

Notice that different values of λ may yield different trees T_{p^λ}, so that $\vartheta(WMST_\lambda)$ is a concave and piecewise linear function of λ. To obtain the best lower bound of the function $\vartheta(WMST_\lambda)$ we have to solve the following Dual Lagrangian problem

$$\vartheta^* = \vartheta(WMST_{\lambda^*}) := \max_{\lambda\geq 0} \vartheta(WMST_\lambda),$$

being λ^* the non-negative multiplier which maximizes $\vartheta(WMST_\lambda)$.

Classically a Lagrangian relaxation is solved using a subgradient optimization procedure [27]. The subgradient optimization procedure starts by initializing the Lagrangian multiplier λ to some value λ_0. After, iteratively at each iteration ($k = 0, 1, \ldots$), it solves the relaxed problem $WMST_{\lambda_k}$, and updates the Lagrangian multiplier λ_k by setting $\lambda_{k+1} = \max\{0, \lambda_k + s_k d_k\}$ using a direction d_k and a step-size s_k. Several directions d_k can be defined [15]. Together with an appropriate choice for the step size s_k [27] produces a convergent method. Finally some stopping criteria is verified.

For the solution $x^k = (x_e^k)$ of the Lagrangian relaxed problem $WMST_{\lambda_k}$, corresponding to the tree $T_{p^{\lambda_k}}$, we have $W(T_{p^{\lambda_k}}) = \sum_{e\in E} w_e x_e^k$, $C(T_{p^{\lambda_k}}) = \sum_{e\in E} c_e x_e^k$ and $\vartheta(WMST_{\lambda_k}) = -\lambda_k W + P(T_{p^{\lambda_k}}) = \lambda_k(W(T_{p^{\lambda_k}}) - W) + C(T_{p^{\lambda_k}})$. When the tree $T_{p^{\lambda_k}}$ is unfeasible, i.e. $W(T_{p^{\lambda_k}}) > W$, it is $\vartheta(WMST_{\lambda_k}) \geq C(T_{p^{\lambda_k}})$, thus $C(T_{p^{\lambda_k}})$ is a lower bound for $\vartheta(WMST_{\lambda_k})$. When the tree $T_{p^{\lambda_k}}$ is feasible, i.e. $W(T_{p^{\lambda_k}}) \leq W$, it is $\vartheta(WMST_{\lambda_k}) \leq C(T_{p^{\lambda_k}})$, hence $C(T_{p^{\lambda_k}})$ is an upper bound for $\vartheta(WMST_{\lambda_k})$.

For the Lagrangian multipliers λ_k, of the subgradient optimization procedure, the following upper bound can be established.

Proposition 4. *The Lagrangian multipliers λ_k satisfy $0 \leq \lambda_k \leq \frac{C(T_w)-C(T_c)}{W-W(T_w)}$.*

Proof. First notice that, by construction, $\lambda_k \geq 0$. Any feasible tree T is such that $C(T_c) \leq C(T) \leq C(T_w)$ and $W(T_w) \leq W(T) \leq W$. The line segment connecting the points $(W(T_w), C(T_w))$ and $(W(T_c), C(T_c))$ is represented by the equation $y = C(T_w) + m(x - W(T_w))$, with coordinates x representing the weights and coordinates y representing the costs, and has non-positive slope

$m = \frac{C(T_c)-C(T_w)}{W(T_c)-W(T_w)} = -\lambda_k \leq 0$. Any feasible tree T has weight $x \leq W$, thus it is such that $y = C(T_w) - \lambda_k(x - W(T_w)) \geq C(T_w) - \lambda_k(W - W(T_w))$. Therefore, any feasible tree T is such that its cost $C(T) \geq C(T_w) - \lambda_k(W - W(T_w))$. On the other hand, when $W(T_c) > W$, the tree T_c is unfeasible implying that $C(T_c) \leq C(T_w) - \lambda_k(W - W(T_w))$, and $\lambda_k \leq \frac{C(T_w)-C(T_c)}{W-W(T_w)}$. □

5 Solution procedure

In this section we propose a general framework for deriving approximate solutions for the WMST.

Iteratively several trees are generated. We start by generating the tree T_c, and the tree T_w. After, at each iteration, a different tree T_p is generated by using appropriate objective function coefficients, the coefficients $p_e = aw_e + bc_e$. For each $e \in E$, these objective function coefficients p_e are linear combination of the cost c_e and of the weight w_e. Different algorithms are obtained depending on the settings for parameters a and b.

The tree T_p can either be feasible or unfeasible. We keep track of the best obtained feasible tree with the tree T_{u_k}, and of the best obtained unfeasible tree with the tree T_{ℓ_k}. The values of the cost of these trees correspond to the best upper bound (UB) and to the best lower bound (LB), respectively. The tree T_{ℓ_k} is initialized with tree T_c, $T_{\ell_0} := T_c$. The tree T_{u_k} is initialized with tree T_w, $T_{u_0} := T_w$, if its weight is less than or equal to W. Otherwise there is no feasible solution. If tree T_p is feasible and its cost is less than or equal to the actual UB value, which is the cost of the current tree T_{u_k}, then the UB value is updated. Otherwise tree T_p is unfeasible and if its cost is greater than or equal to the actual LB value, the cost of the current tree T_{ℓ_k}, then the LB value is updated. If the Interval Reduction procedure returns two trees, both LB and UB values can be updated. Accordingly, the trees associated to the sequences of the UB and of the LB values are also updated. Parameters a and b depend from parameter λ_k and these parameters are iteratively updated according to the algorithm setting. In some algorithm settings the parameter λ_k belongs to the interval $[\ell_0, u_0]$ which is iteratively reduced according to the setting. Different coefficient settings yielding different Lagrangian dual variables and updating rules will be discussed. The general framework is as follows and the specific algorithm settings are displayed in Tables 1 and 2 and will be discussed next.

Algorithm General Framework

Input: Graph $G = (V, E)$, W, tol.

Step 1 - *Initializations.*

Iteration $k := 0$.

Step 1.1 - *Obtain a lower bound.*

Find $T_c = (V, E_{T_c})$ of minimum cost $C(T_c)$ and compute $W(T_c)$.

If $W(T_c) \leq W$ **then** T_c corresponds to an optimal solution. STOP.

Else set $T_{\ell_k} := T_c$, $W(T_{\ell_k}) := W(T_c)$, $LB := C(T_{\ell_k})$.

End-if.

Step 1.2 - *Obtain an upper bound.*

Find $T_w = (V, E_{T_w})$ of minimum weight $W(T_w)$ and compute $C(T_w)$.

If $W(T_w) > W$ **then** there is no solution. STOP.

Else set $T_{u_k} := T_w$, $W(T_{u_k}) := W(T_w)$, $P(T_{u_k}) := P(T_w)$, $UB := C(T_{u_k})$;

if an interval $[\ell_k, u_k]$ is used **then**

initialize parameters ℓ_k and u_k as specified by the algorithm setting;

end-if;

initialize parameters λ_k and ν_k as specified by the algorithm setting.

End-if.

Step 2 - *Compute an approximate tree.*

Obtain the parameters a and b according to the algorithm setting.

Compute $p_e = aw_e + bc_e$, $\forall e \in E_{T_c}$.

Find $T_p = (V, E_{T_p})$ of minimum $P(T_p)$, compute $C(T_p)$ and $W(T_p)$.

If an interval $[\ell_k, u_k]$ is used **then**

use **Interval Reduction** $[\ell_k, u_k]$ according to the algorithm setting.

End-if.

Step 3 - *Update trees and bounds.*

If the output of Step 2 is tree T_p **then**

If $W(T_p) \leq W$ **then**

if $C(T_p) \leq C(T_{u_k})$ then $UB := C(T_p)$; $T_{u_{k+1}} := T_p$; $T_{\ell_{k+1}} := T_{\ell_k}$.

Else if $C(T_p) \geq C(T_{\ell_k})$ then $LB := C(T_p)$; $T_{\ell_{k+1}} := T_p$; $T_{u_{k+1}} := T_{u_k}$.

End-if

Else (the output of Step 2 is tree T_{p_1} and tree T_{p_2})

If $C(T_{p_1}) \geq C(T_{\ell_k})$ and $C(T_{p_2}) \leq C(T_{u_k})$ **then**

$LB := C(T_{p_1})$; $UB := C(T_{p_2})$; $T_{\ell_{k+1}} := T_{p_1}$; $T_{u_{k+1}} := T_{p_2}$.

Else

If $C(T_{p_1}) \geq C(T_{\ell_k})$ **then** $LB := C(T_{p_1})$; $T_{\ell_{k+1}} := T_{p_1}$; $T_{u_{k+1}} := T_{u_k}$.

If $C(T_{p_2}) \leq C(T_{u_k})$ **then** $UB := C(T_{p_2})$; $T_{\ell_{k+1}} := T_{\ell_k}$; $T_{u_{k+1}} := T_{p_2}$.

End-if

End-if.

Step 4 - *Stopping criteria.*

If $|P(T_{u_k}) - P(T_p)| \leq tol$ **or** $(u_{k+1} - \ell_{k+1}) \leq tol$ **then** tree $T_{u_{k+1}}$ corresponds to the approximate solution, STOP.

Else Iteration $k := k + 1$ and go to **Step 2**.

End-if.

Table 1 specifies for each algorithm setting the initialization of the parameters ℓ_0, λ_0, ν_0 and u_0; the definition of the parameters a and b; and the updating rules. For some algorithm settings an interval reduction is used. The details of these interval reduction procedures are given in Table 2. In Step 4 the tolerance *tol* is a small real positive value given as an input of the algorithm.

To evaluate the complexity of this algorithm one has to take into account the complexity of the algorithm used to build the trees, consider that such algorithm has complexity of $\varphi(m, n)$ as it depends on the number $m = |E|$ of edges and on the number $n = |V|$ of nodes of the graph G. One also has to take into account the number of trees formed. If K is the total number of trees that can be formed, then this algorithm stops after $\mathcal{O}(logK)$ iterations (in the worst case, the number of trees is proportional to K). The main effort of the algorithm is on the obtention of a tree. Consequently, the complexity of this algorithm is $\mathcal{O}(\varphi(m, n)\ logK)$.

Different algorithms are obtained depending on the settings for parameters a and b in the definition of the new coefficients in Step 2 and the interval reduction when applied. Notice that the subgradient optimization scheme perfectly fits this algorithm layout and is one of the settings discussed bellow. Next we will discuss different settings for the positive coefficients $p_e = aw_e + bc_e$, with non-negative real scalars a and b, associated to each edge $e \in E_{T_c}$ and their update at each iteration.

First consider a setting for the coefficients p_e characterized by a convex linear combination of costs c_e and of weights w_e, more precisely associate parameter $a = 1 - \lambda_k$ for the weights and parameter $b = \lambda_k \in [0, 1]$ for the costs. These settings were proposed in Xue [31] for the weight-constrained shortest path problem. It is proposed that $\lambda_k = \frac{W(T_{\ell_k}) - W(T_{u_k})}{C(T_{u_k}) - C(T_{\ell_k}) + W(T_{\ell_k}) - W(T_{u_k})} \leq 1$ on an odd iteration and $\lambda_k = \frac{\ell_k + u_k}{2} \leq 1$ on an even iteration, with $\ell_k, u_k \in [0, 1]$ and initializing $\ell_0 = 0$ and $u_0 = 1$, see Table 1. An interval reduction is applied as displayed in Table 2. This setting is referred as Alg1.

Now consider settings for the coefficients p_e characterized by associating a parameter, the Lagrangian multiplier, for the weights, $a = \lambda_k$, and a parameter with value equal to one for the costs, $b = 1$. Some examples of such settings will be given next.

Jüttner et al. [19] built up the LAgrangian Relaxation based Aggregated Cost (LARAC) algorithm which solves the Lagrangian relaxation of the constrained shortest path problem. Xiao et al. [30] establish the equivalence between the LARAC algorithm and other algorithms in [7, 12, 19]. Afterwards Mehlhorn and Ziegelmann [21] also consider this algorithm and Xue [32] presents a variant of the algorithm. Using the ideas of these algorithms, consider the setting $a = \lambda_k = \frac{C(T_{u_k}) - C(T_{\ell_k})}{W(T_{\ell_k}) - W(T_{u_k})}$ and $b = 1$, see Table 1. This setting is referred as Alg2.

The settings for the classical subgradient algorithm consider a direction d_k and an appropriate step-size s_k to update the Lagrange multiplier at each iteration of the algorithm. If the Held, Wolfe and Crowder [15] setting for a direction is considered, then the direction is $d_k = \sum_{e \in E_{T_c}} w_e x_e^k - W = W(T_{p^{\lambda_k}}) - W$. Using, as suggested in Shor [27], the step-size

$$s_k = \rho \frac{C(T_{u_k}) - \vartheta(WMST_{\lambda_k})}{(\sum_{e \in E_{T_c}} w_e x_e^k - W) d_k} = \rho \frac{C(T_{u_k}) - P(T_{p^{\lambda_k}}) + \lambda_k W}{(W(T_{p^{\lambda_k}}) - W) d_k}$$

Table 1: Algorithms settings.

Algorithm	Initialization ℓ_0, λ_0, u_0	Parameters a	b	Updating λ_k	Ref.
Alg1	$\ell_0 = 0 \quad \lambda_0 = \frac{W(T_c)-W(T_w)}{C(T_w)-C(T_c)+W(T_c)-W(T_w)} \quad u_0 = 1$	$a = 1-\lambda_k$	$b = \lambda_k$	$\lambda_k = \frac{W(T_{\ell_k})-W(T_{u_k})}{C(T_{u_k})-C(T_{\ell_k})+W(T_{\ell_k})-W(T_{u_k})}$, k odd $\lambda_k = \frac{\ell_k+u_k}{2}$, k even	[31]
Alg2	$\lambda_0 = \frac{C(T_w)-C(T_c)}{W(T_c)-W(T_w)}$	$a = \lambda_k$	$b = 1$	$\lambda_k = \frac{C(T_{u_k})-C(T_{\ell_k})}{W(T_{\ell_k})-W(T_{u_k})}$	[7, 12, 19, 30]
Alg3	$\lambda_0 = \frac{C(T_w)-C(T_c)}{W(T_c)-W}$	$a = \lambda_k$	$b = 1$	$\lambda_{k+1} = \max\left\{0, \lambda_k + \rho\frac{C(T_{u_k})-P(T_{p^{\lambda_k}})+\lambda_k W}{W(T_{p^{\lambda_k}})-W}\right\}$	[15, 27]
Alg4	$\ell_0 = 0 \quad \lambda_0 = u_0 \quad u_0 = \frac{C(T_w)-C(T_c)}{W-W(T_w)}$	$a = \lambda_k$	$b = 1$	$\lambda_k = \frac{\ell_k+u_k}{2}$	[30]
Alg5	$\ell_0 = 0 \quad \lambda_0 = u_0 \quad u_0 = \frac{C(T_w)-C(T_c)}{W-W(T_w)}$	$a = \lambda_k$	$b = 1$	$\lambda_k = \frac{\ell_k+u_k}{2}$	[5]
Alg6	$\ell_0 = 0 \quad \lambda_0 = u_0 \quad u_0 = \frac{C(T_w)-C(T_c)}{W-W(T_w)}$	$a = \lambda_k$	$b = 1$	$\lambda_k = \frac{\ell_k+u_k}{2}$	
Alg7	$\ell_0 = 0 \quad \lambda_0 = u_0 \quad u_0 = \frac{C(T_w)-C(T_c)}{W-W(T_w)}$ $\nu_0 = tol$	$a = \lambda_k$	$b = 1$	$\lambda_k = \frac{\ell_k+u_k}{2}$ $\nu_k = \frac{(C(T_{u_k})-C(T_{\ell_k}))(W-W(T_{u_k}))}{\lvert\mathcal{C}_k\rvert(W-W(T_{u_k}))-W(T_{u_k})+W(T_{\ell_k})}$	

Table 2: Settings for the interval reduction.

Algorithm	**Interval Reduction** $[\ell_k, u_k]$
Alg1 and Alg4	**If** $W(T_p) \le W$ **then** set $\ell_{k+1} := \lambda_k$ and $u_{k+1} := u_k$. **Else** set $\ell_{k+1} := \ell_k$ and $u_{k+1} := \lambda_k$.

Alg5	**If** $\vartheta(\lambda_k) \ge \max\{\vartheta(u_k), \vartheta(\ell_k)\}$ **then**
	set $\lambda_k^a = \frac{\ell_k+\lambda_k}{2}$ and $\lambda_k^b = \frac{\lambda_k+u_k}{2}$;
	obtain $\vartheta(\lambda_k^a)$ and $\vartheta(\lambda_k^b)$;
	if $\vartheta(\lambda_k^a) \ge \vartheta(\lambda_k)$ **then**
	set $\ell_{k+1} := \ell_k$ and $u_{k+1} := \lambda_k$;
	set $T_p := T_{p^{\lambda_k^a}}$;
	else-if $\vartheta(\lambda_k^b) \ge \vartheta(\lambda_k)$ **then**
	set $\ell_{k+1} := \lambda_k$ and $u_{k+1} := u_k$;
	set $T_p := T_{p^{\lambda_k^b}}$;
	else set $\ell_{k+1} := \lambda_k^a$ and $u_{k+1} := \lambda_k^b$;
	set $T_{p_1} := T_{p^{\lambda_k^a}}$ and $T_{p_2} := T_{p^{\lambda_k^b}}$;
	end-if.
	Else-if $\vartheta(\ell_k) \le \vartheta(\lambda_k) \le \vartheta(u_k)$ **then** set $\ell_{k+1} := \lambda_k$ and $u_{k+1} := u_k$.
	Else set $\ell_{k+1} := \ell_k$ and $u_{k+1} := \lambda_k$.
	End-if.

Alg6 and Alg7	**If** $(W(T_p) - W)(W(T_{\ell_k}) - W) > 0$ **then** set $\ell_{k+1} := \lambda_k$ and $u_{k+1} := u_k$. **Else-if** $(W(T_p) - W)(W(T_{u_k}) - W) > 0$ **then** set $\ell_{k+1} := \ell_k$ and $u_{k+1} := \lambda_k$. **Else** set $\ell_{k+1} := \frac{\ell_k+\lambda_k}{2}$ and $u_{k+1} := \frac{\lambda_k+u_k}{2}$. set $T_{p_1} := T_{\ell_k}$ and $T_{p_2} := T_{u_k}$. **End-if**.

with $\rho \in]0,2[$ and the upper bound $C(T_{u_k})$ to approximate the optimum value of the problem, leads to the setting

$$a = \lambda_{k+1} = \max\left\{0, \lambda_k + \rho \frac{C(T_{u_k}) - P(T_{p^{\lambda_k}}) + \lambda_k W}{W(T_{p^{\lambda_k}}) - W}\right\}$$

and $b = 1$. Initialize $\lambda_0 = \frac{C(T_w)-C(T_c)}{W(T_c)-W}$. The parameter $\rho \in]0,2[$ needs to be carefully chosen, this tuning process is a drawback of this setting. This setting corresponds to the classical subgradient optimization procedure to a Lagrangian relaxation, see Table 1, and is referred as Alg3.

In Xiao et al. [30] another algorithm is proposed to solve the Lagrangian relaxation of the constrained shortest path problem. This algorithm uses a binary search to iteratively reduce the interval $[\ell_k, u_k]$ and obtain the tree that corresponds to the approximate solution. This setting for the approximate tree coefficients is $a = \lambda_k = \frac{\ell_k+u_k}{2}$ and $b = 1$. Initialize $\ell_0 = 0$ and $u_0 = \frac{C(T_w)-C(T_c)}{W-W(T_w)}$, see Table 1, and update the interval extremes ℓ_k and u_k as displayed in Table 2. This setting is referred as Alg4.

Amado and Barcia [5] propose an algorithm to solve several matroidal knapsacks such as the Multiple Choice Knapsack Problem and the WMST. Their algorithm initializes the interval $[\ell_0, u_0]$ with $\ell_0 = 0$ and $u_0 = U$, with $U := \max_{e \in E_{T_c}}\{c_e, w_e\}$, iteratively updates the reduced interval $[\ell_k, u_k]$ until a stopping criteria is satisfied and the tree that corresponds to the approximate solution is identified. By setting $a = \lambda_k = \frac{\ell_k+u_k}{2}$ and $b = 1$ the reduced interval is obtained by comparing the values of $\vartheta(WMST_{\ell_k})$, $\vartheta(WMST_{\lambda_k})$ and $\vartheta(WMST_{u_k})$. To ease notation use $\vartheta(\ell_k)$ instead of $\vartheta(WMST_{\ell_k})$, $\vartheta(\lambda_k)$ instead of $\vartheta(WMST_{\lambda_k})$, and $\vartheta(u_k)$ instead of $\vartheta(WMST_{u_k})$. Initially $\vartheta(\ell_0) = C(T_{\ell_0}) = C(T_c)$ and $\vartheta(u_0)$ is obtained using the tree T_{u_0} of minimum $P(T_{u_0})$, with $p_e = u_0 w_e + c_e$. Notice that $\vartheta(\lambda_k) = \vartheta(WMST_{\lambda_k}) = -\lambda_k W + P(T_{p^{\lambda_k}}) = -\lambda_k W + C(T_{p^{\lambda_k}}) + \lambda_k W(T_{p^{\lambda_k}})$. The tree T_p corresponds to tree $T_{p^{\lambda_k}}$. In the interval reduction procedure to obtain values $\vartheta(\lambda_k^a)$ and $\vartheta(\lambda_k^b)$ two more trees are obtained, tree $T_{p^{\lambda_k^a}}$ and tree $T_{p^{\lambda_k^b}}$. The tree T_p is updated to the best tree chosen among the three trees obtained to reduce the interval. The interval reduction and the update of the tree T_p are displayed in Table 2. Using the Proposition 4, here the initialization of the interval $[\ell_0, u_0]$ is done differently from [5] with $\ell_0 = 0$ and $u_0 = \frac{C(T_w)-C(T_c)}{W-W(T_w)}$. This setting is referred as Alg5.

As the Lagrangian function is concave, the interval extremes can be updated according to the gradient $W - W(T_{p^{\lambda_k}})$ of the Lagrangian function at λ_k and at the interval extremes ℓ_k and u_k. If both slopes at λ_k and at one of the extremes have the same inclination, that extreme can be updated to λ_k. Otherwise, both extremes (lower and upper) can be updated. Therefore, the interval reduction is simplified and the previous procedure for the interval reduction can be modified. This setting we propose is referred as Alg6 and the new interval reduction procedure is displayed in Table 2.

Following Amado and Barcia [5] another Lagrangian based algorithm to solve several matroidal knapsacks can be used. This algorithm is obtained by dualizing not only the weight inequality (2.1) but also an extra valid inequality, inequality (5.1), added to the model as it

follows.

$$
\begin{aligned}
(WMST^{+}): \qquad \min \quad & \sum_{e\in E_{T_c}} c_e x_e \\
s.t. \quad & x \in (MST) \\
& \sum_{e\in E_{T_c}} w_e x_e \leq W \\
& \sum_{e\in\mathcal{C}} x_e \leq |\mathcal{C}| - 1 \qquad (5.1)
\end{aligned}
$$

The valid inequality (5.1) added to the model is a cover inequality [3]. A set $\mathcal{C} \subseteq E$ is called a *cover* if $\sum_{e\in\mathcal{C}} w_e > W$. Given a cover $\mathcal{C}$ the *cover inequality* $\sum_{e\in\mathcal{C}} x_e \leq |\mathcal{C}|-1$ is valid for the WMST [3]. Any other family of valid inequalities can be used, one example is the family of implicit cover inequalities proposed by Agra et al. [3]. By adding a valid inequality to the model and dualizing both constraints (2.1) and (5.1) in the Lagrangian way, a Lagrangian function with a better lower bound can be obtained. This is $\vartheta(WMST_\lambda) \leq \vartheta(WMST_{\lambda,\nu}) \leq \vartheta(WMST)$, for non-negative Lagrangian multipliers λ and ν, with λ the non-negative Lagrangian multiplier associated to inequality (2.1) and ν the non-negative Lagrangian multiplier associated to inequality (5.1). This setting for the approximate tree coefficients p_e is $a = \lambda$ and $b = 1$ with $p_e := aw_e + bc_e$, for all $e \notin \mathcal{C}$ and $p_e := aw_e + bc_e + \nu$, for all $e \in \mathcal{C}$. For the solution $x = (x_e)$ of the Lagrangian relaxed problem $WMST_{\lambda,\nu}$, corresponding to the tree $T_{p^{\lambda,\nu}}$, define $W(T_{p^{\lambda,\nu}}) = \sum_{e\in E} w_e x_e$, $C(T_{p^{\lambda,\nu}}) = \sum_{e\in E} c_e x_e$ and $P(T_{p^{\lambda,\nu}}) = \lambda W(T_{p^{\lambda,\nu}}) + C(T_{p^{\lambda,\nu}}) + \nu \sum_{e\in\mathcal{C}} x_e$. Therefore

$$\vartheta(WMST_{\lambda,\nu}) = -\lambda W - \nu(|\mathcal{C}| - 1) + P(T_{p^{\lambda,\nu}}).$$

Several cover inequalities will be dynamically added to the model. At each iteration, whenever an unfeasible tree is obtained one cover inequality is constructed and added, by dualization, to the model. The proposed algorithm to solve this problem by Amado and Barcia [5] is the two dimension version to the one previously presented. This setting is referred as Alg7, see Tables 1 and 2 for details. The algorithm initializes the interval $[\ell_0, u_0]$. Additionally the multiplier ν must also be initialized in Step 1.2 to a small value, we may use $\nu_0 = tol$. Iteratively the algorithm builds an approximate tree T_p. Notice that this tree T_p uses multiplier λ_k and multiplier ν_k. Thus the tree T_p corresponds to tree $T_{p^{\lambda_k,\nu_k}}$. Whenever this tree T_p is unfeasible, a cover inequality is constructed using a separation algorithm to identify a valid cover $\mathcal{C}_k$ [3]. The corresponding cover inequality is added to the problem associated with multiplier ν_k. When reducing the interval $[\ell_k, u_k]$ for the multiplier λ_k using the previous simplified interval reduction procedure (the same as Alg6), the value of the multiplier ν_k must be updated. Considering $|\mathcal{C}_k| = \sum_{e\in\mathcal{C}_k} x_e$, the multiplier ν_k is updated to the following setting

$$\nu_k = \frac{(C(T_{u_k}) - C(T_{\ell_k}))(W - W(T_{u_k}))}{|\mathcal{C}_k|}(W - W(T_{u_k})) - W(T_{u_k}) + W(T_{\ell_k}).$$

This algorithm we propose, referred as Alg7, obtains good upper bounds and the best lower bounds to the WMST.

The simplicity of all these procedures has its price as it depends greatly on the ability of each algorithm setting to find near optimal multipliers quickly and specific to each instance. As we will observe in the next section different gaps are reported for the same problem instance depending on the specificity of the overall algorithm settings and its ability in finding approximate solutions.

6 Computational experience

Computational results will assess the quality of the approximate solutions obtained with each setting and the corresponding updating process of the Lagrangian scheme presented in Section 5. The algorithms from Section 5 were implemented in C++ and all the tests were performed in an Intel(R) Core(TM)2 Duo CPU (T7100) 2.00 GHz processor and 4Gb of RAM. We present computational results for instances to the WMST defined on complete graphs with a number of nodes varying between 10 and 1000, in a total of 215 instances.

To generate an instance of the WMST, two values to associate to each edge e, a cost c_e and a weight w_e, have to be defined. Afterwards, a (feasible) value to the weight limit W must also be defined. We built three sets of instances, constituting three different ways of generating costs and weights.

In a first set of instances, costs c_e and weights w_e are generated similarly to a set of instances described in Pisinger [22] and named therein as Spanner instances. We use the following values for W: $W = 1000$ for all instances with $n \leq 100$, $W = 1500$ for all instances with $n = 150, 200$, $W = 2000$ for all instances with $n = 300$, $W = 2500$ for all instances with $n = 400$, $W = 3000$ for all instances with $n = 500$ and $W = 3500$ for all instances with $n = 1000$. The costs and the weights are multiples of a small set (the so-called spanner set in [22]) of costs and weights following one particular distribution, we use the Uncorrelated distribution (which is in the Pisinger's [22] proposed list of distributions), and the following two parameters s and m, such that s is the size of the small set and m is the multiplier limit. Pisinger [22] proposes $s = 2$ and $m = 10$. The small set of s pairs of costs and weights is constructed by randomly selecting $\widetilde{w}_j$ and $\widetilde{c}_j$ both in $[1, 100]$, for $j \in \{1, \ldots, s\}$. Then the s costs and weights in the small set are normalized by dividing them by $m + 1$. After the costs c_e and weights w_e are constructed by (1) randomly selecting a pair of costs and weights $(\widetilde{c}_k, \widetilde{w}_k)$, $k \in \{1, \ldots, s\}$, from the normalized small set, (2) randomly selecting a multiplier a in $[1,\ m]$, (3) setting $w_e = a\ \widetilde{w}_k$, (4) setting $c_e = a\ \widetilde{c}_k$. Finally the weights of some edges are manipulated in such a way that the optimal solution has a desired predefined structure. After testing a few structures we obtained some challenging instances when the optimal structure of the WMST instance solution has large diameter values, almost $n - 1$, but not equal to $n - 1$, in such way that the tree is almost a path. Thus we name this instances set as "Almost Path" (AP). To obtain such structured instances, they are generated in such a way that the optimal solution is a graph with very large diameter, but not diameter equal to $n-1$, as follows. (i) Obtain the minimal spanning tree. (ii) Assign big weight values to the edges in the minimal spanning tree. We use $w_e = (W/n) \times a/100$, with $a \in [50,\ 90]$. (iii) For the remaining edges, assign the value $w_e = round(r + W(1-p)/(n-1))$ to their weight, with some $p \in [0.5,\ 1]$ and r randomly selected in the interval $[1,\ W \times p/(n-1)]$. If $w_e = 1$, then assign the value $w_e = r \times a_1 + W(1 - p \times a_2)/(n-1)$, with a_1 and a_2 selected in the interval $[0,\ 10]$.

For the second set of instances, named Random (R) instances, the costs c_e and the weights w_e are uniformly generated in the interval $[1, 1000]$.

For the third set of instances, named Euclidean (E) instances, costs c_e and weights w_e are obtained using Euclidean distances. After randomly generating the coordinates of n points/nodes in a 100×100 grid, the cost c_e of each edge $e = \{i, j\}$ is the integer part of the Euclidean distance between points/nodes i and j. We proceed independently and similarly to obtain the weights.

To define a (feasible) value to the weight limit W for each instance of these two sets of instances (sets R and E), we start by obtaining the weight of the minimum spanning tree $W(T_c)$ and the weight of the minimum weight spanning tree $W(T_w)$ and we select W to be one of the values $W_i = \frac{W(T_c)+W(T_w)}{2^i}$, for $i \in \{1, \ldots, 10\}$.

A total of 215 instances were generated, 95 of the set AP and 60 of each set R and E. For each set AP and each instance size between 10 and 150 we have 10 instances and for each instance size between 200 and 1000 we have 5 instances. For each set R and E and each instance size we have 5 instances.

To use the approximation schemes from Section 5, some parameters were defined as follows. After testing values within the interval $[0.0001, 0.1]$ we fixed $tol = 0.001$. In Alg3 we tested several values for $\rho \in]0, 2[$. For the AP instances we fixed $\rho = 0.001$. For the R instances this value was fixed in 0.001, 0.0065, 0.075, 0.25, 0.095, 0.125, 0.085, 0.06, 0.045, 0.04, 0.04 and 0.03, depending on the number of nodes. For the E instances this value was fixed in 0.0001 for $n = 10, 20, 40, 60, 80$, in 0.04 for $n = 100, 150, 200, 300, 400$, in 0.035 for $n = 500$ and in 0.015 for $n = 1000$. In Alg5 the interval upper bound u_0 must be initialized. We tested $u_0 = \max_{e \in E_{T_c}} \{c_e, w_e\}$ and $u_0 = \frac{C(T_w)-C(T_c)}{W-W(T_w)}$. Better results were obtained with the second initialization. In Alg7, for the construction of the cover inequality, after testing three ordering schemes ((i) increasing order of the edges costs c_e, (ii) decreasing order of the edges weights w_e, (iii) decreasing order of the quotient c_e/w_e) for selecting the edges of the tree T_{ℓ_k} to be in the cover, we noticed that better results are obtained with ordering scheme (ii).

In [18] a Lagrangian decomposition procedure is proposed that separates the WMST problem into two subproblems, a MST and a Knapsack problem. At each iteration both a MST and a Knapsack subproblems have to be solved. In the algorithms described in Section 5 only a MST subproblem has to be solved. Although the reported theoretical bound in [18] is superior, the performance of this decomposition approach depends greatly on the ability to find near optimal multipliers quickly and specific to each instance. Additionaly, when using this decomposition, the number of parameters to tune is larger. We were not able to obtain interesting computational results using this decomposition on our instances and the values obtained are far from the theoretical ones. Therefore we will not report computational results using this Lagrangian decomposition.

In [2, 25] the best results to obtain the optimal value using the software Xpress 7.3 (Xpress Release 2012 with Xpress-Optimizer 23.01.03 and Xpress-Mosel 3.4.0) [9], were obtained with the Branch and Cut algorithm based on a weighted MTZ (Miller-Tucker-Zemlin) formulation with the inclusion of cuts preventing cycles at the root node. This procedure will be denoted by HP (Hybrid Procedure) and is used to access the quality of the approximate solutions obtained with the Lagrangian based algorithms from Section 5. To compare the performance of those algorithms with the performance of the HP, two gaps are calculated, the upper bound gap, gap_U, and the lower bound gap, gap_L. Denote with OPT the optimal value obtained with the HP or the best obtained value with this procedure within a time limit of 10000 seconds. Denote with U_L the upper bound and with L_L the lower bound obtained with a Lagrangian based approximation scheme. The upper bound gap is $gap_U = \frac{U_L-OPT}{OPT} \times 100$, and the lower bound gap is $gap_L = \frac{OPT-L_L}{OPT} \times 100$, which is the Lagrangian relaxation bound.

Table 3: Percentage of instances with $gap_U = 0$, for each instance set AP, R and E and, in the last line, for all instances.

	Alg1	Alg2	Alg3	Alg4	Alg5	Alg6	Alg7
AP	6.32	35.79	16.84	6.32	35.79	35.79	37.89
R	5.00	6.67	1.67	5.00	6.67	6.67	5.00
E	25.00	30.00	11.67	23.33	30.00	30.00	28.33
$\%(gap_U = 0)$	11.16	26.05	11.16	10.70	26.05	26.05	26.05

Table 4: Percentage of instances with $gap_U < gap_L$, for each instance set AP, R and E and, in the last line, for all instances.

	Alg1	Alg2	Alg3	Alg4	Alg5	Alg6	Alg7
AP	54.74	75.79	67.37	97.89	75.79	75.79	76.84
R	16.67	20.00	11.67	13.33	20.00	20.00	18.33
E	38.33	40.00	25.00	31.67	40.00	40.00	35.00
$\%(gap_U < gap_L)$	39.53	50.23	40.00	55.81	50.23	50.23	48.84

Table 3 presents, for each algorithm, the percentage of instances having $gap_U = 0$, for each instance set AP, R and E and, in the last line of the table, the percentage for all instances. Generally, algorithms Alg2, Alg5, Alg6 and Alg7 obtain the higher percentage of null upper bound gap, gap_U, each with the same value of 26.05% (56 instances out of 215). Instances AP and E obtain the higher percentage of null gap_U.

Table 4 presents, for each algorithm, the percentage of instances having the gap_U less than the lower bound gap gap_L, for each instance set AP, R and E and, in the last line, for all instances. This indicates when the gap_U is closer to the optimum value than the gap_L. For the AP instances the upper bound value is closer to the optimum value than the lower bound value. For R and E instances the lower bound, the Lagrangian bound, is closer to the optimal value than the upper bound value obtained using the Lagrangian scheme.

In Figure 1 we compare the mean computational times (in seconds) between all the algorithms for all the instances sets. All the algorithms are fast in obtaining an approximate solution. Clearly Alg3, the classical subgradient algorithm for the Lagrangian relaxation, is more frequently the most time consuming, followed by Alg5.

Algorithms Alg5 and Alg6 are very similar and differ on the interval reduction procedure having, as a consequence, different number of calculated trees. In both algorithms the initialization used is $u_0 = \frac{C(T_w)-C(T_c)}{W-W(T_w)}$. In Figure 2 we compare the mean number of trees that each algorithm Alg5 and Alg6 has to build during its execution. Alg5 builds more trees than Alg6 and this may explain the execution time difference between both algorithms and why Alg5 is much more time consuming than Alg6.

It is worth to mention that in order to try to reduce the number of trees computed in Alg5 and in Alg6 we tested a Fibonacci search [13] for the interval reduction procedure. In the following Fibonacci search interval reduction procedure consider that $F_0 = F_1 = 1$ and $F_n = F_{n-2} + F_{n-1}$, $n \geq 2$, denote the Fibonacci numbers.

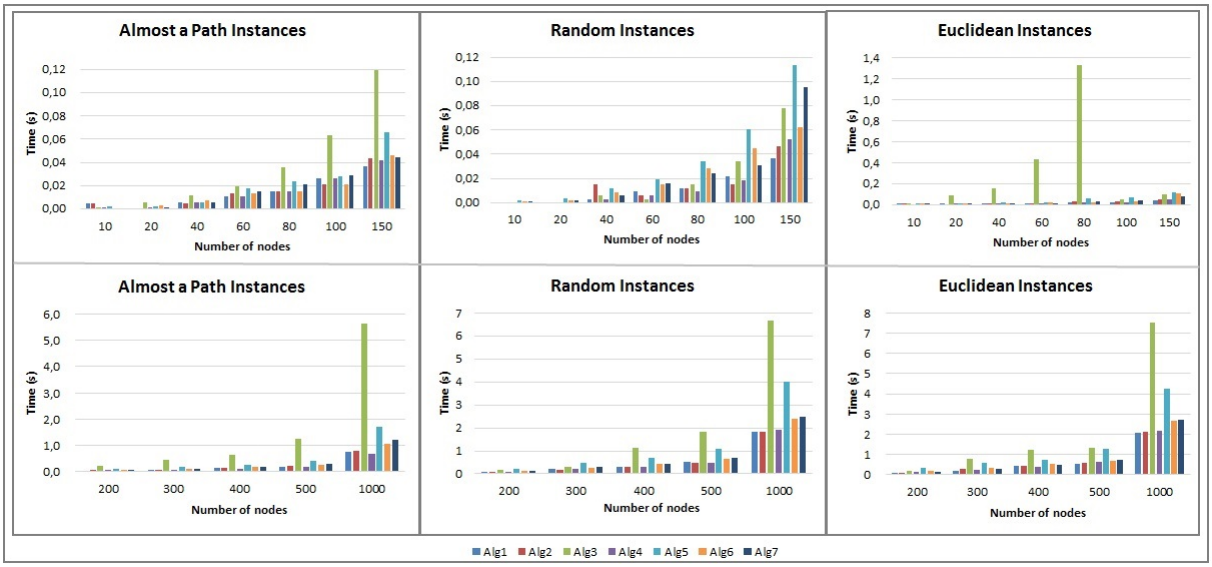

Figure 1: Comparing mean execution times (in seconds) for all the algorithms.

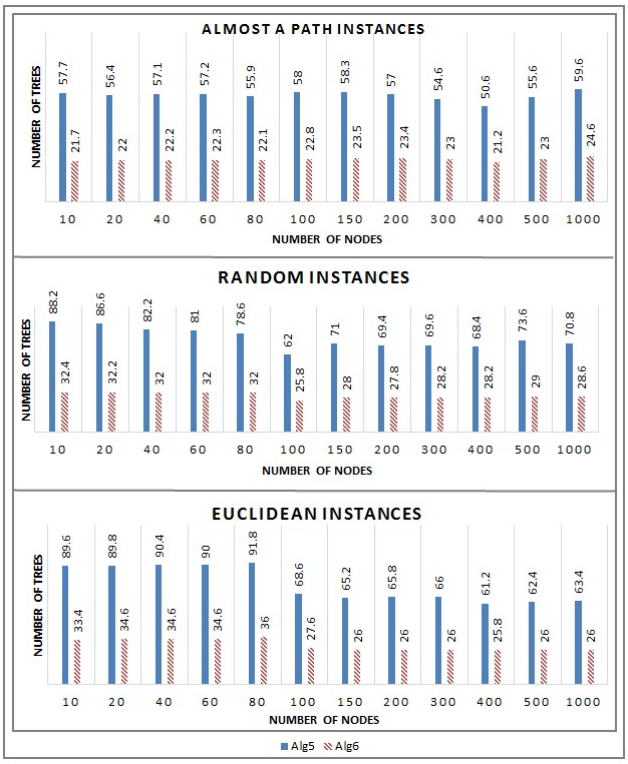

Figure 2: Comparing the mean number of trees of algorithm Alg5 and Alg6.

Interval Reduction $[\ell_k, u_k]$ with Fibonacci search

If $u_k - \ell_k < \xi \times F_n$ **then**

set $\lambda_k^a := u_k - \frac{F_{n-1}}{F_n} \times (u_k - \ell_k)$ and $\lambda_k^b := \ell_k + \frac{F_{n-1}}{F_n} \times (u_k - \ell_k)$;

obtain $\vartheta(\lambda_k^a)$ and $\vartheta(\lambda_k^b)$;

if $\vartheta(\lambda_k^a) < \vartheta(\lambda_k^b)$ **then** set $T_p := T_{p^{\lambda_k^a}}$, $\ell_{k+1} := \lambda_k^a$ and $u_{k+1} := u_k$;

Else-if $\vartheta(\lambda_k^a) > \vartheta(\lambda_k^b)$ **then** set $T_p := T_{p^{\lambda_k^b}}$, $\ell_{k+1} := \ell_k$ and $u_{k+1} := \lambda_k^b$;

Else set $\ell_{k+1} := \lambda_k^a$ and $u_{k+1} := \lambda_k^b$.

End-if.

End-if.

Comparatively to Alg5 the number of computed trees is smaller, however comparatively to Alg6 the number of computed trees is higher. Further, for the trees with more than 400 nodes it is necessary to use a small value for parameter ξ in order to obtain similar quality values for the bounds as Alg6, and the use of a small value for parameter ξ implies an increase on the number of computed trees. Additionally, a drawback of this procedure is that its performance is highly dependent on the parameter ξ that has to be tuned. We do not report computational results with this procedure because even testing several values to the ξ parameter a superiority of this procedure over the Alg6 setting for the interval reduction was not evident.

To compare the performance of the several Lagrangian based solution procedure, for each approximation scheme, for each instance set AP, R, and E and each instance size set we present the mean upper bound gap and the mean lower bound gap together with the corresponding standard deviation values. These results are presented in Tables 5, 7, 9, one table for each instance set. The top part of each table presents the mean gaps and the bottom parte of each table presents the corresponding standard deviation values. We also present the mean execution times (in seconds) and corresponding standard deviation values. These results are presented in Tables 6, 8, 10, one table for each instance set. The top part of each table presents the mean execution times and the bottom parte of each table presents the corresponding standard deviation values. In Figure 3 we compare the lower bound gaps gap_L between Alg6 and Alg7 for the three sets of instances AP, R and E.

Table 5 presents the mean gaps (top part) and corresponding standard deviation values (bottom part) for the AP instances. Algorithms Alg1 and Alg4 present the higher mean gap values. Algorithms Alg2, Alg5 and Alg6 have the same mean gap values. Algorithm Alg7 has the best mean lower bound gaps and has mean upper bound gap gap_U equal to Alg2, Alg5 and Alg6. In Figure 3 we compare the mean lower bound gaps gap_L between Alg6 (which are the same as Alg2 and Alg5) and Alg7. For the upper bound gaps gap_U, algorithm Alg6 obtains lower gaps than the Alg7 in 7.37% of the instances (7 out of 95 instances) and Alg7 presents lower gap_U in 4.21% of the instances (4 out of 95 instances). For the remaining 84 instances the upper bound gaps gap_U are equal in both algorithms.

Table 5: Mean gaps (top part) and corresponding standard deviation values (bottom part) for the AP instances.

	Alg1		Alg2		Alg3		Alg4		Alg5		Alg6		Alg7	
n	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U
10	46.516	45.433	22.292	7.138	23.551	37.666	28.855	28.688	22.292	7.138	22.292	7.138	18.987	7.138
20	26.899	17.027	9.874	0.574	10.732	6.031	31.769	17.533	9.874	0.574	9.874	0.574	9.145	0.574
40	15.654	18.451	6.362	0.579	7.004	9.332	35.398	23.925	6.362	0.579	6.362	0.579	6.101	1.545
60	11.061	11.658	4.831	0.022	5.277	7.870	40.532	12.949	4.831	0.022	4.831	0.022	4.729	0.022
80	10.767	9.798	3.778	0.058	4.277	7.888	42.424	9.798	3.778	0.058	3.778	0.058	3.701	0.058
100	7.406	6.083	3.192	1.048	3.417	6.349	43.984	7.147	3.192	1.048	3.192	1.048	3.141	0.346
150	11.774	8.559	1.680	0.773	1.867	3.893	42.587	9.439	1.680	0.773	1.680	0.773	1.651	0.445
200	4.618	3.716	0.771	0.719	0.831	1.837	42.050	3.716	0.771	0.719	0.771	0.719	0.739	1.837
300	3.647	3.050	0.299	1.628	0.350	2.399	42.251	3.050	0.299	1.628	0.299	1.628	0.282	1.628
400	10.074	9.291	0.232	1.433	0.675	9.291	40.651	9.291	0.232	1.433	0.232	1.433	0.219	2.030
500	3.455	3.314	0.207	1.265	0.224	2.351	41.860	3.894	0.207	1.265	0.207	1.265	0.198	1.771
1000	3.598	0.292	1.757	0.292	1.806	0.292	44.696	0.292	1.757	0.292	1.757	0.292	1.754	0.292
10	26.556	41.545	7.061	8.075	7.058	45.160	11.237	28.365	7.061	8.075	7.061	8.075	6.756	8.075
20	23.118	8.109	4.380	0.610	4.588	9.238	8.347	7.768	4.380	0.610	4.380	0.610	4.074	0.610
40	9.213	10.500	0.840	0.614	1.230	10.831	3.919	5.751	0.840	0.614	0.840	0.614	0.762	2.974
60	6.314	4.918	0.863	0.029	1.020	5.937	3.096	2.721	0.863	0.029	0.863	0.029	0.853	0.029
80	5.351	1.740	0.884	0.051	1.155	4.487	1.198	1.740	0.884	0.051	0.884	0.051	0.883	0.051
100	4.791	2.538	0.274	1.690	0.275	1.444	1.394	0.602	0.274	1.690	0.274	1.690	0.280	1.087
150	8.054	3.462	0.360	1.271	0.582	4.545	1.589	1.750	0.360	1.271	0.360	1.271	0.361	0.931
200	3.300	0.911	0.345	0.679	0.330	1.500	2.141	0.911	0.345	0.679	0.345	0.679	0.350	1.500
300	1.597	0.923	0.214	1.629	0.194	1.539	1.966	0.923	0.214	1.629	0.214	1.629	0.218	1.629
400	8.370	10.450	0.057	1.867	0.677	10.450	4.709	10.450	0.057	1.867	0.057	1.867	0.052	1.966
500	3.565	1.903	0.082	1.295	0.083	2.040	1.856	1.049	0.082	1.295	0.082	1.295	0.079	2.108
1000	1.704	0.652	1.115	0.652	1.157	0.652	3.638	0.652	1.115	0.652	1.115	0.652	1.116	0.652

Table 6: Mean execution times, in seconds, (top part) and corresponding standard deviation values(bottom part) for the AP instances.

n	Alg1	Alg2	Alg3	Alg4	Alg5	Alg6	Alg7
10	0.005	0.005	0.002	0.002	0.002	0.000	0.000
20	0.000	0.000	0.006	0.002	0.002	0.003	0.002
40	0.006	0.005	0.012	0.006	0.006	0.008	0.006
60	0.011	0.014	0.020	0.011	0.018	0.014	0.015
80	0.015	0.015	0.036	0.015	0.024	0.015	0.021
100	0.026	0.021	0.064	0.026	0.028	0.021	0.029
150	0.037	0.043	0.120	0.042	0.066	0.046	0.045
200	0.043	0.056	0.231	0.056	0.091	0.065	0.062
300	0.087	0.087	0.465	0.081	0.165	0.112	0.118
400	0.127	0.130	0.649	0.115	0.260	0.171	0.174
500	0.190	0.212	1.254	0.184	0.429	0.271	0.283
1000	0.742	0.801	5.622	0.689	1.708	1.076	1.232
10	0.010	0.015	0.005	0.005	0.004	0.000	0.000
20	0.000	0.000	0.008	0.005	0.001	0.006	0.005
40	0.008	0.007	0.006	0.008	0.002	0.008	0.008
60	0.007	0.005	0.008	0.007	0.002	0.009	0.000
80	0.007	0.007	0.007	0.007	0.004	0.010	0.008
100	0.008	0.013	0.016	0.008	0.010	0.011	0.011
150	0.018	0.022	0.013	0.023	0.021	0.024	0.014
200	0.013	0.024	0.026	0.009	0.010	0.013	0.011
300	0.008	0.014	0.039	0.007	0.015	0.013	0.021
400	0.007	0.009	0.148	0.014	0.015	0.011	0.007
500	0.006	0.008	0.193	0.012	0.016	0.007	0.006
1000	0.028	0.014	0.100	0.034	0.079	0.036	0.047

Table 6 presents mean execution times, in seconds, (top part) and corresponding standard deviation values (bottom part) for the AP instances. Execution mean times are, almost all, less than 1 second, except for five occurrences for which four of these use less than 2 seconds. Algorithms Alg3 and Alg5 use more execution time than the others.

Table 7 presents the mean gaps (top part) and corresponding standard deviation values (bottom part) for the R instances. In general, and contrary to what happened to the AP instances, algorithms Alg1 and Alg4 do not have much higher gaps when compared with the other algorithms. We have the same mean gap values for algorithms Alg2, Alg5 and Alg6. The upper bound mean gaps for Alg3 are the worse. In Figure 3 we compare the lower bound gaps gap_L between Alg6 (which are the same as Alg2 and Alg5) and Alg7. Algorithm Alg7 presents the lowest values for the lower bound gaps gap_L. All algorithms have the same upper bound gaps gap_U.

Table 8 presents mean execution times, in seconds, (top part) and corresponding standard deviation values (bottom part) for the R instances. As before, execution mean times are, almost all, less than 1 second, except for ten occurrences, for which six use less than 2 seconds. Except for Alg3 all the other algorithms use mean computational times less than 5 seconds. As before we also may say that algorithms Alg3 and Alg5 are the most time consuming. And we can notice that for instances with 300 or less nodes Alg5 is the most time consuming while for instances with 400 or more nodes Alg3 is the most time consuming.

Table 9 presents the mean gaps (top part) and corresponding standard deviation values (bottom part) for the E instances. In general, and contrary to what happened to the AP instances, algorithms Alg1 and Alg4 do not have higher gaps when compared with the other algorithms. In Figure 3 we compare the lower bound gaps gap_L between Alg6 and Alg7. We have the same mean gap values for algorithms Alg2, Alg5 and Alg6. Algorithm Alg7 has the best lower bound mean gaps and has mean upper bound gaps gap_U equal to Alg2, Alg5 and Alg6, except for one instance with 1000 nodes.

Table 10 presents mean execution times, in seconds, (top part) and corresponding standard deviation values (bottom part) for the E instances. As before, execution mean times are, almost all, less than 1 second, except for four occurrences, that use less than 2 seconds. As before we may say that algorithms Alg3 and Alg5 are the most time consuming, being Alg3 more time consuming than Alg5.

In Table 11 we compare the HP procedure (see page 14 for more details) with Alg7, the algorithm that obtains the best lower bounds. For these two procedures we display the mean gaps for the three sets of instances considered, AP, R and E, and specified in the first line of the table. For each node set size (specified in each line 3 to 14) and for each instance set (specified in the first line) we show in columns two, four and six named HP the mean of the linear programming gap. For each instance this gap is $gap_{LP} = \frac{OPT-LP}{OPT} \times 100$, where LP is the linear programming bound obtained by the weighted MTZ formulation used in HP. For each node set size (specified in each line 3 to 14) and for each instance set (specified in the first line)

Table 7: Mean gaps (top part) and corresponding standard deviation values (bottom part) for the R instances.

	Alg1		Alg2		Alg3		Alg4		Alg5		Alg6		Alg7	
n	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U
10	15.055	3.840	15.055	3.840	15.612	7.713	16.780	3.840	15.055	3.840	15.055	3.840	12.542	3.840
20	4.196	6.208	4.079	5.285	7.304	22.292	6.350	14.955	4.079	5.285	4.079	5.285	3.477	5.285
40	1.788	2.743	1.769	2.743	3.549	8.828	1.889	2.743	1.769	2.743	1.769	2.743	1.556	2.743
60	0.622	1.312	0.589	1.312	0.818	2.775	0.622	1.312	0.589	1.312	0.589	1.312	0.529	1.312
80	0.401	1.505	0.387	1.505	1.255	5.343	0.447	2.242	0.387	1.505	0.387	1.505	0.328	1.505
100	0.072	0.547	0.069	0.547	0.250	3.581	0.070	0.547	0.069	0.547	0.069	0.547	0.054	0.547
150	0.064	0.787	0.063	0.787	0.132	2.068	0.065	0.787	0.063	0.787	0.063	0.787	0.055	0.787
200	0.044	0.533	0.040	0.533	0.299	4.111	0.046	0.704	0.040	0.533	0.040	0.533	0.034	0.533
300	0.016	0.567	0.016	0.567	0.057	1.811	0.020	0.713	0.016	0.567	0.016	0.567	0.013	0.567
400	0.003	0.145	0.003	0.145	0.030	0.930	0.004	0.145	0.003	0.145	0.003	0.145	0.003	0.145
500	0.003	0.220	0.003	0.220	0.060	1.445	0.004	0.275	0.003	0.220	0.003	0.220	0.002	0.220
1000	0.036	0.143	0.036	0.071	0.038	0.303	0.036	0.143	0.036	0.071	0.036	0.071	0.036	0.117
10	4.034	3.430	4.034	3.430	3.950	8.592	3.811	3.430	4.034	3.430	4.034	3.430	3.685	3.430
20	0.365	4.670	0.312	5.006	3.816	28.414	3.873	17.027	0.312	5.006	0.312	5.006	0.142	5.006
40	0.183	1.676	0.182	1.676	3.073	12.972	0.267	1.676	0.182	1.676	0.182	1.676	0.209	1.676
60	0.117	1.074	0.077	1.074	0.205	3.226	0.090	1.074	0.077	1.074	0.077	1.074	0.066	1.074
80	0.029	1.320	0.037	1.320	1.862	8.213	0.077	1.645	0.037	1.320	0.037	1.320	0.034	1.320
100	0.047	0.264	0.045	0.264	0.294	3.901	0.046	0.264	0.045	0.264	0.045	0.264	0.042	0.264
150	0.035	0.684	0.035	0.684	0.123	1.619	0.036	0.684	0.035	0.684	0.035	0.684	0.034	0.684
200	0.019	0.379	0.019	0.379	0.418	5.374	0.025	0.515	0.019	0.379	0.019	0.379	0.018	0.379
300	0.001	0.621	0.001	0.621	0.043	1.256	0.005	0.692	0.001	0.621	0.001	0.621	0.002	0.621
400	0.003	0.098	0.003	0.098	0.054	1.417	0.003	0.098	0.003	0.098	0.003	0.098	0.003	0.098
500	0.002	0.136	0.002	0.136	0.106	1.861	0.002	0.223	0.002	0.136	0.002	0.136	0.002	0.136
10000	0.046	0.120	0.046	0.081	0.046	0.183	0.046	0.120	0.046	0.081	0.046	0.081	0.046	0.085

Table 8: Mean execution times, in seconds, (top part) and corresponding standard deviation values (bottom part) for the R instances.

n	Alg1	Alg2	Alg3	Alg4	Alg5	Alg6	Alg7
10	0.000	0.000	0.000	0.000	0.001	0.001	0.001
20	0.000	0.000	0.000	0.000	0.003	0.002	0.002
40	0.003	0.015	0.006	0.003	0.012	0.008	0.006
60	0.009	0.006	0.003	0.006	0.020	0.015	0.016
80	0.012	0.012	0.015	0.009	0.034	0.028	0.024
100	0.021	0.015	0.034	0.018	0.060	0.045	0.031
150	0.037	0.046	0.078	0.052	0.114	0.062	0.095
200	0.078	0.081	0.190	0.075	0.205	0.149	0.121
300	0.208	0.162	0.315	0.215	0.463	0.275	0.303
400	0.318	0.302	1.120	0.309	0.699	0.446	0.436
500	0.502	0.480	1.812	0.477	1.101	0.661	0.686
1000	1.853	1.828	6.686	1.922	4.034	2.405	2.477
10	0.000	0.000	0.000	0.000	0.001	0.000	0.000
20	0.000	0.000	0.000	0.000	0.001	0.001	0.001
40	0.007	0.000	0.008	0.007	0.005	0.004	0.001
60	0.008	0.008	0.007	0.008	0.003	0.004	0.005
80	0.007	0.007	0.000	0.008	0.004	0.016	0.007
100	0.009	0.000	0.013	0.007	0.032	0.027	0.007
150	0.008	0.011	0.019	0.009	0.029	0.015	0.078
200	0.000	0.007	0.102	0.007	0.087	0.067	0.043
300	0.067	0.008	0.243	0.124	0.100	0.043	0.120
400	0.052	0.018	0.349	0.007	0.101	0.064	0.055
500	0.063	0.047	1.048	0.067	0.062	0.075	0.073
1000	0.035	0.063	1.931	0.147	0.423	0.074	0.142

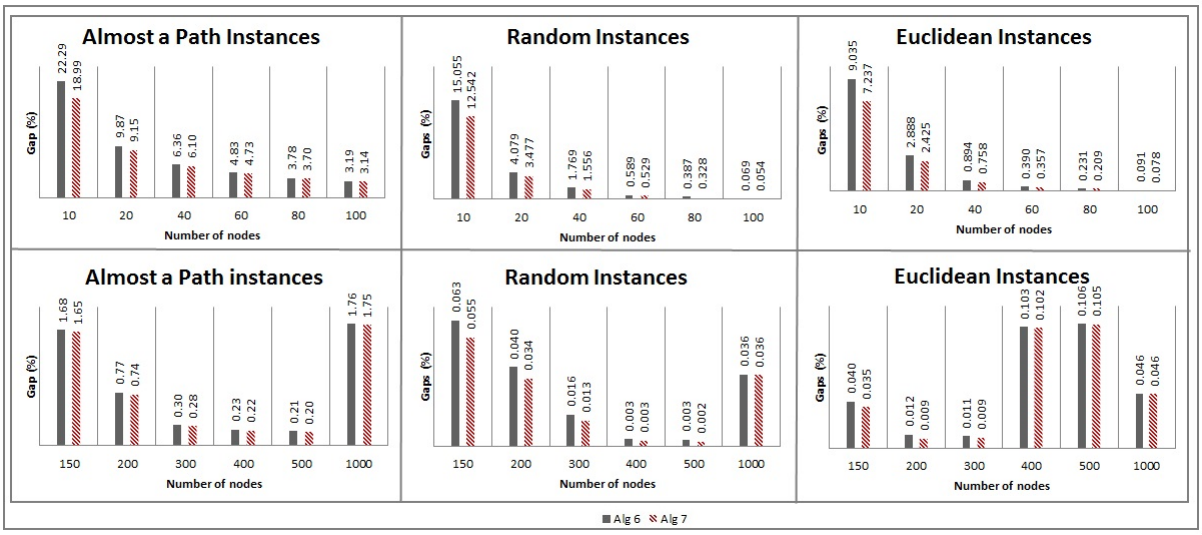

Figure 3: Comparing the lower bound gaps gap_L between Alg6 and Alg7 for the three sets of instances AP, R and E.

Table 9: Mean gaps (top part) and corresponding standard deviation values (bottom part) for the E instances.

	Alg1		Alg2		Alg3		Alg4		Alg5		Alg6		Alg7	
n	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U	gap_L	gap_U
10	9.035	5.274	9.035	5.274	9.039	5.274	10.816	5.274	9.035	5.274	9.035	5.274	7.237	5.274
20	2.938	2.250	2.888	2.250	2.888	2.250	6.058	11.323	2.888	2.250	2.888	2.250	2.425	2.250
40	0.894	1.959	0.894	1.959	0.894	1.959	0.969	1.959	0.894	1.959	0.894	1.959	0.758	1.959
60	0.399	0.890	0.390	0.747	0.394	0.747	0.428	1.039	0.390	0.747	0.390	0.747	0.357	0.747
80	0.239	0.576	0.231	0.576	0.232	0.576	0.239	0.576	0.231	0.576	0.231	0.576	0.209	0.576
100	0.104	0.706	0.091	0.563	0.120	1.378	0.127	1.480	0.091	0.563	0.091	0.563	0.078	0.563
150	0.043	0.284	0.040	0.284	0.058	0.628	0.041	0.284	0.040	0.284	0.040	0.284	0.035	0.284
200	0.013	0.402	0.012	0.402	0.064	1.416	0.012	0.402	0.012	0.402	0.012	0.402	0.009	0.402
300	0.011	0.280	0.011	0.219	0.014	0.280	0.012	0.280	0.011	0.219	0.011	0.219	0.009	0.219
400	0.103	0.000	0.103	0.000	0.115	0.500	0.103	0.000	0.103	0.000	0.103	0.000	0.102	0.000
500	0.106	0.034	0.106	0.000	0.114	0.572	0.106	0.108	0.106	0.000	0.106	0.000	0.105	0.000
1000	0.046	0.007	0.046	0.000	0.052	0.496	0.046	0.030	0.046	0.000	0.046	0.000	0.046	0.007
10	0.993	3.493	0.993	3.493	0.994	3.493	2.833	3.493	0.993	3.493	0.993	3.493	0.915	3.493
20	0.610	2.685	0.640	2.685	0.640	2.685	2.726	11.121	0.640	2.685	0.640	2.685	0.459	2.685
40	0.106	0.390	0.106	0.390	0.106	0.390	0.126	0.390	0.106	0.390	0.106	0.390	0.099	0.390
60	0.056	0.793	0.059	0.708	0.060	0.708	0.094	1.336	0.059	0.708	0.059	0.708	0.066	0.708
80	0.026	0.428	0.016	0.428	0.015	0.428	0.013	0.428	0.016	0.428	0.016	0.428	0.016	0.428
100	0.049	0.675	0.029	0.665	0.040	0.890	0.076	1.526	0.029	0.665	0.029	0.665	0.027	0.665
150	0.018	0.194	0.016	0.194	0.025	0.578	0.017	0.194	0.016	0.194	0.016	0.194	0.015	0.194
200	0.008	0.195	0.007	0.195	0.069	1.585	0.007	0.195	0.007	0.195	0.007	0.195	0.006	0.195
300	0.040	0.156	0.040	0.187	0.042	0.156	0.040	0.156	0.040	0.187	0.040	0.187	0.040	0.187
400	0.115	0.000	0.115	0.000	0.120	0.687	0.115	0.000	0.115	0.000	0.115	0.000	0.115	0.000
500	0.057	0.076	0.056	0.000	0.064	0.497	0.056	0.242	0.056	0.000	0.056	0.000	0.056	0.000
1000	0.041	0.016	0.041	0.000	0.040	0.439	0.041	0.050	0.041	0.000	0.041	0.000	0.041	0.016

Table 10: Mean execution times, in seconds, (top part) and corresponding standard deviation values (bottom part) for the E instances.

n	Alg1	Alg2	Alg3	Alg4	Alg5	Alg6	Alg7
10	0.003	0.003	0.003	0.000	0.008	0.003	0.000
20	0.009	0.000	0.084	0.003	0.006	0.001	0.003
40	0.003	0.003	0.149	0.006	0.015	0.010	0.004
60	0.009	0.006	0.427	0.012	0.022	0.022	0.013
80	0.015	0.031	1.332	0.015	0.062	0.022	0.025
100	0.021	0.025	0.046	0.018	0.064	0.033	0.043
150	0.043	0.053	0.100	0.052	0.118	0.107	0.076
200	0.090	0.078	0.174	0.128	0.318	0.200	0.125
300	0.174	0.258	0.792	0.230	0.572	0.336	0.297
400	0.449	0.408	1.213	0.387	0.723	0.528	0.471
500	0.539	0.596	1.307	0.617	1.253	0.684	0.739
10	0.007	0.007	0.007	0.000	0.015	0.004	0.000
20	0.008	0.000	0.171	0.007	0.004	0.001	0.004
40	0.007	0.007	0.032	0.008	0.007	0.006	0.004
60	0.008	0.008	0.107	0.007	0.005	0.013	0.004
80	0.011	0.035	0.271	0.000	0.038	0.002	0.004
100	0.009	0.009	0.025	0.007	0.024	0.006	0.019
150	0.007	0.014	0.045	0.014	0.024	0.060	0.016
200	0.030	0.000	0.058	0.094	0.094	0.089	0.031
300	0.007	0.099	0.500	0.074	0.118	0.091	0.076
400	0.072	0.086	0.790	0.071	0.052	0.133	0.051
500	0.101	0.093	0.800	0.106	0.117	0.056	0.101
1000	0.085	0.094	2.808	0.072	0.120	0.086	0.066

Table 11: Comparing the mean gaps between HP procedure and algorithm Alg7

	AP		R		E	
n	HP	Alg7	HP	Alg7	HP	Alg7
10	22.292	18.987	15.056	12.542	9.035	7.237
20	9.874	9.145	4.079	3.477	2.888	2.425
40	6.362	6.101	1.769	1.556	0.894	0.758
60	4.830	4.729	0.589	0.529	0.390	0.357
80	3.778	3.701	0.387	0.328	0.231	0.209
100	3.192	3.141	0.068	0.328	0.091	0.209
150	1.680	1.651	0.063	0.054	0.040	0.078
200	0.770	0.739	0.040	0.034	0.012	0.009
300	0.299	0.282	0.016	0.013	0.011	0.009
400	0.232	0.219	0.021	0.003	2.848	0.102
500	0.207	0.198	0.003	0.002	2.938	0.105
1000	1.757	1.754	0.094	0.036	3.517	0.046

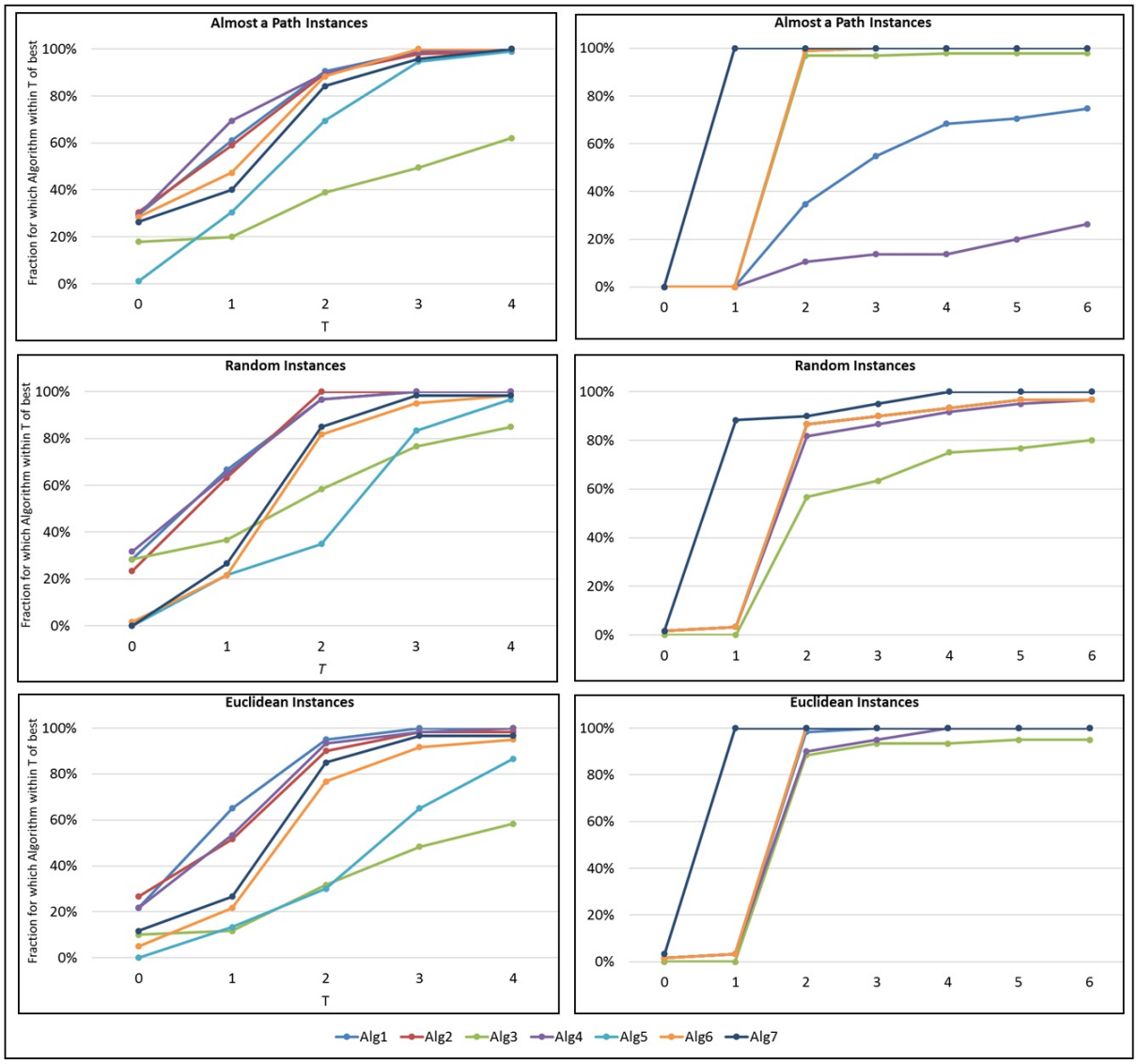

Figure 4: Performance profiles for the computational times (in seconds) in the left figure and for the gaps in the right figure, for each set of instances.

we show in columns three, five and seven named Alg7 the mean of the lower bound gaps for the Alg7, which is $gap_{LB} = \frac{OPT-LB}{OPT} \times 100$, where LB is the lower bound obtained by Alg7.

It is worth to note that, generally, the gaps decrease with the increase on the number of nodes. This can be explained because in such cases the number of edges that can replace an edge discarded from an infeasible solution also increases. Therefore the obtention of a feasible solution does not gets harder.

In Figure 4, following [8, 10], we present some performance profiles to compare the performance of the algorithms Alg1, Alg2, Alg3, Alg4, Alg5, Alg6, Alg7 for each one of the three sets of instances, sets AP, E and R. We used $n_{AP} = 95$ instances of the set AP and $n_E = n_R = 60$ instances of each set E and R. For each set of instances two performance measures were considered: the computational times (in seconds) presented in the left part of the figure, and the lower bound gaps in the right part of the figure.

We explain the construction of the performance profiles for the computational times. Similarly

they are build for the lower bound gaps. Consider that t_{ia} is the computational time (in seconds) used by algorithm $a \in \{\text{Alg1, Alg2, Alg3, Alg4, Alg5, Alg6, Alg7}\}$ to obtain an approximate value to instance i from a set of instances. To build the performance profiles a baseline for comparisons is required. Therefore, we compare the performance of instance i by algorithm a with the best performance by any algorithm on this instance; that is, we use the performance ratio $r_{ia} = t_{ia}/\underline{t}_i$ where $\underline{t}_i = \min\{t_{ia}, a \in \{\text{Alg1, } \ldots, \text{ Alg7}\}\}$. To obtain an overall assessment of the performance of the algorithms define $\mu_a(T) = s_a(T)/n_j$ where $s_a(T)$ is the number of instances such that the performance ratio $r_{ia} \leq T$ and where $n_j = 95$ or 60, depending on the set of instances in consideration. Thus, $\mu_a(T)$ is a probability estimate for algorithm a that a performance ratio r_{ia} is within a factor $T \in R$ of the best possible ratio. The function μ_a is the empirical (cumulative) distribution function for the performance ratio.

We presented several Lagrangian based schemes to approximate the WMST problem solution. Their simplicity has its price as the quality of the approximation depends greatly on their ability to find near optimal multipliers quickly and specific to each instance. In many cases the method can only give a coarse approximation of the optimal value. As a consequence different gaps and computational times are reported for the same problem instance depending on the specificity of the overall algorithm settings.

The following final remark can be done. When the computational time is a concern Alg2 is better suited if one is interested with obtaining a good solution fast. If the interest is with the quality of the solution, Alg7 is a good recommendation to obtain good lower and upper bounds.

7 Conclusions

Our computational results show that the Lagrangian based algorithms are fast (use less than 13 seconds in our experiments) and present small gap values. Therefore these algorithms are a good choice in obtaining both a lower and an upper bound for the WMST. We present seven different settings, among them four were published by others, another, Alg3, is the classical subgradient setting and two other settings, Alg6 and Alg7, are new. Four of the algorithms Alg2, Alg5, Alg6 and Alg7 are very efficient in all instances sets, and several optimal solutions were obtained when using those settings. Algorithm Alg5 has the disadvantage of being very time consuming.

The lower bound values obtained using the Lagrangian based algorithms Alg2, Alg5 and Alg6 are equal to the lower bound values obtained with the linear programming of the weighted MTZ model used within the HP procedure. Algorithm Alg7 obtains better lower bounds than the lower bound values obtained with the linear programming of the weighted MTZ model.

If the computational time is a concern Alg2 obtains a good solution fast. However the algorithm Alg7 is the best algorithm as it obtains the best lower bounds.

Acknowledgements

The research of the two authors has been partially supported by Portuguese funds through the *Center for Research and Development in Mathematics and Applications (CIDMA)* and FCT, the Portuguese Foundation for Science and Technology, within project UID/MAT/04106/2013.

The research of Eulália Santos has been supported by a research fellowship (grant SFRH/BD/-46394/2008) through FCT, the Portuguese Foundation for Science and Technology.

References

[1] V. Aggarwal, Y. P. Aneja, and K. P. K. Nair. Minimal spanning tree subject to a side constraint. *Computers & Operations Research*, 9:287–296, 1982.

[2] A. Agra, A. Cerveira, C. Requejo, and E. Santos. On the weight-constrained minimum spanning tree problem. In *Proceedings of the International Network Optimization Conference*, volume 6701 of *Lecture Notes in Computer Science*, pages 156–161, 2011.

[3] A. Agra, C. Requejo, and E. Santos. Implicit cover inequalities. *Journal of Combinatorial Optimization*, 31(3):1111–1129, 2016.

[4] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, 1993.

[5] L. Amado and P. Bárcia. New polynomial bounds for matroidal knapsacks. *European Journal of Operational Research*, 95:201–210, 1996.

[6] K. Andersen, K. Jörnsten, and M. Lind. On bicriterion minimal spanning trees: an approximation. *Computers & Operations Research*, 23:1171–1182, 1996.

[7] D. Blokh and G. Gutin. An approximation algorithm for combinatorial optimization problems with two parameters. *Australasian Journal of Combinatorics*, 14:157–164, 1996.

[8] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[9] *FICO Xpress Optimization Suite*. http://www.fico.com/en/products/fico-xpress-optimization-suite.

[10] N. Gould and J. Scott. A note on performance profiles for benchmarking software. *ACM Transactions on Mathematical Software*, 43(2):15:1–15:5, 2016.

[11] H. Hamacher and G. Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52:209–230, 1994.

[12] G. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[13] R. Hassin. On maximizing functions by Fibonacci search. *The Fibonacci Quarterly*, 19(4):347–351, 1981.

[14] R. Hassin and A. Levin. An efficient polynomial time approximation scheme for the constrained minimum spanning tree problem using matroid intersection. *SIAM Journal on Computing*, 33(2):261–268, 2004.

[15] M. Held, P. Wolfe, and H. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

[16] S. Henn. Weight-constrained minimum spanning tree problem. Master's thesis, University of Kaiserslautern, Kaiserslautern, Germany, 2007.

[17] S. Hong, S. Chung, and B. H. Park. A fully polynomial bicriteria approximation scheme for the constrained spanning tree problem. *Operations Research Letters*, 32:233–239, 2004.

[18] K. Jörnsten and S. Migdalas. Designing a minimal spanning tree network subject to a budget constraint. *Optimization*, 19(4):475–484, 1988.

[19] A. Jüttner, B. Szviatovszki, I. M?cs, and Z. Rajk?. Lagrange relaxation based method for the QoS routing problem. In *Proceedings IEEE INFOCOM 2001, 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 859–868, 2001.

[20] T. L. Magnanti and L. A. Wolsey. Optimal trees. In M. Ball, T. L. Magnanti, C. Monma, and G. L. Nemhauser, editors, *Network Models*, Handbooks in Operations Research and Management Science, Vol. 7, pages 503–615. Elsevier Science Publishers, North-Holland, 1995.

[21] K. Mehlhorn and M. Ziegelmann. CNOP - a package for constrained network optimization. In A. Buchsbaum and J. Snoeyink, editors, *Algorithm Engineering and Experimentation*, volume 2153 of *Lecture Notes in Computer Science*, pages 17–31. Springer Berlin / Heidelberg, 2001.

[22] D. Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.

[23] R. Ramos, S. Alonso, J. Sicilia, and C. González. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, 111:617–628, 1998.

[24] R. Ravi and M. Goemans. The constrained minimum spanning tree problem. In *Proceedings of the Scandinavian Workshop on Algorithmic Theory*, volume 1097 of *Lecture Notes in Computer Science*, pages 66–75, 1996.

[25] C. Requejo, A. Agra, A. Cerveira, and E. Santos. Formulations for the weight-constrained minimum spanning tree problem. In *Proceedings of the International Conference on Numerical Analysis and Applied Mathematics*, volume 1281 of *AIP Conference Proceedings*, pages 2166–2169, 2010.

[26] A. Shogan. Constructing a minimal-cost spanning tree subject to resource constraints and flow requirements. *Networks*, 13:169–190, 1983.

[27] N. Shor. *Minimization Methods for Non-Differentiable Functions*. Springer-Verlag, 1985. English translation.

[28] F. Sourd and O. Spanjaard. A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3):472–484, 2008.

[29] S. Steiner and T. Radzik. Computing all efficient solutions of the biobjective minimum spanning tree problem. *Computers & Operations Research*, 35(1):198–211, 2008.

[30] Y. Xiao, K. Thulasiraman, G. Xue, and A. Jüttner. The constrained shortest path problem: Algorithmic approaches and an algebraic study with generalization. *AKCE International Journal of Graphs and Combinatorics*, 2(2):63–86, 2005.

[31] G. Xue. Primal-dual algorithms for computing weight-constrained shortest paths and weight-constrained minimum spanning trees. In *Proceedings IEEE IPCCC '00, IEEE International Conference on Performance, Computing, and Communications*, pages 271–277, 2000.

[32] G. Xue. Minimum-cost QoS multicast and unicast routing in communication networks. *IEEE Transactions on Communications*, 51(5):817–824, 2003.

[33] T. Yamada, K. Watanabe, and S. Kataoka. Algorithms to solve the knapsack constrained maximum spanning tree problem. *International Journal of Computer Mathematics*, 82:23–34, 2005.