Corinna Marlene Mathwieser, BSc

The Minimum Spanning Tree Problem Under Explorable Uncertainty

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieurin

Master's degree programme: Mathematics

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dr. Eranda Dragoti-Çela

Institute of Discrete Mathematics

Graz, October 2020

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date, Signature

Abstract

The Minimum Spanning Tree Problem under Explorable Uncertainty (MST-U) is an uncertainty version of the Minimum Spanning Tree Problem where for each edge a set, usually an open interval, containing the weight of the edge is given and the precise weight of an edge can be revealed upon request. Querying an edge for its weight comes at extra cost. The goal is to query edges at minimum cost such that a minimum spanning tree can be identified with certainty. In this master's thesis we review different algorithms for the MST-U, as well as existing bounds on the performance of algorithms. For the new special case of cactus graphs we provide an optimal randomized algorithm. A different version of the problem is the Minimum Spanning Tree Problem under Vertex Uncertainty (V-MST-U) where vertices are points in the plane whose positions are given as open sets and the edge weights correspond to the distances between the end vertices of an edge. For this problem we derive a lower bound of 2.5 on the performance of any randomized algorithm and introduce a special case of instances for which we present a randomized algorithm with performance guarantee 3. We review different explorable uncertainty problems that are related to MST-U, such as the approximate MST-U or MST weight under uncertainty. Moreover, we introduce the problem of finding a minimum weight spanning star under uncertainty for which we show that no algorithm can achieve constant competitive ratio.

Contents

Introduction

Many real world problems do not allow to work with precise data. Instead, parts of the input are uncertain or only known approximately. This motivates the concept of combinatorial optimization under uncertainty. Different approaches in this category include stochastic optimization, where the input data is known to follow a specific probability distribution, robust optimization which aims to find good solutions for all possible inputs (for some appropriate measure of solution quality) and explorable uncertainty. In the setting of explorable or queryable uncertainty, it is possible to obtain more precise or even exact data by making queries. However, any query causes additional exploration cost. In an applied scenario this might be time, money or other resources which are needed for further measurements.
This master's thesis deals with the *Minimum Spanning Tree Problem under Explorable Uncertainty* (MST-U), which was first introduced by Erlebach and Hoffmann [7]. In an instance of MST-U, each edge is equipped with an uncertainty set and a query cost. The uncertainty set, usually an interval, is guaranteed to contain the edge's weight. An edge query (we also say update) reveals the edge's true weight. The goal is to find a set of queries of minimum cost which allows to find a minimum spanning tree (MST). The queries may be chosen adaptively, i.e. we are allowed to choose the next update based on the previous outcomes of edge queries. In general, we consider two types of algorithms: deterministic algorithms and randomized algorithms. *Deterministic algorithms* are allowed to make their choices only based on the graph structure, the uncertainty sets, the outcome of edge queries and the query costs. *Randomized algorithms* may use additional randomized parameters which are drawn from a specified probability distribution. We measure an algorithm's performance by its *competitive ratio*, the ratio between the query cost of the algorithm's solution and the optimal query cost (see Definition 1.2.)

The first work on problems where parts of the input are uncertain and can be queried is due to Kahan [13]. Since then, explorable uncertainty has been considered for different combinatorial problems, e.g. shortest paths (see Feder et al. [8]), scheduling (see Dürr et al. [3]) and the Knapsack Problem (see Goerigk [11]). Erlebach et al. were the first to introduce the Minimum Spanning Tree Problem under Uncertainty in [7]. They showed that without restrictions made on the uncertainty sets, no algorithm can achieve constant competitive ratio, which is why subsequently only instances with uncertainty sets in the form of open intervals or singletons are considered. Erlebach et al. [7] presented the deterministic algorithm U-RED for the MST-U with uniform query

costs which achieves competitive ratio 2 and proved that no deterministic algorithm can have a smaller competitive ratio. Moreover, they introduced a different version of this problem, the *Minimum Spanning Tree Problem under Vertex Uncertainty* (V-MST-U), where vertices are points in the plane with uncertain locations and the edge weights correspond to the distances of the respective end vertices. They showed that their algorithm can be adapted to work for this setting as well, if uncertainty sets are (topologically) open. An important question left open was the effect of randomization in the setting of MST-U. This question was successfully answered by Megow et al. in [15] where they provided the randomized algorithm RANDOM with performance ratio $1 + \frac{1}{\sqrt{2}}$. The best known bound for randomized algorithms is 1.5 and was observed by Erlebach and Hoffmann in [5]. Furthermore, [15] transformed their randomized algorithm into the deterministic algorithm BALANCE with performance ratio 2, which solves instances with general query cost. Megow et al. [15] also introduced the problem of finding the *weight of an MST under uncertainty* (W-MST-U), the problem α-MST-U of finding an *α-approximate MST* (see Defintion 5.2) under uncertainty as well as a version of MST-U, where queries return subintervals instead of the precise edge weights (*OP-OP model*). Test results for the MST-U are provided by Focke et al. in [9]. In [4], Erlebach and Hoffmann deal with the *verification problem* for the MST-U with uniform query costs, i.e. the problem of computing an optimal query set if the uncertainty sets as well as the exact edge weights are given. They show that the verification problem for MST-U with uniform query costs is solvable in polynomial time, while the verification problem for the vertex uncertainty problem V-MST-U is NP-hard.

Chapter 1 of this master's thesis gives the precise definition of MST-U and further basic definitions and concepts used through the rest of the thesis. Chapter 2 deals with deterministic algorithms and gives an overview on existing bounds, algorithms and the concepts behind U-RED and BALANCE. In Chapter 3, which treats randomized algorithms for MST-U, we consider the special case of cactus graphs and introduce the randomized algorithm RANDOM$_C$. RANDOM$_C$ achieves competitive ratio 1.5 for the specified instance type, which is optimal. Chapter 4 deals with the connection between MST-U and the Minimum Bipartite Vertex Cover Problem which was established by [4] when presenting their verification algorithm for MST-U with uniform query cost. We show that the verification problem for MST-U with general query costs can be solved in polynomial time by adapting the algorithm of [4] to the non-uniform case. In Chapter 5, related problems from the literature are discussed: the previously mentioned vertex uncertainty problem V-MST-U, α-MST-U, W-MST-U, the OP-OP model as well as the generalization of MST-U to the *Minimum Matroid Base Problem under Uncertainty*. So far, V-MST-U has only been considered in the deterministic setting. We prove that no randomized algorithm can have a performance guarantee better than 2.5. For the special case of instances where no two cycles share a non-trivial vertex we introduce the

algorithm V-RANDOM$_C$ and show that V-RANDOM$_C$ achieves competitive ratio at most 3. Finally, we introduce the *Minimum Spanning Star Problem under (Explorable) Uncertainty* (MSS-U). The MSS-U is defined analogously to the MST-U except that we are not aiming to find a general minimum spanning tree but a spanning star of minimum weight. (A star is a tree where all but one vertices have degree 1.) For the MSS-U we derive a negative result with respect to competitive analysis, i.e we show that no algorithm for MSS-U can achieve constant competitive ratio.

1 Problem definition and notations

The Minimum Spanning Tree Problem under Explorable Uncertainty (MST-U) is defined on an undirected graph $G = (V, E)$ with edge weights. However, the precise weight function is not known a priori. Instead, for each edge $e \in E$, we are given an *uncertainty set* A_e, that contains the actual edge weight $\bar{w}_e$ of e. If an uncertainty set A_e contains a single element, we say A_e is *trivial*. An edge e is trivial if A_e is trivial. For an edge $e \in E$, we denote by $L_e := \inf A_e$ the infimum and by $U_e := \sup A_e$ the supremum of the uncertainty set. We will also refer to L_e and U_e as the *lower and the upper limit* of A_e (or of e) respectively. We can query an edge e to determine its weight $\bar{w}_e$. If we query e, the set A_e is updated to a singleton set containing only $\bar{w}_e$. This is referred to as *query* or *update*. The cost of querying an edge e is $q_e \in \mathbb{R}_{>0}$. Given uncertainty sets A_e, $e \in E$ for the edge weights, a weight function $w : E \to \mathbb{R}_{\geq 0}$ such that $w_e := w(e)$ is contained in A_e is called a *realisation of edge weights*. The task is to find an edge set T of a minimum spanning tree in G with edge weights $\bar{w}_e$, while minimizing the total cost of queries needed to find T. This means, that we are aiming to decide on a query set $Q \subset E$ of minimum query cost, such that after querying the edges in Q, we are able to identify an edge set $T \subset E$, which for any possible realisation w of edge weights is the edge set of a minimum spanning tree in G with weight function w. In this case we say that a minimum spanning tree *can be identified with certainty*. Q is called a *feasible query set*. We also say that Q *verifies* T.

Definition 1.1. For an instance of MST-U an edge set $Q \subset E$ is called a feasible query set if after querying all edges in Q, an MST can be determined with certainty. Q is called an optimal query set if it is a feasible query set of minimum total cost.

We are considering the online version of the problem, i.e. the algorithm's choice of the next query may depend on the outcome of prior queries. To analyze the performance of a solution found by a (randomized) algorithm we compute the competitive ratio between the (expected) query cost of the algorithm's solution and the cost of an optimal query set.

Definition 1.2. Let I be an instance of MST-U. By $OPT(I)$ we denote the cost of an optimal query set for instance I. We say that a feasible solution for I which consists of a feasible query set Q achieves competitive ratio $c \geq 1$ if

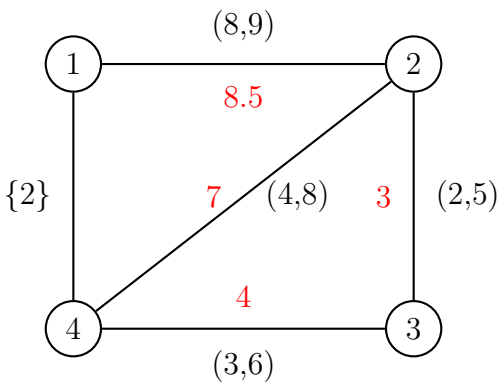
$$\frac{\sum_{e \in Q} q_e}{OPT(I)} \leq c.$$

For an algorithm ALG we denote by $ALG(I)$ the cost of the query set which the algorithm outputs when applied to I. We say that ALG achieves competitive ratio $c \geq 1$ or is c- (update) competitive if

$$\frac{ALG(\tilde{I})}{OPT(\tilde{I})} \leq c$$

for all instances $\tilde{I}$ of MST-U.

Example 1.1. Consider the following example of an instance of MST-U, where the uncertainty sets are indicated as intervals next to the edges in the graph together with the underlaying realization of edge weights, which is depicted in red. The query cost of each edge is one.



Note that without querying $\{1, 2\}$ we know that it has strictly larger weight than all edges in the cycle (1234). Hence it can be disregarded. Assume we test $\{2, 3\}$ and $\{3, 4\}$, revealing the weights 3 and 4 respectively. Thus $\{2, 4\}$ has largest weight in the cycle (234) and can be excluded.
An optimal solution would query $\{2, 4\}$ and see that its weight exceeds the suprema of $A_{\{2,3\}}$ and $A_{\{3,4\}}$. Thus the competitive ratio of this solution is 2.

Figure 1.1: An instance of MST-U: The realization of edge weights is depicted in red. All query costs equal 1.

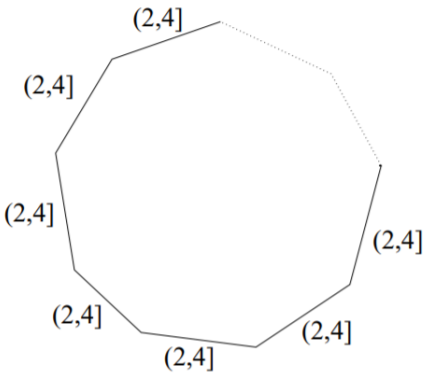
2 Deterministic Algorithms

We will see that for arbitrary uncertainty sets, no deterministic algorithm can achieve a constant competitive ratio. For the case where uncertainty sets are either trivial or do not contain neither their infimum nor their supremum, the algorithm U-RED presented in Section 2.2 was introduced by Erlebach et al. in [7] and has competitive ratio 2. This can be shown to be optimal.

2.1 Performance of deterministic algorithms

For some instances it is enough to identify a single edge that needs to be excluded (e.g. the maximum weight edge in a cycle) or included (such as the minimum weight edge in a cut). If we allow arbitrary uncertainty sets, an optimal solution might be able to identify this edge with a single query, while without knowledge of the true edge weights an algorithm might need to query all of the edges. Erlebach et al. [7] give the following example:

Example 2.1. Consider a cycle C with k edges such that all edges have uncertainty set $(2, 4]$ and all but one have weight 3, while the remaining edge $\bar{e}$ has weight 4.



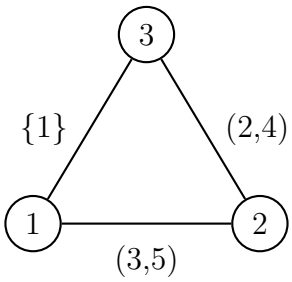
An optimal solution queries only edge $\bar{e}$. As 4 is the largest possible weight, this is sufficient to know that $\bar{e}$ has maximum weight and can be excluded. For any order however, in which an algorithm ALG queries the edges in C, there is a realization such that $\bar{e}$ is the last edge queried. This leads to a competitive ratio of k.

Figure 2.1: No algorithm has constant competitive ratio. ([7], page 286)

Although it would be sufficient to restrict uncertainty sets to singletons and sets that do not contain neither their supremum nor their infimum, we will, for the sake of simplicity, assume that from now on uncertainty sets are either singletons or open

intervals. This means that without further notion all results in the remaining master's thesis are meant to hold for instances where uncertainty sets are trivial or open intervals.
However, we still cannot achieve a competitive ratio better than two. This was observed by [7] and can be seen in the following example:

Example 2.2. Consider the following triangle where the edge $\{1, 3\}$ lies in any MST but without querying an edge it is impossible to tell whether $\{2, 3\}$ or $\{1, 2\}$ is included in an MST.



First assume that an algorithm decides to query edge $\{2, 3\}$ first and consider the following realization of edge weights: $\bar{w}_{\{2,3\}} = 3.5$, $\bar{w}_{\{1,2\}} = 4.5$. As $\bar{w}_{\{2,3\}} \in A_{\{1,2\}}$, the algorithm has to query $\{1, 2\}$ too. An optimal solution however, only queries $\{1, 2\}$ to see that it has maximum weight.
Now assume that the algorithm decides to query edge $\{1, 2\}$ first while the edge weights are as follows: $\bar{w}_{\{2,3\}} = 2.5$, $\bar{w}_{\{1,2\}} = 3.5$.

Figure 2.2: Lower bound computation for deterministic algorithms

Again $\bar{w}_{\{1,2\}}$ lies in $A_{\{2,3\}}$, so the algorithm has to query $\{2, 3\}$ too. An optimal solution however, only queries $\{2, 3\}$ to see that it has less weight than $\{1, 2\}$. So in both cases there exists an instance for which the algorithm produces a solution with two times the optimal query cost.

2.2 Uniform query costs: the algorithm U-RED

In this section we consider instances with query cost $q_e = 1$, $e \in E$, i.e. we want to minimize the number of queries needed to find an MST.

Definition 2.1. A set $W \subset E$ of edges of G is called a witness set, if any set of updates that suffices to verify that a specific tree is a minimum spanning tree must update at least one of the uncertainty sets of edges in W.

An algorithm that repeatedly finds a witness set and updates all elements in it, until the solution can be computed with certainty is called a *witness algorithm*. The following result is known about witness algorithms and was proved in a different setting by Bruce et al. [2] and carried over to the setting of MST-U by [7].

Theorem 2.1. ([7], page 282) *If the size of any witness set used by the witness algorithm is at most k, then the witness algorithm is k-update competitive.*

The algorithm relies on the following well known essential property of minimum spanning trees:

Proposition 2.1. *Let $G = (V, E)$ be a weighted graph with edge weights w_e, $e \in E$ and let C be a cycle in G. If there exists an edge $e \in C$ with $w_e \geq w_{e'}$ for all $e' \in C - \{e\}$, then there exists an MST of G that does not contain e.*

The algorithm U-RED is based on Kruskal's algorithm. Edges are added in order of increasing lower limit L_e. More precisely, the order relation is defined as follows:

Definition 2.2. For two edges $e, f \in E$, $e < f$ if $L_e < L_f$ or $L_e = L_f$ and $U_e < U_f$. We have $e \leq f$ if $e < f$ or $e = f$.

Once a cycle is created, we try to remove an edge which has largest weight for all possible realisations of edge weights. Such edges are called always maximal:

Definition 2.3. Let C be a cycle in G. We say the edge $e \in C$ is an always maximal edge in C if $L_e \geq U_c$ for all $c \in C - \{e\}$.

If we cannot find an always maximal edge, the algorithm finds a witness set of size 2 among the edges in the cycle, consisting of an edge f with largest upper limit and an edge g such that A_g intersects A_f (g exists, otherwise f would be always maximal.) The algorithm updates these two edges and restarts the procedure of adding edges to the empty tree. By showing, that the above choice of f and g is indeed a witness set, the competitive ratio of 2 follows from Theorem 2.1. The description of algorithm U-RED is presented in Algorithm 1.

```
1  Index all edges such that e_1 ≤ e_2 ≤ ... ≤ e_m;
2  Let Γ := ∅ ;
3  for i ← 1 to m do
4  |  Add e_i to Γ ;
5  |  if Γ has a cycle C then
6  |  |  if C contains an always maximal edge e then
7  |  |  |  delete e from Γ
8  |  |  else
9  |  |  |  let f ∈ C such that U_f = max{U_c | c ∈ C};
10 |  |  |  let g ∈ C − {f} such that U_g > L_f;
11 |  |  |  update f and g;
12 |  |  |  restart the algorithm
```

Algorithm 1: The algorithm U-RED for MST-U with uniform query costs (adapted from [7], page 283)

Remark 2.1. U-RED runs in polynomial time. Note that the algorithm is (re-)started $O(m)$ times because prior to every restart two edges are queried and after querying all edges we can identify an MST with certainty. Sorting the edges can be done in $O(m\log(m)) = O(m\log(n))$ time. During each iteration of the for-loop we have to check for acyclicity or find a cycle and identify f and g, each of which can be done in $O(n)$ time. Finding an always maximal edge in a cycle requires finding an edge with the largest upper limit among edges in the cycle and comparing its lower limit with the upper limit of the other cycle edges. This also requires $O(n)$ time. Thus a naive analysis yields a time complexity of $O(m^2n)$.

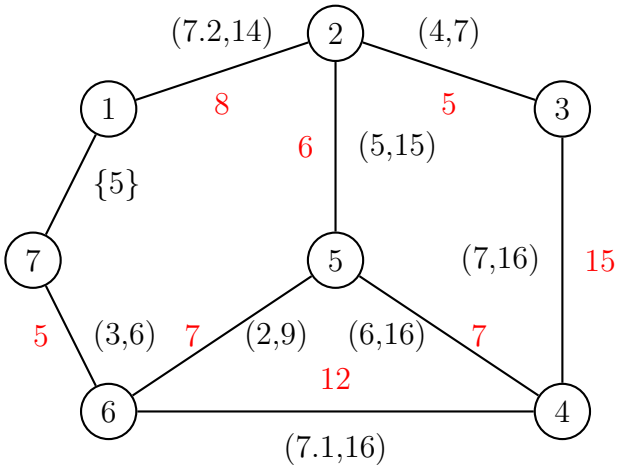
Erlebach et al. [7] show the following essential property of their algorithm.

Lemma 2.2. *The edges f and g in line 9 and 10 of U-RED form a witness set.*

This, along with Theorem 2.1, guarantees a competitive ratio of 2:

Theorem 2.3. ([7], page 285) *The algorithm U-RED is 2-update competitive.*

Example 2.3. We will demonstrate by an example how the algorithm U-RED works. Consider the graph G with uncertainty sets as in the figure below.



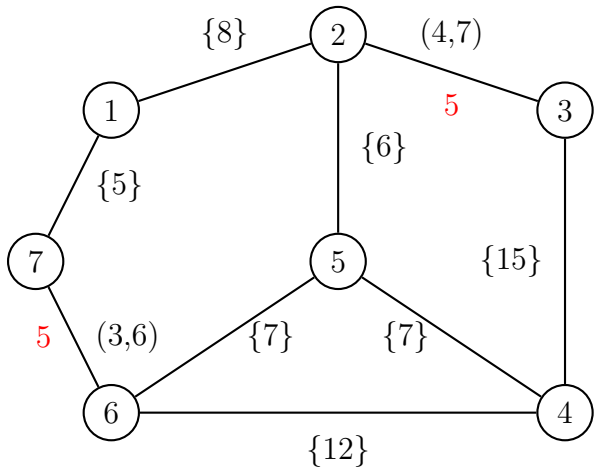
For the sake of notation let $e_1 := \{5,6\}$, $e_2 := \{6,7\}$, $e_3 := \{2,3\}$, $e_4 := \{1,7\}$, $e_5 := \{2,5\}$, $e_6 := \{4,5\}$, $e_7 := \{3,4\}$, $e_8 := \{4,6\}$, $e_9 := \{1,2\}$ and $\bar{w}_i := \bar{w}_{e_i}$, $A_i = A_{e_i}$ for $i = 1,..,9$. Note that prior to the first iteration we have $e_1 \leq e_2 \leq ... \leq e_9$ with respect to the ordering in Definition 2.2 but this might change during the execution of the algorithm.

Figure 2.3: An instance of MST-U with uniform query costs. The realisation of edge weights is depicted in red.

Iteration 1: $e_1 \leq e_2 \leq ... \leq e_9$. Let $\Gamma := \emptyset$.
For $i = 1,..,6$ we simply add e_i to Γ without closing a cycle. Once we add e_7 to Γ, Γ contains the cycle $C := (2,3,4,5)$. Then $f := e_7$ is an edge with largest upper limit among edges in C and the uncertainty set of $g := e_6$ intersects A_7. Hence we update both edges, such that $A_7 = \{15\}$ and $A_6 = \{7\}$. We restart the algorithm.

Iteration 2: $e_1 \leq e_2 \leq e_3 \leq e_4 \leq e_5 \leq e_6 \leq e_8 \leq e_9 \leq e_7$. Let $\Gamma := \emptyset$.
We add e_1, e_2, e_3, e_4, e_5 and e_6 to Γ without closing a cycle. Then we add e_8 to Γ such that Γ contains the cycle $C := (4, 5, 6)$. Then $f := e_8$ has the largest upper limit among edges in C and the uncertainty set of $g := e_1$ intersects A_8. Hence U-RED updates both edges, such that $A_8 = \{12\}$ and $A_1 = \{7\}$ and restarts the algorithm.

Iteration 3: Now $e_2 \leq e_3 \leq e_4 \leq e_5 \leq e_6 \leq e_1 \leq e_9 \leq e_8 \leq e_7$. Let $\Gamma := \emptyset$.
Again e_2, e_3, e_4, e_5, e_6 and e_1 are added to Γ without closing a cycle. Then e_9 is added to Γ and Γ contains the cycle $C := (1, 2, 5, 6, 7)$. Then $f := e_5$ has the largest upper limit among the edges in C and the uncertainty set of $g := e_9$ intersects A_5. Once we have updated the edges, we get $A_5 = \{6\}$ and $A_9 = \{8\}$ such that the instance is now as in the figure below. The algorithm is restarted.



Iteration 4: $e_2 \leq e_3 \leq e_4 \leq e_5 \leq e_1 \leq e_6 \leq e_9 \leq e_8 \leq e_7$. Let $\Gamma := \emptyset$.
Now we add e_2, e_3, e_4, e_5, e_1 and e_6 to Γ without closing a cycle. Then e_9 is added to Γ and Γ contains the cycle $C := (1, 2, 5, 6, 7)$. Then $f := e_9$ has the largest upper limit among the edges in C and is always maximal. Thus e_9 is deleted from Γ.

Figure 2.4: The instance prior to the last iteration of U-RED: Each cycle has an always maximal edge.

When e_8 is added to Γ, it closes the cycle $C := (4, 5, 6)$. Now $f := e_8$ is always maximal in C and is deleted from Γ. Finally we add e_9 to Γ, which closes cycle $C := (2, 3, 4, 5)$. The edge $f := e_9$ is always maximal in C and needs to be deleted from Γ.

Thus U-RED has found the minimum spanning tree with edge set $\{e_i | i = 1, ..., 6\}$ after having queried the edges in $\{e_1, e_5, e_6, e_7, e_8, e_9\}$. Note that e_7, e_8 and e_9 would have been always maximal in the respective cycles after only updating e_1, e_5, e_6. Hence U-RED achieves the claimed worst-case competitive ratio of 2.

2.3 Non-uniform query costs

In this section we will present the algorithm BALANCE introduced by Megow et al. in [15]. BALANCE works for general query costs and achieves an optimal competitive ratio of 2.

2.3.1 Framework of BALANCE

The algorithm BALANCE as well as the algorithm RANDOM in the next chapter start out with a certain spanning tree T_0, which is considered to be the first candidate for a minimum spanning tree. In each iteration i we add an edge f_i to the tree, in order of increasing lower limit (more precisely, in the order as in Definition 2.2.) The edge f_i closes a unique cycle C_i. We then try to identify an edge of maximum weight in C_i. In order to decide which edges should be queried it is essential to better understand MST-U on cycles. Once we have found a maximum weight edge e_i, this edge can be disregarded as there exists an MST of G which does not contain e_i. Hence for the sequence of subgraphs $(V, E_i)_{i=0}^k$, where $E_0 = T_0$, $E_i = E_{i-1} + f_i$ and $E_k = E$, we find a nested sequence of query sets $\emptyset = Q_0 \subset Q_1 \subset ... \subset Q_k$, such that Q_i verifies that $T_i := T_{i-1} + f_i - e_i$ is a minimum spanning tree of (V, E_i), $i = 0, ..., k$.

2.3.2 Lower Limit Tree

A *lower limit tree* is the edge set of a Minimum Spanning Tree in G, where all edge weights are set equal to the lower limit of the edge's uncertainty set. An *upper limit tree* is defined analogously. Let T_L, T_U denote a lower and an upper limit tree respectively.

Remark 2.2. Megow et al. [15] show that all edges in $T_L \backslash T_U$ with non-trivial uncertainty sets lie in any feasible query set.

Thus the instance can be preprocessed such that $T_L \backslash T_U$ consists of trivial edges only. Now assume $T_L \setminus T_U \neq T_U \setminus T_L$. The sum of upper limits of edges in $T_U \setminus T_L$ cannot be less than the sum of weights of edges in $T_L \setminus T_U$, otherwise $T_U \setminus T_L \cup (T_U \cap T_L) = T_U$ would be the edge set of a lower limit tree of smaller weight than T_L. Hence $\sum_{e \in T_U \setminus T_L} U_e = \sum_{e \in T_L \setminus T_U} \bar{w}_e = \sum_{e \in T_L \setminus T_U} U_e$ which is why we can w.l.o.g. assume $T_L = T_U$. We choose our initial tree T_0 to be T_L.

Throughout the discussion of BALANCE (as well as all algorithms which are based on the same framework, e.g. RANDOM) we agree on the following notation:

Definition 2.4. Let $R := E \backslash T_L =: \{f_1, ..., f_{m-n+1}\}$, such that $f_1 \leq f_2 \leq ... \leq f_{m-n-1}$, where $\leq$ is the ordering as defined in Definition 2.2. Set $T_0 := T_L$, $E_0 := T_0$ and $Q_0 := \emptyset$. For $i = 1, ..., m-n+1$ we denote by $E_i := E_{i-1} \cup \{f_i\} = T_L \cup \{f_1, ..., f_i\}$ the set of edges which the algorithm has considered after i iterations and by Q_i the set of all edges which the algorithm has queried during the first i iterations. Moreover, let T_i

be the minimum spanning tree which the algorithm has verified for the graph (V, E_i) after having queried the edges in Q_i during the first i iterations, $i = 1, ..., m - n + 1$. Finally, by C_i we denote the cycle closed during the i'th iteration when the algorithm adds f_i to T_{i-1}.

2.3.3 Finding a maximum weight edge in a cycle

In a cycle C an edge f with largest upper limit is a candidate for a maximum weight edge, along with all the edges e in C with A_e intersecting A_f. By the choice of $T_0 = T_L$ and the fact that $T_L = T_U$, it turns out that the edge f_i has largest upper limit in the cycle C that it closes with T_{i-1}.

Lemma 2.4. *Let $i \in \{1, ..., m - n + 1\}$. Let C_i be the cycle that f_i closes when added to T_{i-1}. Then f_i has largest upper limit among all edges in C_i.*

Proof. We show that for all $i = 0, ...m-n$ any edge not in E_i has largest upper limit in the cycle it closes with T_i. The assertion is true for $i = 0$ because $T_0 = T_L = T_U$. Now assume it is true for i and show that it holds for $i + 1$. Let f be an edge in $E \setminus E_{i+1}$. If the cycle C that f closes with T_{i+1} is the same as the cycle C' that f closes with T_i, the assertion is trivially true. Thus we assume that $C \neq C'$. This means that when adding f_{i+1} to T_i an edge $e \in C'$ is removed and replaced by f_{i+1}, which only happens if f_{i+1}'s weight as well as the weight or upper limit of all other edges in C_{i+1} are smaller than the upper limit of the deleted edge e. As C consists only of edges in C' and C_{i+1}, f has largest upper limit in C. □

Note that even after querying all edges with uncertainty sets intersecting A_{f_i}, f_i could still turn out to have larger weight than any of these edges and thus not be contained in any MST. Hence, whenever we decide to keep f_i as a tree edge, we must have queried f_i. In order to find a maximum weight edge in C_i it is thus sufficient to only consider those edges with uncertainty interval intersecting A_{f_i} that lie in T_L.

Definition 2.5. Let f_i be the edge added in iteration i and C_i the unique cycle it closes with T_{i-1}. Then we define the neighbor set of f_i as

$$X(f_i) = \{g \in C_i \cap T_L | A_g \cap A_{f_i} \neq \emptyset\}.$$

In each iteration i we try to find a maximum weight edge in the cycle C_i that f_i closes. If $X(f_i)$ is not empty (otherwise f_i is always maximal) we either query f_i or all edges in $X(f_i)$. In fact any feasible query set must contain f_i or all edges in $X(f_i)$.

Lemma 2.5. *Given a graph G with uncertain edge weights and a realization of edge weights, let T_L be its lower limit tree. Let T_{i-1} be a verified MST for $G_{i-1} = (V, E_{i-1})$ and let C_i be the cycle closed by adding edge f_i to T_{i-1}. Then any feasible query set Q of $G_i = (V, E_i)$ contains f_i or $X(f_i)$.*

Proof. Assume we want to verify an MST of G_i that contains f_i. Then Q must contain f_i because otherwise there are possible realizations for which f_i has strictly largest weight in C_i as it has the largest upper limit in C_i.
Now assume we want to verify an MST T of G_i with $f_i \notin T$. Then f_i must have largest weight in the cycle C' it closes with T. Note that T has to be an MST of G_{i-1} as well, thus f_i must have largest weight in C_i. If $f_i \notin Q$, then the only possible way to guarantee that f_i has largest weight in C_i is to show that the weights of all other edges do not lie in A_{f_i}. Thus we have to test all neighbors of f_i. □

So far we have shown that any feasible query set for (V, E_i) contains f_i or all edges in $X(f_i)$.
However, we want that any feasible query set of the entire graph G contains these edges as well. This is guaranteed by the following lemma, which was shown by [15]:

Lemma 2.6. *Let $i \in \{0, ..., m-n+1\}$. Given a feasible query set Q for the uncertainty graph $G = (V, E)$, then the set $Q \cap E_i$ is a feasible query set for $G_i = (V, E_i)$.*

Remark 2.3. If we still do not know which edge has maximum weight in C_i we go on querying edges in order of decreasing upper limit until we find a maximum weight edge. These edges were shown by [15] to be part of any feasible query set as well, due to the fact that f_i has largest upper and lower limit in C_i.

2.3.4 Core of the algorithm BALANCE

To decide whether to query f_i or all of its neighbors, the algorithm proceeds as follows: "By default" we query the neighbor set $X(f_i)$. We want to query the neighbors because they might reappear in neighbor sets of later iterations. However, we want to make sure that we maintain a competitive ratio of 2. This is done by assigning an *edge potential* y_e to every edge $e \in T_L$, which is initially set to 0. In each iteration i, the potential y_e of each edge e in $X(f_i)$ is either raised to a common level $t(f_i)$ or remains the same, if y_e is already greater than $t(f_i)$. The value of $t(f_i)$ is obtained by maximizing $t \leq 1$ s.t. $\sum_{e \in X(f_i)} q_e \cdot \max\{0, t - y_e\} \leq q_{f_i}$ and can be computed by applying the subroutine displayed in Algorithm 3. Note that $t(f_i) \geq 0$ and that the choice of $t(f_i)$ guarantees for each edge e that $0 \leq y_e \leq 1$ holds at all times. The edge potential of e can be interpreted as the share of the query cost of e which has already been balanced by the query cost in the optimal solution and is increased whenever e is part of a neighbor set. Thus we make sure that edges which appear in many neighbor sets are queried.

A formal description of the algorithm BALANCE can be found in Algorithm 2.

Remark 2.4. Note that BALANCE runs in polynomial time. The preprocessing requires solving two MST instances repeatedly but at most m times because prior to

input : An instance of MST-U with graph $G = (V, E)$, uncertainty sets A_e and query costs q_e, $e \in E$
output: A feasible query set Q

Preprocess the instance such that $T_L = T_U$ and set $\Gamma := T_L$;
Index the edges in $R := E \setminus T_L$ s.t $f_1 \leq ... \leq f_{m-n+1}$;
Initialize $Q = \emptyset$;
Initialize $y_e = 0$ for all $e \in T_L$;
for $i \leftarrow 1$ **to** $m - n + 1$ **do**
  Add f_i to Γ and let C_i be the unique cycle closed;
  Let $X(f_i)$ be the set of edges $g \in T_L \cap C_i$ with $U_g > L_{f_i}$;
  if $X(f_i)$ *is not empty* **then**
    Maximize the threshold $t(f_i) \leq 1$ s.t. $\sum_{e \in X(f_i)} q_e \cdot \max\{0, t(f_i) - y_e\} \leq q_{f_i}$;
    Increase edge potentials $y_e := \max\{t(f_i), y_e\}$ for all $e \in X(f_i)$;
    if $t(f_i) < 1$ **then**
      Add f_i to Q and query it.
    else
      Add all edges in $X(f_i)$ to Q and query them.
  while *no edge in the cycle* C_i *is always maximal* **do**
    Query the unqueried edge $e \in C_i \setminus Q$ with maximum U_e and add it to Q.
  Delete an always maximal edge from Γ

Algorithm 2: The algorithm BALANCE for MST-U with general query costs (adapted from [15], page 1223 and 1231)

every new computation of a lower and an upper limit tree we query at least one edge. This yields $O(m^2 \log(n))$ time with e.g. Kruskal's algorithm. Sorting the edges in R can be done in $O(m \log(n))$ time. We also sort all edges in order of non-increasing upper limit. We update this sorting whenever we make an edge query. This takes $O(m \log(n))$ time for the initial sorting and $O(\log(n))$ time per query. During the i'th iteration, finding C_i and $X(f_i)$ can be done in $O(n)$ time. The value of $t(f_i)$ can be computed with the subroutine in Algorithm 3.

This subroutine requires sorting the edge potentials ($O(n \log(n))$ time) and $O(k)$ constant time operations (i.e. $O(n)$ time.)
After having queried f_i or all of $X(f_i)$, we need to check whether $\bar{w}_{f_i} \geq U_e$ or $\bar{w}_e (= U_e) \leq L_{f_i}$ for all $e \in X(f_i)$ respectively. Both can be done in constant time due to the sorting with respect to the upper limits. Once we have updated f_i, there exists an always maximal edge in C_i iff the edge with maximum upper limit is trivial. (Hence this can be checked in constant time.) The time needed for edge queries, up-

input : An edge f_i with neighbor set $X(f_i)$ and edge potentials y_e for $e \in X(f_i)$
output: $t(f_i)$

1 Sort the edges in $X(f_i) = \{e_1, ..., e_k\}$ such that $y_{e_1} \leq ... \leq y_{e_k}$;
2 **for** $j \leftarrow k$ **to** *1* **do**
3 | Compute t such that $\sum_{l=1}^{j} q_{e_l}(t - y_{e_l}) = c_{f_i}$;
4 | **if** $t - y_{e_j} \geq 0$ **then**
5 | | STOP, output $t(f_i) := \min\{1, t\}$

Algorithm 3: Computing $t(f_i)$

dating the ordering of upper limits or checking whether there is an always maximal edge is not counted per iteration, because it sums up to $O(m \log(n))$ over all iterations. The run-time per iteration without these operations can thus be bounded by $O(n \log(n))$. Hence the overall run-time for the algorithm, without the preprocessing, is bounded by $O(m \cdot n \log(n))$ time and by $O(m^2 \log(n))$ time if the preprocessing is included.

Theorem 2.7. ([15], page 1230) *The algorithm BALANCE achieves competitive ratio 2.*

Proof. To compare the query cost of the set Q found by BALANCE with the cost of an optimal query set Q^* we consider three subsets of Q separately: edges which also lie in Q^*, edges in $T_L \cap Q$ which do not lie in Q^* and edges in $R = E \setminus T_L$ which are queried by BALANCE but are not in the optimal query set Q^*. Then

$$\sum_{e \in Q} q_e = \sum_{e \in Q \cap Q^*} q_e + \sum_{e \in (Q \cap T_L) \setminus Q^*} q_e + \sum_{i: f_i \in (Q \cap R) \setminus Q^*} q_{f_i}.$$

Consider first edges in $(Q \cap R) \setminus Q^*$. An edge in $Q \cap R$ is queried, thus we know that $t(f_i) < 1$ is chosen such that we obtain equality in line 9 of Algorithm 2. We denote by y_e^i the potential of e at the beginning of the i'th iteration. Hence $t(f_i) = y_e^{i+1}$ for all edges whose potential changes in iteration i and we have

$$\sum_{i: f_i \in (Q \cap R) \setminus Q^*} q_{f_i} = \sum_{i: f_i \in (Q \cap R) \setminus Q^*} \sum_{e \in X(f_i)} (y_e^{i+1} - y_e^i) \cdot q_e.$$

Note that even though we do not sum over all iterations, for a fixed e the sum

$$\sum_{\substack{i: f_i \in (Q \cap R) \setminus Q^* \\ e \in X(f_i)}} (y_e^{i+1} - y_e^i)$$

can be bounded by y_e, where y_e is the potential of e after the last iteration, because the potential never decreases. As for any $f_i \in R \setminus Q^*$ the neighbor set $X(f_i)$ lies entirely in Q^*, we can sum over all edges $e \in T_L \cap Q^*$. Hence

$$\sum_{i:f_i \in (Q \cap R) \setminus Q^*} q_{f_i} \leq \sum_{e \in T_L \cap Q^*} y_e \cdot q_e \leq \sum_{e \in T_L \cap Q^*} q_e$$

Now we consider the query cost of $(Q \cap T_L) \setminus Q^*$. Note that for $e \in T_L \cap Q$ we have $y_e = 1$. Again we use that for $e \in X(f_i)$ with $e \notin Q^*$ we know by Lemma 2.5 that f_i lies in Q^*. Thus:

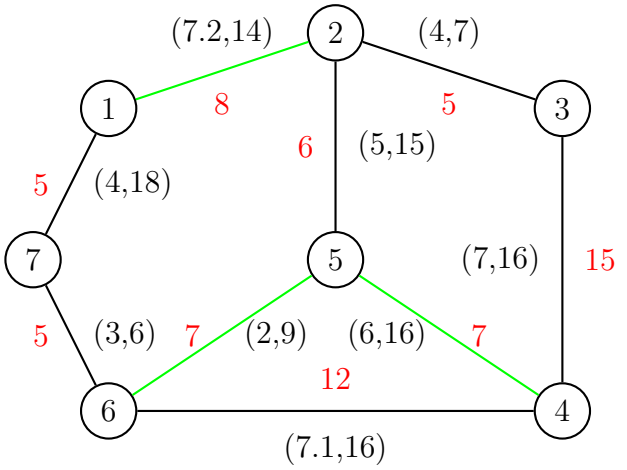
$$\begin{aligned} \sum_{e \in (Q \cap T_L) \setminus Q^*} q_e &\leq \sum_{e \in T_L \setminus Q^*} q_e \cdot y_e = \sum_{e \in T_L \setminus Q^*} q_e \sum_{\substack{i:f_i \in R \cap Q^* \\ e \in X(f_i)}} (y_e^{i+1} - y_e^i) \\ &\leq \sum_{i:f_i \in R \cap Q^*} \sum_{e \in X(f_i)} q_e (y_e^{i+1} - y_e^i) \leq \sum_{i:f_i \in R \cap Q^*} q_{f_i}. \end{aligned}$$

Finally the cost of edges in $Q \cap Q^*$ can simply be bounded by the cost of all of Q^*. Hence

$$\sum_{e \in Q} q_e \leq \sum_{e \in Q^*} q_e + \sum_{i:f_i \in R \cap Q^*} q_{f_i} + \sum_{e \in T_L \cap Q^*} q_e = 2 \sum_{e \in Q^*} q_e.$$

□

Example 2.4. We will now demonstrate by an example how the algorithm BALANCE works. Consider the graph G with uncertainty sets as in the figure below. We use the same edge names $(e_1, ..., e_9)$ and notation for the uncertainty sets as in Example 2.3, as well as $q_j := q_{e_j}$, $j = 1, ..., 9$ for the query costs which all equal 1, except for $q_1 = q_6 = 2$ and $q_9 = 1.8$.



Preprocessing: $\{e_1, e_2, e_3, e_4, e_5, e_6\}$ is the edge set of a lower limit tree and $\{e_2, e_3, e_1, e_9, e_5, e_6\}$ is the edge set of an upper limit tree. Thus by Remark 2.2 we know that e_4 lies in any feasible query set, thus we update e_4. We set $T_L (:= T_U) := \{e_1, e_2, e_3, e_4, e_5, e_6\}$ and $f_1 := e_7, f_2 := e_8, f_3 := e_9$.

Figure 2.5: An instance of MST-U with non-uniform query costs. The realisation of edge weights is depicted in red. The query costs are equal to one except for $q_1 = q_6 = 2, q_9 = 1.8$ (in green).

Now we set $\Gamma := T_L, Q := \emptyset$ and $y_j := y_{e_j} = 0$ for all $j \in \{1, ..., 6\}$.

i=1: The edge e_7 closes the cycle $C_1 = (2, 3, 4, 5)$ in $\Gamma := T_L \cup \{e_7\}$. The uncertainty sets A_5 and A_6 intersect A_7, hence $X_{e_7} = \{e_5, e_6\}$.
Then $t(e_7) = \arg\max_{t \in [0,1]} \{q_5 \cdot \max\{0, (t - y_5)\} + q_6 \cdot \max\{0, (t - y_6)\}\} = \frac{1}{3}$ and we have $q_5(t(e_7) - y_5) + q_6(t(e_7) - y_6) = 1 \cdot (\frac{1}{3} - 0) + 2 \cdot (\frac{1}{3} - 0) = 1 = q_7$. We increase

$$y_5 := \frac{1}{3}, y_6 := \frac{1}{3}.$$

As $t(e_7) = \frac{1}{3} < 1$, we query $f_1 = e_7$, i.e. $Q := \{e_7\}$. This yields $\bar{w}_7 = 15 \in A_6$, hence we cannot identify a maximum weight edge in C_1. Thus, we query the edge e_6, which has now the largest upper limit among edges in C_1 and add it to Q, i.e. $Q := \{e_6, e_7\}$. Now we know, that e_7 is a maximum weight edge in C_1, we remove it from Γ, i.e $\Gamma := T_L$.

i=2: The edge e_8 closes the cycle $C_2 = (4, 5, 6)$ in $\Gamma := T_L \cup \{e_8\}$. The uncertainty set of A_1 intersects A_8 and we have $X_{e_8} = \{e_1\}$. BALANCE sets $t(e_8) := 0.5$ and we have $q_1(t(e_8) - y_1) = 2 \cdot (0.5 - 0) = 1 = q_8$. BALANCE sets

$$y_1 := \frac{1}{2}.$$

Moreover, we set $Q := \{e_6, e_7, e_8\}$ and query e_8, revealing $\bar{w}_8 = 12$, which is sufficient to see that e_8 has maximum weight among the edges in C_2. Set $\Gamma := T_L$.

i=3: The edge e_9 closes the cycle $C_3 = (1, 2, 5, 6, 7)$ in $\Gamma := T_L \cup \{e_9\}$. The uncertainty sets of e_5 and e_1 intersect A_9, hence $X(e_9) := \{e_1, e_5\}$. Now in this iteration $t(e_9) = 1$ and $q_1(t(e_9) - y_1) + q_5(t(e_9) - y_5) = 2 \cdot (1 - \frac{1}{2}) + 1 \cdot (1 - \frac{1}{3}) = \frac{5}{3} \leq q_9$. Hence we query the edges in the neighbor set which is sufficient to identify e_9 as a maximum weight edge in C_3. Set $Q := \{e_1, e_5, e_6, e_7, e_8\}$ and output $\Gamma := T_L$.
The cost of this solution for the preprocessed instance is 2+1+2+1+1 = 7, while an optimal solution queries e_6, e_5 and e_1, which yields a query cost of 5. This yields a competitive ratio of $\frac{7}{5} = 1.4$ for the preprocessed instance and $\frac{8}{6} \approx 1.33$ for the original instance.

Now let us see how BALANCE performs in the uniform query cost case for the instance in Example 2.3.

Example 2.5. Consider the instance of MST-U and the notation of Example 2.3. Here we already have $T_L = T_U = \{e_1, ..., e_6\}$ without prior queries. As for $f_1 = e_7$, $f_2 = e_8$ and $f_3 = e_9$ the cycles stay the same as in the non-uniform case, we will just briefly state the neighbor sets, how the potentials increase and which edges are added to Q.
First set $Q := \emptyset$ and $y_j := y_{e_j} = 0$ for all $j \in \{1, ..., 6\}$.

i=1: $X(e_7) = \{e_5, e_6\}$ as in the non-uniform case. We have $t(e_7) = \frac{1}{2}$ and $q_5(t(e_7) - y_5) + q_6(t(e_7) - y_6) = 1 \cdot (\frac{1}{2} - 0) + 1 \cdot (\frac{1}{2} - 0) = 1 = q_7$. We increase

$$y_5 := \frac{1}{2}, y_6 := \frac{1}{2}.$$

As $t(e_7) = \frac{1}{2} < 1$, we query $f_1 = e_7$ and then e_6, as querying e_7 does not turn out to be sufficient to see that e_7 is a maximum weight edge in C_1. So $Q := \{e_6, e_7\}$.

i=2: Again the neighbor set $X(e_8) = \{e_1\}$ stays the same as in the non-uniform case. BALANCE sets $t(e_8) := 1$ which satisfies $q_1(t(e_8) - y_1) = 1 \cdot (1 - 0) = 1 = q_8$. The edge potential of y_1 is now

$$y_1 := 1.$$

Moreover, as $t(e_8) = 1$, we set $Q := \{e_1, e_6, e_7\}$ and query e_1, revealing $\bar{w}_1 = 7$, which is sufficient to see that e_8 has maximum weight among the edges in C_2.

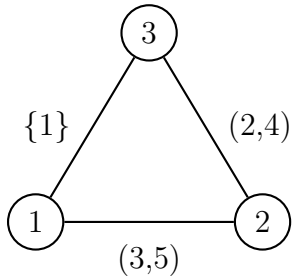
i=3: Now the neighbor set of edge e_9 only contains e_5. Thus we have $t(e_9) = 1$ such that $q_5(t(e_9) - y_5) = 1 \cdot (1 - \frac{1}{2}) = \frac{1}{2} \leq 1 = q_9$. Hence we query e_5, which makes it clear that e_9 is a maximum weight edge in C_3. Set $Q := \{e_1, e_5, e_6, e_7\}$ and output $\Gamma := T_L$.

Remember that U-RED queried 6 edges when applied to this instance. Hence, for this instance BALANCE performs better than U-RED. This is not necessarily the case in general.

3 Randomization

3.1 Lower bound

Consider again Example 2.2. An algorithm queries edge $\{2,3\}$ first with probability p.



If we consider again the realization $\bar{w}_{\{2,3\}} = 3.5$, $\bar{w}_{\{1,2\}} = 4.5$, the expected number of queries is $2p + (1-p)$, as $\bar{w}_{\{2,3\}} \in A_{\{1,2\}}$ and thus the algorithm has to query $\{1,2\}$ too if it queries $\{2,3\}$ first.
Now we assume that the underlying realization is $\bar{w}_{\{2,3\}} = 2.5$, $\bar{w}_{\{1,2\}} = 3.5$. Then the expected number of queries is $p + 2(1-p)$.

Figure 3.1: Lower bound computation for randomized algorithms

As $\max\{2p + (1-p), p + 2(1-p)\}$ is minimal for $p = \frac{1}{2}$, no randomized algorithm can achieve a competitive ratio less than 1.5 when applied to this instance. The lower bound of 1.5 for randomized algorithms was observed by [5].

3.2 Uniform query costs

3.2.1 The algorithm RANDOM

The algorithm RANDOM which was introduced by [15] works similar as BALANCE and uses the same framework. In each iteration i we try to find a maximum weight edge in the cycle C_i that f_i closes. If $X(f_i)$ is not empty (otherwise f_i is always maximal) we either query f_i or all edges in $X(f_i)$. If we still do not know which edge has maximum weight in C_i we go on querying edges in order of decreasing upper limit until we find a maximum weight edge. To decide whether to query f_i or all of its neighbours the algorithm uses randomization. Again each edge e in T_L is assigned an edge potential y_e which is initially set to 0 and can only increase throughout the algorithm. In the setting of RANDOM however, y_e can be interpreted as the probability that e is queried. In each iteration i, a total potential of $\frac{1}{\sqrt{2}}$ is spread among edges in $X(f_i)$ (the choice

of $\frac{1}{\sqrt{2}}$ becomes evident in the analysis of the algorithm.) Thereby the potential y_e of each edge e in $X(f_i)$ is either raised to a common level $t(f_i)$ (yet to be defined) or remains the same, if y_e is already greater than $t(f_i)$.
In the beginning of the algorithm we agree on a bound b that is drawn uniformly at random from $[0, 1]$. Whenever $t(f_i)$ exceeds b, i.e. whenever all edges in $X(f_i)$ have potential at least b, we decide to query all edges in $X(f_i)$. Otherwise we query f_i. This means that an edge $e \in T_L \setminus Q^*$ is queried if its potential exceeds the query bound b, where Q^* denotes an optimal query set. Hence an edge $e \in T_L \setminus Q^*$ is queried with probability $\mathbb{P}[y_e \geq b] = y_e$ and an edge $f_i \in R \setminus Q^*$ is queried with probability $\mathbb{P}[t(f_i) < b] = 1 - t(f_i)$.
Algorithm 4 gives a formal description of the algorithm RANDOM.

input : An instance of MST-U with graph $G = (V, E)$, uncertainty sets A_e, $e \in E$ and uniform query costs
output: A feasible query set Q

Draw b uniformly at random from $[0, 1]$;
Preprocess the instance such that $T_L = T_U$ and set $\Gamma := T_L$;
Index the edges in $R := E \setminus T_L$ by increasing lower limit $f_1, ..., f_{m-n+1}$;
Initialize $Q = \emptyset$;
Initialize $y_e = 0$ for all $e \in T_L$;
for $i \leftarrow 1$ **to** $m - n + 1$ **do**
  Add f_i to Γ and let C_i be the unique cycle closed;
  Let $X(f_i)$ be the set of edges $g \in T_L \cap C_i$ with $U_g > L_{f_i}$;
  if $X(f_i)$ *is not empty* **then**
    Maximize the threshold $t(f_i) \leq 1$ s.t. $\sum_{e \in X(f_i)} \max\{0, t(f_i) - y_e\} \leq \frac{1}{\sqrt{2}}$;
    Increase edge potentials $y_e := \max\{t(f_i), y_e\}$ for all $e \in X(f_i)$;
    if $t(f_i) < b$ **then**
      Add f_i to Q and query it.
    else
      Add all edges in $X(f_i)$ to Q and query them.
  while *no edge in the cycle* C_i *is always maximal* **do**
    Query the unqueried edge $e \in C_i \setminus Q$ with maximum U_e and add it to Q.
  Delete an always maximal edge from Γ

Algorithm 4: The algorithm RANDOM for MST-U with uniform query costs (adapted from [15], page 1223 and 1227)

Analysis of RANDOM

Again we partition the query set Q into the three subsets $Q \cap Q^*$, $(Q \cap R) \setminus Q^*$ and $(Q \cap T_L) \setminus Q^*$, where Q^* is an optimal query set. Similar to the proof of the competitive ratio of BALANCE the expected query cost of each of these sets can be bounded separately. Megow et al. [15] show the following lemmata:

Lemma 3.1. *For any feasible query set Q^* it holds that*

$$\sum_{e \in T_L \setminus Q^*} y_e \leq \frac{1}{\sqrt{2}} \cdot |R \cap Q^*|,$$

where y_e denotes the edge potential of e after an execution of RANDOM.

Lemma 3.2. *For any feasible query set Q^*, it holds that*

$$\sum_{i: f_i \in R \setminus Q^*} (1 - t(f_i)) \leq \frac{1}{\sqrt{2}} \cdot |T_L \cap Q^*|.$$

Hence we obtain the following result about the competitive ratio of RANDOM:

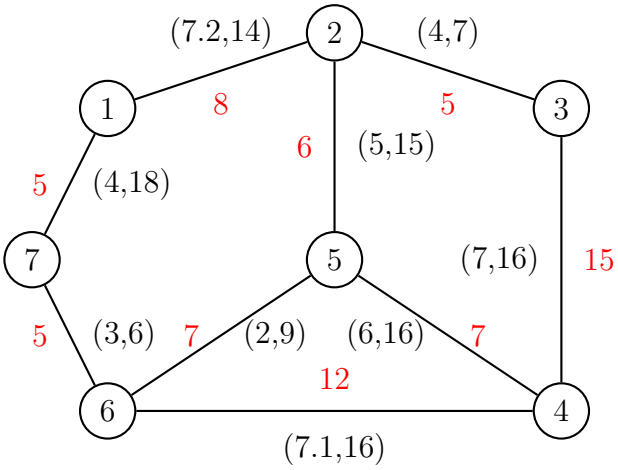
Theorem 3.3. ([15], page 1228) *RANDOM has competitive ratio $1 + \frac{1}{\sqrt{2}}$ (≈ 1.707).*

Proof. By applying Lemma 3.1 and Lemma 3.2 to the overall expected value, we get:

$$\begin{aligned}
\mathbb{E}[|Q|] &= \mathbb{E}[|Q \cap Q^*|] + \mathbb{E}[|(Q \cap T_L) \setminus Q^*|] + \mathbb{E}[|(Q \cap R) \setminus Q^*|] \\
&= \mathbb{E}[|Q \cap Q^*|] + \sum_{e \in T_L \setminus Q^*} \mathbb{P}[e \in Q] + \sum_{i: f_i \in R \setminus Q^*} \mathbb{P}[f_i \in Q] \\
&= \mathbb{E}[|Q \cap Q^*|] + \sum_{e \in T_L \setminus Q^*} y_e + \sum_{i: f_i \in R \setminus Q^*} (1 - t(f_i)) \\
&\overset{3.1}{\leq} \mathbb{E}[|Q \cap Q^*|] + \frac{1}{\sqrt{2}} |R \cap Q^*| + \sum_{i: f_i \in R \setminus Q^*} (1 - t(f_i)) \\
&\overset{3.2}{\leq} \mathbb{E}[|Q \cap Q^*|] + \frac{1}{\sqrt{2}} |R \cap Q^*| + \frac{1}{\sqrt{2}} |T_L \cap Q^*| \\
&\leq |Q^*| + \frac{1}{\sqrt{2}} |R \cap Q^*| + \frac{1}{\sqrt{2}} |T_L \cap Q^*| = (1 + \frac{1}{\sqrt{2}}) \cdot |Q^*|.
\end{aligned}$$

□

Example 3.1. We will now demonstrate by an example how the algorithm RANDOM works. Consider the graph G with uncertainty sets as in the figure below. We use the same edge names $(e_1, ..., e_9)$ and notation for the uncertainty sets as in Example 2.3.



Preprocessing: As in Example 2.4 the preprocessing results in querying e_4, which lies in any feasible query set and yields $T_L(:= T_U) := \{e_1, e_2, e_3, e_4, e_5, e_6\}$. The edges in $R := E \setminus T_L = \{e_7, e_8, e_9\}$ are indexed via $f_1 := e_7$, $f_2 := e_8$ and $f_3 := e_9$ such that $f_1 \leq f_2 \leq f_3$. Assume that drawing b uniformly at random from $[0, 1]$ yields $b = 0.4$.

Figure 3.2: An instance of MST-U with uniform query costs. The realisation of edge weights is depicted in red.

Now we set $\Gamma := T_L, Q := \emptyset$ and $y_j := y_{e_j} = 0$ for all $j \in \{1, ..., 6\}$.

i=1: The edge e_7 closes the cycle $C_1 = (2, 3, 4, 5)$ in $\Gamma := T_L \cup \{e_7\}$. The uncertainty sets A_5 and A_6 intersect A_7, hence $X(e_7) = \{e_5, e_6\}$. Then $t(e_7) = \frac{1}{2\sqrt{2}}$ and we have $(t(e_7) - y_5) + (t(e_7) - y_6) = (\frac{1}{2\sqrt{2}} - 0) + (\frac{1}{2\sqrt{2}} - 0) = \frac{1}{\sqrt{2}}$. We increase

$$y_5 := \frac{1}{2\sqrt{2}}, y_6 := \frac{1}{2\sqrt{2}}.$$

As $t(e_7) = \frac{1}{2\sqrt{2}} \approx 0.35 < 0.4$, we query e_7, i.e. $Q := \{e_7\}$. As $\bar{w}_7 = 15 \in A_6$, we have to query e_6 too, $Q := \{e_6, e_7\}$. Now we know, that e_7 is a maximum weight edge in C_1 and remove it from Γ, i.e $\Gamma := T_L$.

i=2: The edge e_8 closes the cycle $C_2 = (4, 5, 6)$ in $\Gamma := T_L \cup \{e_8\}$. The uncertainty set of A_1 intersects A_8 and we have $X(e_8) = \{e_1\}$. RANDOM sets $t(e_8) := \frac{1}{\sqrt{2}}$ such that $t(e_8) - y_1 = \frac{1}{\sqrt{2}} - 0 = \frac{1}{\sqrt{2}}$. RANDOM sets

$$y_1 := \frac{1}{\sqrt{2}}.$$

Moreover, we set $Q := \{e_1, e_6, e_7\}$ and query e_1, because $t(e_8) = \frac{1}{\sqrt{2}} \approx 0.707 > 0.4$. Now e_8 has become an always maximal edge among the edges in C_2. Set $\Gamma := T_L$.

i=3: The edge e_9 closes the cycle $C_3 = (1, 2, 5, 6, 7)$ in $\Gamma := T_L \cup \{e_9\}$. Note that the neighbor set of e_9 is $\{e_5\}$. Now $t(e_9) = 1$ as $1 - y_5 = 1 - \frac{1}{2\sqrt{2}} \leq \frac{1}{\sqrt{2}}$. Thus we query e_5, as $t(e_9) = 1 \geq 0.4$, remove e_9 from Γ, i.e. $\Gamma := T_L$ and output $Q := \{e_1, e_5, e_6, e_7\}$.

This means that for $b = 0.4$, RANDOM finds a solution with competitive ratio $\frac{5}{4}$.

3.2.2 An optimal randomized algorithm for cactus graphs with uniform query costs

An essential aspect of solving the MST-U is handling edges which appear on multiple cycles. In this section however, we consider the special case of cactus graphs, where a *cactus graph* is a connected graph in which two cycles share at most one vertex. Speaking in the terminology of RANDOM, this means that no edge in T_L will appear in more than one neighbor set. Once we are able to treat cycles separately it turns out that it is possible to achieve an optimal expected competitive ratio of 1.5. Note that RANDOM does not necessarily achieve expected competitive ratio 1.5 for instances of this type: Consider the instance used to compute the lower bound in Section 3.1. T_L consists of the edges $\{1,3\}$ and $\{2,3\}$. When adding $\{1,2\}$ to T_L, potential $\alpha = \frac{1}{\sqrt{2}}$ is distributed among the neighbors of $\{1,2\}$, consisting only of the edge $\{2,3\}$. Hence, $\{2,3\}$ is queried first with probability $\frac{1}{\sqrt{2}}$, while $\{1,2\}$ is queried first with probability $1 - \frac{1}{\sqrt{2}}$. If we face the realization $\bar{w}_{\{2,3\}} = 3.5$ and $\bar{w}_{\{1,2\}} = 4.5$, then the expected competitive ratio equals $1 - \frac{1}{\sqrt{2}} + 2\frac{1}{\sqrt{2}} = 1 + \frac{1}{\sqrt{2}} > 1.5$.

For cactus graphs it is possible to achieve competitive ratio 1.5 in expectation, using the following observation: The framework of RANDOM guarantees that once we have queried f_i or all of $X(f_i)$, querying edges in the current cycle C_i in order of decreasing upper limit until an MST can be identified only adds edges to the query set which must lie in any feasible query set. Thus, if we start by querying f_i then we have queried at most one edge on C_i, which is not in the optimal solution.

Remark 3.1. In an instance of MST-U where the graph G is a cycle, querying $f_i = f_1$ first leads to a competitive ratio of at most $\frac{OPT+1}{OPT}$, which is at most 1.5 unless $OPT = 1$.

Proposition 3.1. *For cactus graphs there exists an algorithm $RANDOM_C$ with competitive ratio at most 1.5, which is optimal. Moreover, if G is a cycle with $T_L = T_U$ in which the edge with maximum upper limit has k neighbors, $RANDOM_C$ achieves competitive ratio $1 + \frac{k}{k^2+1}$.*

Proof. We first consider a graph C which consists of a single cycle. Assume again that we have applied the same preprocessing as for RANDOM and BALANCE and that $T_L = T_U$. Let f be the edge in $E \setminus T_L$ and let $k := |X(f)|$ be the number of neighbors of f in C. Our algorithm starts by querying all edges in $X(f)$ with probability p. With probability $1 - p$, its first step is to query f. Once it has queried $X(f)$ or f, it proceeds as RANDOM and queries edges in order of decreasing upper limit until a maximum weight edge can be identified.
If an optimal solution does not query f, it must query all of the neighbors in $X(f)$

and thus queries k edges. Hence, with probability p, we query the same edges as the optimal solution and achieve competitive ratio 1. With probability $1-p$, we query f and possibly all of the edges in the neighbor set, such that the competitive ratio is at most $\frac{k+1}{k}$. In this case the expected competitive ratio is at most $\frac{kp+(k+1)(1-p)}{k}$.
If an optimal solution queries f, we produce the optimal solution if we start by querying f, i.e. with probability $1-p$. If we start by querying all of $X(f)$, we might have to query f too, while the optimal solution might query f only. In this case the expected competitive ratio is bounded by $(k+1)p+(1-p)$. As

$$\max\left\{\frac{kp+(k+1)(1-p)}{k}, (k+1)p+(1-p)\right\}$$

is minimized for $p=\frac{1}{k^2+1}$, we achieve competitive ratio at most $1+\frac{k}{k^2+1}$, if we can guarantee that we start by querying $X(f)$ with probability $\frac{1}{k^2+1}$ or by querying f with probability $1-\frac{1}{k^2+1}$. For general G, we treat each cycle separately. A description of the algorithm RANDOM$_C$ is depicted in Algorithm 5.
Let C_i be the cycle closed in iteration i. For an instance which consists only of C_i, let Q_i^* denote an optimal query set and $OPT_i := |Q_i^*|$. As cycles in G do not share any edges, the disjoint union Q^* of the Q_i^* is an optimal solution for G and thus $OPT = \sum_{i=1}^{m-n+1} OPT_i$. Moreover, this structure guarantees that $X(f_i)$ is independent of the choice of queries in previous iterations as well as their outcome. As
$\mathbb{P}[\text{RANDOM}_C \text{ starts by querying all of } X(f_i)] = \mathbb{P}[b \leq \frac{1}{k^2+1}] = \frac{1}{k^2+1}$, we thus expect at most $(1+\frac{k}{k^2+1})OPT_i \leq 1.5 \cdot OPT_i$ queries in iteration i. Let c denote the number of queries made in the preprocessing.
Thus,

$$\begin{aligned}\frac{\mathbb{E}[ALG]}{OPT} = \frac{c+\mathbb{E}[\sum_{i=1}^{m-n+1} ALG_i]}{OPT} = \frac{c+\sum_{i=1}^{m-n+1}\mathbb{E}[ALG_i]}{OPT} &\leq \frac{c+\sum_{i=1}^{m-n+1} 1.5\cdot OPT_i}{c+\sum_{i=1}^{m-n+1} OPT_i} \\ &\leq 1.5.\end{aligned}$$

□

Remark 3.2. Without preprocessing, the algorithm RANDOM$_C$ runs in $O(mn)$ time and the preprocessing can be done in $O(m^2\log(n))$ time. The analysis is the same as for BALANCE in Remark 2.4, except for the computation of edge potentials which is not needed here.

Example 3.2. We will now demonstrate by an example how the algorithm RANDOM$_C$ works. Consider the graph G with uncertainty sets as in Figure 3.3 below.

Preprocessing: Note that this graph is already preprocessed, in the sense that $T_L = T_U = E \setminus \{\{1,2\},\{5,6\},\{7,9\}\}$. Assume that drawing b uniformly at random

input : An instance of MST-U with cactus graph $G = (V, E)$, uncertainty sets A_e, $e \in E$ and uniform query costs
output: A feasible query set Q

Draw b uniformly at random from $[0, 1]$;
Preprocess the instance such that $T_L = T_U$ and set $\Gamma := T_L$;
Index the edges $f_1, ..., f_{m-n+1}$ in $R := E \setminus T_L$ arbitrarily ;
Initialize $Q = \emptyset$;
for $i \leftarrow 1$ **to** $m - n + 1$ **do**
    Add f_i to Γ and let C_i be the unique cycle closed;
    Let $X(f_i)$ be the set of edges $g \in T_L \cap C_i$ with $U_g > L_{f_i}$;
    Let $k := |X(f_i)|$;
    if $X(f_i) = \emptyset$ **then**
        delete f_i from Γ
    else
        if $b \leq \frac{1}{k^2+1}$ **then**
            add all edges in $X(f_i)$ to Q and query them.
        else
            Add f_i to Q and query f_i.
        while *no edge in the cycle* C_i *is always maximal* **do**
            Query the unqueried edge $e \in C_i \setminus Q$ with maximum U_e and add it to Q.
        Delete an always maximal edge from Γ

Algorithm 5: The algorithm RANDOM$_C$ for MST-U with uniform query costs in cactus graphs

from $[0, 1]$ yields $b = 0.4$. Pick the ordering $f_1 = \{1, 2\}$, $f_2 = \{5, 6\}$ and $f_3 = \{7, 9\}$.

i=1: The edge f_1 closes the cycle $C_1 = (1, 2, 3, 9)$, where it has precisely one neighbor ($X(f_1) = \{\{2, 3\}\}$.) Thus $k := 1$ and we have $\frac{1}{k^2+1} = 0.5 > 0.4 = b$, which is why we query $\{2, 3\}$. As $\bar{w}_{\{2,3\}} = 6 < L_{f_1}$, we can exclude f_1 from Γ.

i=2: Next we add f_2 to Γ, where it closes the cycle $C_2 = (4, 5, 6, 9)$. Its neighbor set is $X(f_2) = \{\{4, 9\}, \{4, 5\}, \{6, 9\}\}$ and thus $k := 3$. As $\frac{1}{k^2+1} = \frac{1}{10} = 0.1 < 0.4$, RANDOM$_C$ decides to query f_2. Now C_2 does still not have an always maximal edge, thus we go on by querying $\{6, 9\}$ and then $\{4, 5\}$. The latter turns out to have maximum weight among edges in C_2 and is excluded from Γ.

i=3: Finally we consider $C_3 = (7, 8, 9)$, which arises from adding f_3 to Γ. Here the neighbor set is $X(f_3) = \{\{7, 8\}, \{8, 9\}\}$ and hence $k := 2$. Again we decide to query

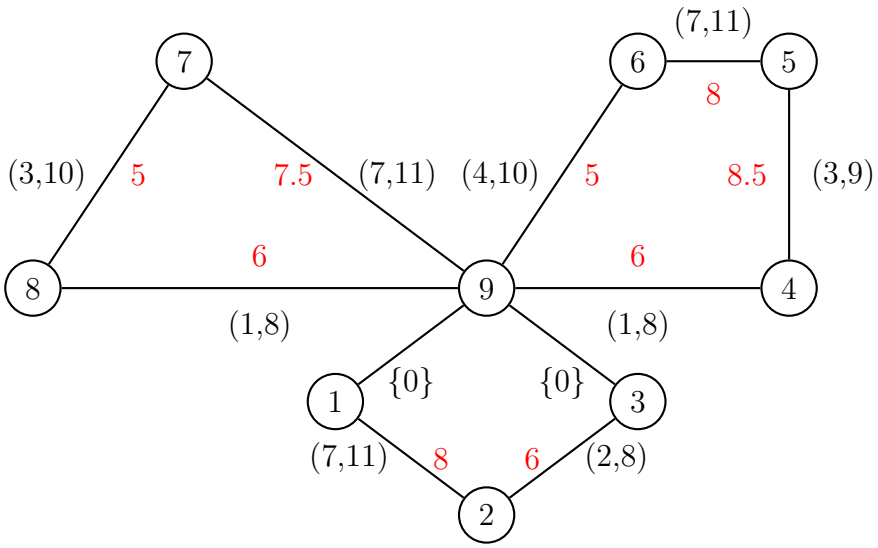


Figure 3.3: An instance of MST-U with uniform query costs in a cactus.The realisation of edge weights is depicted in red.

f_3, because $\frac{1}{k^2+1} = \frac{1}{5} = 0.2 < 0.4$. Unfortunately $\bar{w}_{f_3}$ lies inside the uncertainty sets of both its neighbors, so we have to query $\{7, 8\}$ and $\{8, 9\}$ too in order to see that f_3 has maximum weight among edges in C_3.

Hence for this instance, RANDOM$_C$ outputs the query set $\{\{2, 3\}, \{5, 6\}, \{6, 9\}, \{4, 5\}, \{7, 9\}, \{7, 8\}, \{8, 9\}\}$ if $b = 0.4$. Note that an optimal solution for the cycle C_1 only queries one edge, an optimal solution for C_2 queries $\{4, 5\}, \{5, 6\}$ and $\{6, 9\}$, while an optimal solution for C_3 queries edges of the neighbor set only, i.e. $\{7, 8\}$ and $\{8, 9\}$. Thus for C_1 and C_2 we have found an optimal solution and $OPT_1 = 1$, $OPT_2 = 3$, $OPT_3 = 2$. This yields a competitive ratio of $\frac{7}{6}$ for this instance if $b = 0.4$.

3.3 Non-uniform query costs

3.3.1 Adaption of RANDOM to the non-uniform case

The algorithm RANDOM can be adapted to work for non-uniform query costs as well. In this case we distribute $\frac{q_{f_i}}{\sqrt{2}}$ potential among the neighbors of f_i. This means that in order to determine $t(f_i)$ we have to

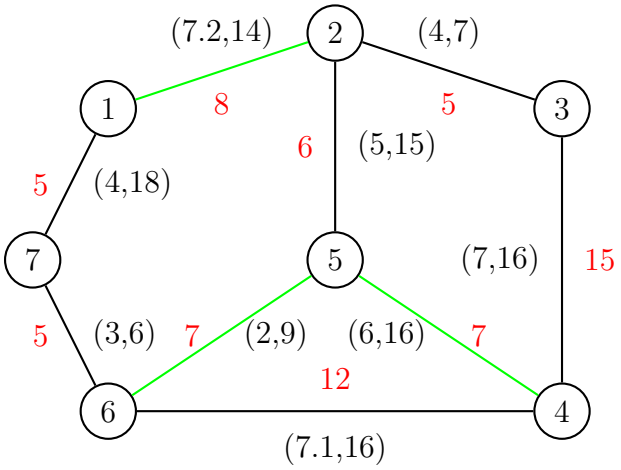
$$\textbf{maximize } t(f_i) \leq 1 \textbf{ s.t. } \sum_{e \in X_{f_i}} q_e \cdot \max\{t(f_i) - y_e, 0\} \leq \frac{q_{f_i}}{\sqrt{2}}. \tag{3.1}$$

The results on the competitiveness of RANDOM extend to this adaption, as shown by [15]:

Theorem 3.4. ([15], page 1230) *For the non-uniform query cost setting the algorithm RANDOM adapted according to (3.1) achieves competitive ratio* $1 + \frac{1}{\sqrt{2}}$.

Remark 3.3. The algorithm RANDOM for uniform as well as for non-uniform query costs runs in polynomial time. The analysis is identical to the one of BALANCE (see Remark 2.4.)

Example 3.3. We will now demonstrate by an example how the algorithm RANDOM works in the case of non-uniform query costs. Consider the graph G with uncertainty sets as in the figure below. We use the same edge names $(e_1, ..., e_9)$ and notation for the uncertainty sets as in Example 2.3, as well as $q_j := q_{e_j}$, $j = 1, ..., 9$ for the query costs.



Preprocessing: As in Example 2.4 the preprocessing results in querying e_4 and yields $T_L(:= T_U) := \{e_1, e_2, e_3, e_4, e_5, e_6\}$. The edges in $R := E \setminus T_L = \{e_7, e_8, e_9\}$ are again indexed via $f_1 := e_7$, $f_2 := e_8$ and $f_3 := e_9$. Assume that drawing b uniformly at random from $[0, 1]$ yields $b = 0.4$ like in Example 3.1.

Figure 3.4: An instance of MST-U with non-uniform query costs. The realisation of edge weights is depicted in red. The query costs are equal to one except for $q_1 = q_6 = 2$ and $q_7 = 1.8$ (in green.)

Now we set $\Gamma := T_L$, $Q := \emptyset$ and $y_j := y_{e_j} = 0$ for all $j \in \{1, ..., 6\}$.

i=1: Adding e_7 to Γ yields $C_1 = (2, 3, 4, 5)$ and $X(e_7) = \{e_5, e_6\}$. Then $t(e_7) = \frac{1}{3\sqrt{2}}$ such that we have $q_5(t(e_7) - y_5) + q_6(t(e_7) - y_6) = 1 \cdot (\frac{1}{3\sqrt{2}} - 0) + 2 \cdot (\frac{1}{3\sqrt{2}} - 0) = \frac{1}{\sqrt{2}} = \frac{q_7}{\sqrt{2}}$. We thus set

$$y_5 := \frac{1}{3\sqrt{2}}, y_6 := \frac{1}{3\sqrt{2}}.$$

As $t(e_7) = \frac{1}{3\sqrt{2}} < 0.4 = b$, we query $f_1 = e_7$, i.e. $Q := \{e_7\}$. We still cannot identify a maximum weight edge in C_1. Thus, we query the edge e_6 too and add it to Q, i.e.

$Q := \{e_6, e_7\}$. The edge e_7 as a maximum weight edge in C_1 is removed from Γ, i.e $\Gamma := T_L$.

i=2: The edge e_8 closes the cycle $C_2 = (4, 5, 6)$ in $\Gamma := T_L \cup \{e_8\}$ and we have $X(e_8) = \{e_1\}$. RANDOM sets $t(e_8) := \frac{1}{2\sqrt{2}}$ and we have $q_1(t(e_8) - y_1) = 2 \cdot (\frac{1}{2\sqrt{2}} - 0) = \frac{1}{\sqrt{2}} = \frac{q_8}{\sqrt{2}}$. The potential of y_1 is increased:

$$y_1 := \frac{1}{2\sqrt{2}}.$$

As $t(e_8) = \frac{1}{2\sqrt{2}} \approx 0.35 < 0.4$, we set $Q := \{e_6, e_7, e_8\}$ and query e_8, revealing $\bar{w}_8 = 12$, which is sufficient to see that e_8 has maximum weight among the edges in C_2. Set $\Gamma := T_L$.

i=3: The edge e_9 closes the cycle $C_3 = (1, 2, 5, 6, 7)$ in $\Gamma := T_L \cup \{e_9\}$. The uncertainty sets of e_1 and e_5 intersect A_9, hence $X(e_9) := \{e_1, e_5\}$. Now in this iteration $t(e_9) = \frac{47}{45\sqrt{2}} \approx 0.74$ and satisfies $q_1(t(e_9) - y_1) + q_5(t(e_9) - y_5) =$
$2 \cdot (\frac{47}{45\sqrt{2}} - \frac{1}{2\sqrt{2}}) + 1 \cdot (\frac{47}{45\sqrt{2}} - \frac{1}{3\sqrt{2}}) = \frac{9}{5\sqrt{2}} = \frac{q_9}{\sqrt{2}}$. Hence, as $t(e_9) > 0.4$, we query e_1 and e_5 which is sufficient to identify e_9 as a maximum weight edge in C_3. Set $Q := \{e_1, e_5, e_6, e_7, e_8\}$, set $\Gamma := T_L$ and output Q.

This means that for $b = 0.4$, the cost of this solution for the preprocessed instance is $2+1+2+1+1 = 7$, while an optimal solution queries e_6, e_5 and e_1, which yields a query cost of 5. This yields a competitive ratio of $\frac{7}{5} = 1.4$ for the preprocessed instance and $\frac{8}{6} \approx 1.33$ for the original instance.

3.3.2 An optimal randomized algorithm for cactus graphs with non-uniform query costs

The algorithm RANDOM$_C$ can easily be adapted to work for non-uniform query costs as well. Let C_i denote the cycle closed by f_i during the i'th iteration and let $X(f_i)$ be the neighbor set of f_i. We denote by q_i the cost of querying all edges in $X(f_i)$, i.e. $q_i := \sum_{e \in X(f_i)} q_e$. With probability $p_i = \frac{q_{f_i}^2}{q_{f_i}^2 + q_i^2}$ we start by querying all edges in $X(f_i)$ and with probability $1 - p_i$ we start by querying f_i. Afterwards we proceed as usually, i.e. as in the framework of BALANCE. A precise description of RANDOM$_C$ for general query costs is given by Algorithm 6.

Proposition 3.2. *The algorithm RANDOM$_C$ for MST-U instances in cactus graphs with general query costs achieves competitive ratio at most 1.5, which is optimal.*

Proof. Let Q_i^* be an optimal query set for the cycle C_i. We set $OPT_i := |Q_i^*|$, $Q_i := Q \cap C_i$ and denote by ALG_i the sum of query costs of edges in Q_i. We aim to show

input : An instance of MST-U with cactus graph $G = (V, E)$, uncertainty sets A_e and query costs q_e, $e \in E$
output: A feasible query set Q

Draw b uniformly at random from $[0, 1]$;
Preprocess the instance such that $T_L = T_U$ and set $\Gamma := T_L$;
Index the edges $f_1, ..., f_{m-n+1}$ in $R := E \setminus T_L$ arbitrarily ;
Initialize $Q = \emptyset$;
for $i \leftarrow 1$ **to** $m - n + 1$ **do**
    Add f_i to Γ and let C_i be the unique cycle closed;
    Let $X(f_i)$ be the set of edges $g \in T_L \cap C_i$ with $U_g > L_{f_i}$;
    Let $q_i := \sum_{e \in X(f_i)} q_e$;
    if $X(f_i) = \emptyset$ **then**
        delete f_i from Γ
    else
        if $b \leq \frac{q_{f_i}^2}{q_{f_i}^2 + q_i^2}$ **then**
            add all edges in $X(f_i)$ to Q and query them.
        else
            Add f_i to Q and query f_i.
        while *no edge in the cycle* C_i *is always maximal* **do**
            Query the unqueried edge $e \in C_i \setminus Q$ with maximum U_e and add it to Q.
        Delete an always maximal edge from Γ

Algorithm 6: The algorithm RANDOM$_C$ for MST-U with general query costs in cactus graphs

that $\frac{\mathbb{E}[ALG_i]}{OPT_i} \leq 1.5$ for all $i = m - n + 1$. Analogously to the proof of Proposition 3.1 we obtain that the competitive ratio is bounded by

$$\frac{p_i q_i + (1 - p_i)(q_i + q_{f_i})}{q_i} = 1 + (1 - p_i)\frac{q_{f_i}}{q_i},$$

if $f_i \notin Q_i^*$ and by

$$\frac{(1 - p_i)q_{f_i} + p_i(q_i + q_{f_i})}{q_{f_i}} = 1 + p_i\frac{q_i}{q_{f_i}},$$

if $f_i \in Q_i^*$. For $p_i = \frac{q_{f_i}^2}{q_{f_i}^2 + q_i^2}$ both these ratios yield the value

$$1 + \frac{q_i q_{f_i}}{q_{f_i}^2 + q_i^2}.$$

Note that

$$\begin{aligned}
&\frac{q_i \cdot q_{f_i}}{q_{f_i}^2 + q_i^2} \leq \frac{1}{2} \iff \\
&2q_i q_{f_i} \leq q_{f_i}^2 + q_i^2 \iff \\
&0 \leq (q_{f_i} - q_i)^2
\end{aligned}$$

and is thus true.
As $\frac{ALG_i}{OPT_i} \leq 1.5$ for all i, we obtain that $\frac{\mathbb{E}[ALG]}{OPT} \leq 1.5$ by the same arguments as used in the proof of Proposition 3.1. □

4 Connection of MST-U to Minimum Bipartite Vertex Cover

In all examples so far we were able to compute the competitive ratio of a solution because with the knowledge of the underlying realization it was "easy to see" which queries an optimal solution would make. This leads to the question whether there is an algorithmic way of computing the optimal solution (i.e. an optimal query set) given the knowledge of the real edge weights. This problem is called Minimum Spanning Tree Verification under Uncertainty and was studied by Erlebach and Hoffmann in [4]. They established a connection between the verification problem of MST-U and Minimum Bipartite Vertex Cover which was later modified by [15] to construct an instance of Online Bipartite Vertex Cover (see Definition 4.2) from an instance of MST-U.

4.1 Minimum Spanning Tree Verification under Uncertainty

Definition 4.1. Consider a graph $G = (V, E)$ such that for each $e \in E$ we are given an uncertainty set A_e, where A_e is an open interval or trivial, a query cost q_e, as well as the edge weight $w_e \in A_e$. The Minimum Spanning Tree Verification Problem under Uncertainty (MST-U-VER) consists in finding a minimum cost query set $Q \subset E$ such that if updating the edges in Q verifies the edge weights w_e, $e \in Q$, then the edge set of an MST can be calculated.

Erlebach and Hoffmann [4] present an algorithm based on U-RED which computes an optimal query set in polynomial time and makes use of a connection between MST-U and Bipartite Vertex Cover. The following approach is similar to the one in [4] but we adapted it to the framework of BALANCE instead of U-RED.

The algorithm VERIFICATION works in three phases. In Phase 1 it identifies a set A of edges that have to be in any feasible query set, as well as a set

$$P := \{(d_1, B_1), ..., (d_k, B_k)\} \subset R \times \mathcal{P}(T_L),$$

where $k \in \mathbb{N}$, $R := E \setminus T_L$ and P fulfills the following two properties:

1. The set $Q' = \{A \cup \{d_i | i \in I\} \cup \bigcup_{j \in J} B_j |\ I,\ J$ form a partition of $\{1, ..., k\}\}$ is a set of feasible query sets,

2. if Q is a feasible query set, it contains an element of Q' as a subset.

Phase 2 is a sorting phase to "tidy up" the set P constructed during Phase 1. During Phase 3, VERIFICATION identifies an element of Q' that minimizes the query cost by establishing a connection to the (weighted) Bipartite Vertex Cover Problem. In the following we describe the three phases in more detail. A formal description of VERIFICATION can be found in Algorithm 7.

Phase 1 : Let $f_i \in R$ be the edge added to T_{i-1} during the execution of BALANCE and let C_i be the unique cycle closed. Then either the edge f_i is always maximal or by Lemma 2.5 and Remark 2.3 one of five cases might occur:

1. Querying f_i is sufficient to find a maximum weight edge in C_i while querying $X(f_i)$ is not.
   In that case f_i which has maximum weight in C_i has to be in any feasible query set, thus VERIFICATION adds f_i to A and deletes it from the current tree.

2. Querying $X(f_i)$ is sufficient to find a maximum weight edge in C_i while querying only f_i is not. However, $\exists S \subsetneq X(f_i)$ such that querying $\{f_i\} \cup S$ is sufficient too.
   In this case it is non-trivial to decide whether $\{f_i\} \cup S$ or $X(f_i)$ should be queried because it not only depends on the query cost but also on future cycles which might contain edges in $X(f_i)$. However, all edges in S lie in every feasible query set with certainty. So VERIFICATION adds S to A and the pair $(f_i, X(f_i) - S)$ to P and removes f_i from the current tree.

3. Querying $X(f_i)$ is sufficient to find a maximum weight edge in C_i while querying only f_i is not and $\nexists S \subsetneq X(f_i)$ such that querying $\{f_i\} \cup S$ is sufficient too.
   Hence $X(f_i)$ is contained in any feasible query set, thus VERIFICATION adds all edges in $X(f_i)$ to A and deletes f_i from the current tree.

4. Querying either f_i only or all of $X(f_i)$ is sufficient to identify a maximum weight edge in C_i. Hence it is again unsure whether OPT queries f_i or $X(f_i)$. Thus VERIFICATION adds the pair $(f_i, X(f_i))$ to P and deletes f_i from $T_{i-1} + f_i$.

5. Neither querying f_i nor querying $X(f_i)$ is sufficient to determine which edge in C_i has maximum weight. This means that $\{f_i\} \cup S$ must lie in every feasible query set, where $S \subset X(f_i)$ contains edges of C_i in order of decreasing upper limit up to the point where a maximum weight edge e can be identified. Thus we add all edges in $S \cup \{f_i\}$ to A and delete the edge e from the current tree.

Phase 2: In Phase 2 we take care of the possibility that for a pair (d, B) in P some edges in B also lie in A. After the execution of VERIFICATION Phase 1, the set P is updated in the following way: For every $(d, B) \in P$, all elements of $B \cap A$ are removed from B. In the case where B becomes empty, the pair (d, B) is removed from P entirely. Properties 1 and 2 remain true.

Phase 3: During this phase, the optimal solution is computed. Consider a weighted bipartite graph $G' = (V_1 \cup V_2, E')$, where $V_1 = \{d_1, ..., d_k\}$, $V_2 = \bigcup_{i=1,...,k} B_i$ and $d_i b$ is an edge in E' if $b \in B_i$. The weight of a vertex e in G' is given by the query cost q_e of the corresponding edge $e \in E$. We compute a vertex cover L in G' of minimum weight. The edges corresponding to vertices in L together with the necessary edges in A yield our query set Q.

Theorem 4.1. *The algorithm VERIFICATION solves MST-U-VER with general query costs correctly and runs in polynomial time.*

Proof. Note that with the knowledge of the edge weights, VERIFICATION is able to identify which of the five cases described above occurs by determining the maximum weight edge in C_i and comparing w_e with L_{f_i} and w_{f_i} with U_e for $e \in X(f_i)$.
First we argue that the required Properties 1 and 2 of P are indeed fulfilled. Property 1 holds due to the following observations: If we add a pair (d, B) to P in Case 4, it is sufficient to query either d or B to determine the maximum weight edge in the cycle of the current iteration of BALANCE. If we add (d, B) to P in Case 2, a maximum weight edge in the current cycle of BALANCE can be determined by querying either d or $B \cup S$, where $S \subset A$.
Property 2 of P is fulfilled by Lemma 2.5 and Remark 2.3. Note that Lemma 2.5 and Remark 2.3 also guarantee that edges in A lie in any feasible query set.

Now we argue that the output Q is indeed an optimal query set.
By Property 2 of P, it is sufficient to find a cheapest query set within Q'. Elements of A are in every feasible solution. Thus, we need to find an element of

$$P' = \{\{d_i | i \in I\} \cup \bigcup_{j \in J} B_j |\ I, J \text{ form a partition of } \{1, ..., k\}\}$$

with smallest query cost, as each element of P' is disjoint to A.

input : A graph $G = (V, E)$ and for each $e \in E$ an uncertainty set A_e, the query cost q_e and the edge weight $w_e \in A_e$
output: An optimal query set Q

Preprocess the instance such that $T_L = T_U$ and set $\Gamma := T_L$;
Index the edges in $R := E \setminus T_L$ such that $f_1 \leq ... \leq f_{m-n+1}$;
Initialize $A = \emptyset$, initialize $P = \emptyset$;
// Phase 1
for $i \leftarrow 1$ **to** $m - n + 1$ **do**
  Add f_i to Γ and let C_i be the unique cycle closed;
  Let $X(f_i)$ be the set of edges $g \in T_L \cap C_i$ with $U_g > L_{f_i}$;
  if $X(f_i)$ *is empty* **then**
    Remove f_i from Γ.
  else
    if $w_{f_i} \geq U_e$ *for all* $e \in X(f_i)$ *and there exists an edge* $e \in X(f_i)$ *such that* $w_e > L_{f_i}$ **then**
      $A := A \cup \{f_i\}$ and $\Gamma := \Gamma - f_i$.
    if $w_e \leq L_{f_i}$ *for all* $e \in X(f_i)$ *and there exists an edge* $e \in X(f_i)$ *such that* $U_e > w_{f_i}$ **then**
      Let $S = \{e \in X(f_i) | U_e > w_{f_i}\}$;
      if $S \neq X(f_i)$ **then**
        $A := A \cup S$, $P := P \cup \{(f_i, X(f_i) \setminus S)\}$ and $\Gamma := \Gamma - f_i$
      else
        $A := A \cup X(f_i)$ and $\Gamma := \Gamma - f_i$
    if $w_{f_i} \geq U_e$ *and* $w_e \leq L_{f_i}$ *for all* $e \in X(f_i)$ **then**
      $P := P \cup \{(f_i, X(f_i))\}$ and $\Gamma := \Gamma - f_i$
    else
      Let $S \subset X(f_i)$ contain edges of C_i in order of decreasing upper limit up to the point where querying $\{f_i\} \cup S$ allows to identify a maximum weight edge e in C_i;
      Set $A := A \cup \{f_i\} \cup S$ and $\Gamma := \Gamma - e$.
// Phase 2
for $(d, B) \in P$ **do**
  $B := B - (B \cap A)$;
  if $B = \emptyset$ **then**
    $P := P - (d, B)$
// Phase 3
Construct the graph G' from P, let L be a minimum vertex cover in G';
Output $Q := A \cup L$

Algorithm 7: The algorithm VERIFICATION for MST-U-VER with general query costs

(For a pair $(d, B) \in P$ we have that B is disjoint to A due to the execution of Phase 2 and $d \notin A$ because an edge f_i in R will never become part of a neighbor set, nor reappear as $f_j \in R$ during a later iteration j.)
Our problem of finding an element of P' with minimum overall query cost translates to a Minimum Vertex Cover Problem in G' in the following sense: An element of P' contains either d_i or all vertices inside B_i for each $i \in \{1, ..., k\}$ and is thus a vertex cover of G'. Now let L be a vertex cover of G'. If there is an i such that $d_i \notin L$, then it must contain all neighbors of d_i in G', which means that $B_i \subset L$. Hence every vertex cover of G' contains an element of P' as a subset. Thus a minimum vertex cover of G' corresponds to an element of P' with minimum query cost.

Finally note that VERIFICATION runs in polynomial time:
Phase 1: The first phase needs at most $O(m^2 \log(n))$ time with the same arguments as used for BALANCE in Remark 2.4 (bottleneck preprocessing!)
Phase 2: P has $O(m)$ elements and each B_i has $O(n)$ elements, hence "tidying up" P can be done in at most $O(mn)$ time.
Phase 3: The instance of Minimum Bipartite Vertex Cover has $O(m)$ vertices and $O(mn)$ edges. Hence it can be solved in $O(\sqrt{m^5 n})$ time by reducing to the maximum flow problem and using the Push-Relabel algorithm. □

We demonstrate the algorithm VERIFICATION by the following two examples:

Example 4.1. First we consider our usual instance of Example 2.3 (alongside the notation of Example 2.3.) Assume all query costs equal 1. We initialize $A := \emptyset$ and $P := \emptyset$.

Phase 1:

i=1: For $f_1 = e_7$ and $X(e_7) = \{e_5, e_6\}$, we have $\bar{w}_e \leq L_{e_7}$ for all $e \in X(e_7)$ and $S := \{e \in X(e_7) | U_e > \bar{w}_7\} = \{e_6\} \neq X(e_7)$. Thus we set $A := A \cup S = \{e_6\}$ and $P := \{(e_7, \{e_5\})\}$.

i=2: Similarly, for $f_2 = e_8$ and $X(e_8) = \{e_1, e_6\}$, we have $\bar{w}_e \leq L_{e_8}$ for all $e \in X(e_8)$ and $S := \{e \in X(e_8) | U_e > \bar{w}_8\} = \{e_6\} \neq X(e_8)$. Thus A remains unchanged and $P := P \cup \{(e_8, \{e_1\})\} = \{(e_7, \{e_5\}), (e_8, \{e_1\})\}$.

i=3: Finally for $f_3 = e_9$ and $X(e_9) = \{e_1, e_5\}$, we have $\bar{w}_e \leq L_{e_9}$ for all $e \in X(e_9)$ but $S := \{e \in X(e_9) | U_e > \bar{w}_9\} = X(e_9)$. Thus $A := A \cup X(e_9) = \{e_1, e_5, e_6\}$ and P remains unchanged.

Phase 2:
As both edges, e_1 and e_5 lie in A, $(e_7, \{e_5\}), (e_8, \{e_1\})$ are both removed from P.

Phase 3:
As P is empty, we obtain that A is an optimal query set without solving any Bipartite Vertex Cover instance.

A little more meaningful is the following example:

Example 4.2. Consider an instance of MST-U as depicted in Figure 4.1 below. Note that the instance is preprocessed in the sense that $T_L = T_U = E \backslash \{\{1,2\},\{2,3\},\{5,6\}\}$. W.l.o.g. we set $f_1 := \{2,3\}, f_2 := \{1,2\}$ and $f_3 := \{5,6\}$. We initialize $A := \emptyset$ and $P := \emptyset$.

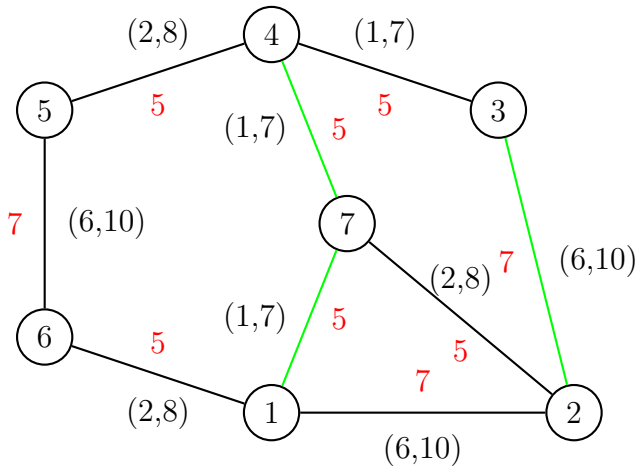


Figure 4.1: An instance of MST-U with non-uniform query costs, where red numbers denote the realization of edge weights. The query costs are one for all edges, except $q_{\{2,3\}} = q_{\{1,7\}} = q_{\{4,7\}} = 0.5$. These edges are drawn in green.

Phase 1:

i=1: The edge f_1 closes the cycle $C_1 := (2,3,4,7)$ such that $X(f_1) = \{\{2,7\},\{3,4\},\{4,7\}\}$. Now either querying all of $X(f_1)$ or querying $\{f_1,\{2,7\}\}$ is sufficient to see that f_1 has maximum weight among edges in C_1 (Case 2 in the explanation of the algorithm.) We thus have $S = \{\{2,7\}\} \neq X(f_1)$. We set $A := A \cup S = \{\{2,7\}\}$ and $P := \{(f_1,\{\{3,4\},\{4,7\}\})\}$.

i=2: Similarly, for f_2 and $X(f_2) = \{\{2,7\},\{1,7\}\}$, we could either query all edges in $X(f_2)$ or query f_2 and {2,7}. Hence $S := \{\{2,7\}\}$, A remains unchanged and $P := \{(f_1,\{\{3,4\},\{4,7\}\}),(f_2,\{\{1,7\}\})\}$.

i=3: Finally for f_3 and $X(f_3) = \{\{1,7\},\{4,7\},\{4,5\},\{1,6\}\}$, we can query the entire neighbor set or f_3 together with $\{1,6\}$ and $\{4,5\}$. The latter two must hence

lie in any feasible query set. Thus, we set $A := \{\{2,7\},\{1,6\},\{4,5\}\}$ and $P := \{(f_1,\{\{3,4\},\{4,7\}\}),(f_2,\{\{1,7\}\}),(f_3,\{\{1,7\},\{4,7\}\}\}$.

Phase 2:
As A does not intersect any of the B_i, $i = 1,2,3$, the set P remains unaltered.

Phase 3:
Now we obtain an instance of Minimum Weight Bipartite Vertex Cover as displayed in Figure 4.2. It is easy to see that $\{f_1,\{4,7\},\{1,7\}\}$ yields a minimum weight vertex cover. Hence $\{\{2,7\},\{1,6\},\{4,5\},\{2,3\},\{4,7\},\{1,7\}\}$ is an optimal query set with weight 4.5.

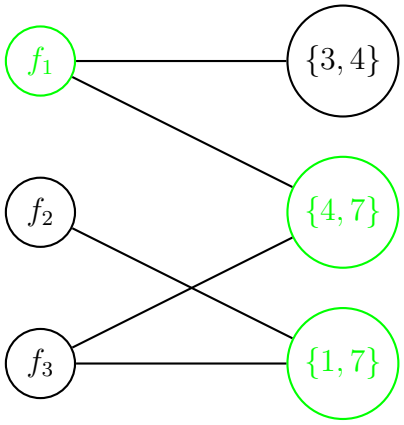


Figure 4.2: VERIFICATION needs to solve an instance of Minimum Weighted Bipartite Vertex Cover. Green vertices correspond to edges with query cost 0.5.

4.2 Online Bipartite Vertex Cover

Megow et al. [15] use the connection between Minimum Bipartite Vertex Cover and MST-U-VER to build an instance of the Online Bipartite Vertex Cover.

Definition 4.2. An instance of Online Bipartite Vertex Cover consists of a bipartite graph $G = (A \cup B, E)$ with $B = \{b_1,...,b_k\}$ and vertex weights $w_v \geq 0$, $v \in A \cup B$. We aim to find a sequence $(C_i)_{i=1}^k$ such that $C_i \subset V$ for $i = 1,...,k$ and $C_{i-1} \subset C_i$ for $i = 2,...,k$, where C_i is a vertex cover of the graph $G[A \cup \{b_1,...,b_i\}]$ which is induced by the vertices in A and the first i vertices in B.
However, only the vertices of A,the so-called offline vertices, are given a priori, while the vertices in B are initially unknown (and thus also their ordering $b_1,...,b_k$). They appear one at a time alongside their incident edges. In any iteration an algorithm has

to maintain a valid vertex cover of the current graph, while keeping the overall weight of the cover as small as possible. Once a vertex is added to the vertex cover it cannot be removed during a later iteration. (Otherwise we could simply compute the offline solution in the end.)

Now consider the following graph $G' = (A \cup B, E')$ constructed throughout the execution of RANDOM: The edges of the graph G in our instance of MST-U give rise to the vertices of the instance of Online Bipartite Vertex Cover. Let $A := T_L$ be the offline vertices. The edges in $R := E \setminus T_L$ give rise to the set B of online vertices and appear in the order in which they arise during the execution of RANDOM. Once the edge $f_i \in R$ appears, its vertex in G' is connected to all vertices corresponding to edges in $X(f_i)$ (defined as in Definition 2.5) via edges in E'. The weights of the vertices in G' are given by the query costs of the corresponding edges. This is indeed an instance of Online Bipartite Vertex Cover, as it depends on the cycles which are closed during the execution of RANDOM and thus on the realization of edge weights. Hence G' cannot be constructed offline.

Remark 4.1. In fact, [15] make use of the online vertex cover graph G' in their algorithm RANDOM: The water-filling scheme used to compute the increase of edge potentials during the execution of RANDOM is an adaption of an algorithm by Wang and Wong for Online Bipartite Vertex Cover in [16].

5 Alternative versions of MST under uncertainty

5.1 Minimum Spanning Tree with Vertex Uncertainty

The model of vertex uncertainty (V-MST-U) is introduced in [7]. In this setting the vertices correspond to points in the Euclidean plane and the weight of an edge is the distance between its end vertices. The locations of the points are initially uncertain but an algorithm can update a vertex v at query cost q_v to reveal its exact location. As usual, V-MST-U can be defined for general query costs $q_v > 0$. In this master's thesis however (as well as in the existing literature so far), we will only see results for the case where query costs are uniform.

5.1.1 Deterministic algorithm

Erlebach et al. [7] argue that U-RED can be adapted to the setting of vertex uncertainty.

Definition 5.1. Given an instance I of V-MST-U with graph $G = (V, E)$ and uncertainty sets A_v, $v \in V$ the associated edge instance $\tilde{I}$ is an instance of MST-U with graph G and uncertainty sets $A_{\{u,v\}} = \{d(u', v') | u' \in A_u, v' \in A_v\}$.

Remark 5.1. Observe the following two relations between V-MST-U and MST-U. Let I be an instance of V-MST-U and $\tilde{I}$ the associated edge instance.

- If $e = uv$ is an edge of G we will not gain any information about e's weight if we query neither u nor v. Hence, if W is a witness set of the associated edge instance then $W' := \{u|\ u \text{ is an end-vertex of an edge in } W\}$ is a witness set of I.

- If all uncertainty sets of I are either trivial or open then every uncertainty set of $\tilde{I}$ is also either trivial or open. Thus, Theorem 2.1 yields that whenever every witness set in a witness algorithm A for MST-U has size at most k, we can turn it into a $2k$-competitive algorithm A' for V-MST-U by simulating an edge query through querying its two end-vertices.

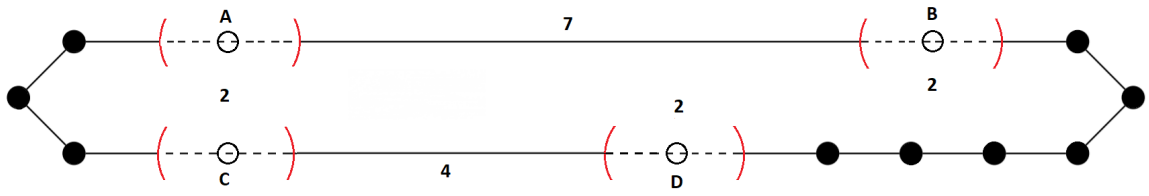


Figure 5.1: Lower bound for V-MST-U (adapted from [7], page 287)

Hence U-RED yields a 4-competitive algorithm for V-MST-U with uniform query costs, under the restriction to trivial or open uncertainty sets. If we update a vertex v which is incident to multiple edges, we obtain information about every incident edge. This leads to the question if we might achieve a better competitive ratio if we choose an approach that is not related to the associated edge instance. The answer to this question is in fact no, as shown by [7]:

Theorem 5.1. ([7], page 287) *No deterministic algorithm for V-MST-U under the restriction to trivial or open uncertainty sets can achieve a competitive ratio less than 4. This remains true even under the assumption of uniform query costs.*

Proof. Consider the graph displayed in Figure 5.1, where all black dots represent vertices with trivial uncertainty sets and all query costs equal 1. The vertices A, B, C and D have non-trivial, open uncertainty sets of length 2 and small positive width. (More precisely, we consider open sets A_v in $\mathbb{R}^2$, such that the largest distance between points in the closure of A_v equals 2 and is attained only for the two points in the intersection between the boundary of A_v and the upper horizontal line, if $v = A$ or $v = B$, or the lower horizontal line, if $v = C$ or $v = D$, of the graph embedding depicted in Figure 5.1.) The length of each edge between trivial vertices is 1 and the distance between each non-trivial uncertainty set and its closest incident vertex is 1 as well. Thus all but two edges lie in any MST with certainty and we only need to determine whether $\{A, B\}$ or $\{C, D\}$ should be included in an MST.
We distinguish four cases, depending on which of the non-trivial uncertainty sets is not among the first three to be queried by an algorithm. For each of these cases a malicious adversary confronts the algorithm with a different realization, which forces the algorithm to query the fourth vertex too:

- If A (D) is among the first three vertices to be queried, it is located at distance at most ϵ from the far right end of A_A (A_D) and
- if B (C) is among the first three vertices to be queried, it is located at distance at most ϵ from the far left end of A_B (A_C),

where $\epsilon > 0$ is small.

Case 1: The algorithm queries B, C and D first. Then $d(A, B) \in (7 + \epsilon, 9 + \epsilon)$ and $d(C, D) = 8 - 2\epsilon \in (7 + \epsilon, 9 + \epsilon)$, where d denotes the Euclidean distance. Hence the algorithm needs to query A in order to know if $\{A, B\}$ or $\{C, D\}$ has smaller weight. Now assume that the location of A turns out to be located at distance at most ϵ from the far left end of A_A. Then by updating only A, the optimal solution finds out that $d(A, B) \in (9 - \epsilon, 11 - \epsilon)$ and $d(C, D) \in (4, 8)$. Hence an optimal solution only needs a single query to exclude the edge $\{A, B\}$.

Case 2: The case where the algorithm queries A, C and D first works analogously to Case 1.

Case 3: The algorithm queries A, B and D first. Then $d(C, D) \in (6 - \epsilon, 8 - \epsilon)$ and $d(A, B) = 7 + 2\epsilon \in (6 - \epsilon, 8 - \epsilon)$. Hence, C needs to be queried too. If C is located at distance at most ϵ from the far right end of A_C, then by updating only C instead of A, B and D, the optimal solution sees that $d(A, B) \in (7, 11)$ while $d(C, D) \in (4-\epsilon, 6-\epsilon)$, which means that $\{C, D\}$ is inside the MST.

Case 4: The case where the algorithm queries A, B and C first works again analogously to Case 3.

□

A surprising difference between the edge and the vertex setting was shown by [4] with respect to the verification problem. The verification problem of V-MST-U is defined analogously to Definition 4.1. Remember that for MST-U the verification problem can be solved in polynomial time (Theorem 4.1). Erlebach and Hoffmann [4] however show, that this is not the case for vertex uncertainty:

Theorem 5.2. ([4], page 175) *The verification problem of V-MST-U is NP-hard.*

5.1.2 Randomization

Lower Bound

We will first prove a lower bound for the performance of any randomized algorithm for V-MST-U.

Proposition 5.1. *No randomized algorithm for V-MST-U under the restriction to trivial or open uncertainty sets can achieve a competitive ratio less than 2.5. This remains true even if query costs are restricted to be uniform.*

Proof. Consider again the instance of V-MST-U as in the proof of Theorem 5.1. We denote the instances constructed by the adversary in the proof of Theorem 5.1 with R_A, R_B, R_C and R_D. More precisely for sufficiently small $\epsilon > 0$,

- let R_A be a realization such that A, B and C are located at distance at most ϵ from the far left end of their uncertainty sets A_A, A_B, A_C and D is located at distance at most ϵ from the far right end of A_D.
- Let R_B be such that C is located at distance at most ϵ from the far left end of its uncertainty set A_C and A, B, D are located at distance at most ϵ from the far right end of A_A, A_B resp. A_D.
- Let R_C be such that B is located at distance at most ϵ from the far left end of its uncertainty set A_B and A, C, D are located at distance at most ϵ from the far right end of A_A, A_C resp. A_D.
- Let R_D be such that B, C and D are located at distance at most ϵ from the far left end of their uncertainty sets A_B, A_C, A_D and A is located at distance at most ϵ from the far right end of A_A.

In the proof of Theorem 5.1 we have seen that if G has realization A_v, $v \in \{A, B, C, D\}$, it is sufficient to query v to identify an MST, while querying all edges in $V \setminus \{v\}$ is not enough to exclude a maximum weight edge.
Consider the randomized family of instances $(\mathcal{R}, p)$, where $\mathcal{R} = \{R_A, R_B, R_C, R_D\}$ and $p(R_v) = \mathbb{P}[R = R_v] = 0.25$ for $v \in \{A, B, C, D\}$. Then no deterministic algorithm ALG achieves a better expected competitive ratio $\mathbb{E}_{R\sim_p\mathcal{R}}(\frac{ALG(G,R)}{OPT(G,R)})$, than the algorithm ALG_1 which queries A, B, C, D (or less if an MST can already be identified) in this order independently from the queries' results. This is due to the fact that querying v only reveals whether or not we are facing realization R_v but if not, it is indistinguishable which of the remaining realizations it might be.
Then by a variant of Yao's Principle (see Borodin and El-Yaniv [1], Theorem 8.5), no randomized algorithm has a better performance than

$$\begin{aligned}
&\min_{ALG\in\mathcal{A}} \mathbb{E}_{R\sim_p\mathcal{R}}(\frac{ALG(G,R)}{OPT(G,R)}) = \mathbb{E}_{R\sim_p\mathcal{R}}(\frac{ALG_1(G,R)}{OPT(G,R)}) = \\
&0.25(ALG_1(G,R_A) + ALG_1(G,R_B) + ALG_1(G,R_C) + ALG_1(G,R_D)) = \\
&0.25(1+2+3+4) = 2.5,
\end{aligned}$$

where $\mathcal{A}$ denotes the class of all deterministic algorithms. □

Preprocessing

A first step towards a randomized algorithm for V-MST-U is to adapt the preprocessing of RANDOM to the vertex setting. Let I be an instance of V-MST-U. If we compute T_L and T_U for the associated edge instance $\tilde{I}$, then by Remark 2.2, any edge in $T_L \setminus T_U$ lies in any feasible edge query set for $\tilde{I}$. Hence for each edge $e = \{x, y\} \in T_L \setminus T_U$, the

set $\{x, y\}$ is a witness set of I by the first property in Remark 5.1. However, when simulating an edge query in the preprocessing of $\tilde{I}$ by querying its end vertices, it is important to update one edge at the time before rebuilding T_L and T_U and updating the next edge in the new $T_L \setminus T_U$.

Remark 5.2. Each time that we simulate an edge query in the preprocessing for V-MST-U, we update a witness set of size 2. Hence half of the vertices queried during the preprocessing for V-MST-U lie in any feasible solution.

Note that this preprocessing is not as "stable" as in the edge version: Whenever we update an edge by querying its end vertices we not only get the edge's weight but also alter the uncertainty sets of adjacent edges. This means that an edge which has largest upper and lower limit in a cycle in $\tilde{I}$ is not guaranteed to still have this property after an edge query was made. Thus it is not straightforward to carry over the algorithm RANDOM to the setting of V-MST-U by simulating an edge query by updating its end vertices. Note for instance, that if a maximum weight edge in a cycle C_i cannot be identified after having queried the end vertices of f_i or of all of the neighbors in $X(f_i)$, it is not necessarily true that an edge which has largest upper limit in the current cycle of the associated edge instance also forms a witness set in I. Moreover, for $i > 1$ it is not even sure that f_i has largest upper and lower limit in C_i at the moment the cycle is closed. We can handle the first of these two issues. In order to avoid the second one we consider again a special case similar to Section 3.2.2.

Special Case: Disjoint Cycles

In this section we consider only instances of V-MST-U where no two cycles share a vertex with non-trivial uncertainty set. For instances of this type it is possible to obtain a randomized algorithm whose performance is better than in the deterministic case.

Consider an instance I consisting of a cycle C which has been preprocessed such that there is an edge in the associated edge instance $\tilde{I}$ which has largest upper and lower limit and which we denote by f. We will first establish an analogue of the framework of RANDOM where edges are queried in order of decreasing upper limit, once either f or all of $X(f)$ has been queried. Assume we have queried f (its two end vertices) and still no edge is known to have maximum weight. Let g be an edge with maximum upper limit in C. We distinguish two cases:

- A_g contains the weight of f. Then an optimal vertex query set must contain at least one end vertex of g by Remark 2.3 and Remark 5.1. Note that this is always the case if both end vertices of g have not yet been queried.

- $L_g \geq w_f$. Let h be an edge with largest upper limit in $C \setminus \{g\}$. Then by Lemma 2.2 and Remark 5.1 the end vertices of g and h give a witness set of I. Note that the size of this witness set is at most 3, because $L_g \geq w_f$ implies that one end vertex of g has already been queried.

Remark 5.3. Thus, once we have queried f and still no edge is known to have maximum weight, we can solve the instance by additionally querying at most three times the number of vertices an optimal solution queries. If f has maximum weight then we will always end up in the first case and thus solve the instance by additionally querying at most two times the number of vertices an optimal solution queries.

We will now briefly describe how the algorithm V-RANDOM$_C$ works. The algorithm first preprocesses the instance such that $T_L = T_U$ in the associated edge instance. This is done by updating witness sets of size 2 (remember Remark 5.2). Like RANDOM, the algorithm continues by adding edges $f_i \notin T_L$ to the current tree one at a time and each time removes a longest edge in the cycle C_i closed by f_i. Now remember Remark 3.1 for the edge uncertainty problem. A similar observation is used for the vertex uncertainty version too: It is beneficial to start by querying end vertices of f_i unless the optimal solution is small and does not contain end vertices of f_i. Thus V-RANDOM$_C$ distinguishes three scenarios.

Scenario 1: The edges in the neighbor set have at least four different end vertices. This means that we have at least two edges in the neighbor set and if the neighbor set consists of precisely two edges, these two edges do not intersect. In this scenario V-RANDOM$_C$ performs deterministically. It starts by querying the end vertices of f_i. In case we still do not know which edge in the current cycle is the longest, we update either an edge with largest upper limit or two edges g and h such that g has largest upper limit in C_i and h has largest upper limit in $C_i \setminus \{g\}$ until we find a longest edge. During this step it is guaranteed that we update witness sets of size at most 3 (see Remark 5.3). A detailed description of how the algorithm performs in Scenario 1 is displayed in the subroutine V-DET$_C$ in Algorithm 9.

Scenario 2: The neighbor set of f_i contains precisely one edge e. In this scenario, we query the at most four different end vertices of e and f_i in random order up to the point where a longest edge in C_i can be identified. To this end we draw a permutation σ uniformly at random from S_4, if e and f_i are not adjacent, or from S_3 if e and f_i are adjacent and permute the vertices accordingly.

Scenario 3: The neighbor set of f_i contains precisely two edges which intersect in a common vertex w. Now the algorithm wants to make sure that priority is given to those vertices which possibly form an optimal query set Q_i^* of size one in C_i. Remember that a feasible solution which queries neither end vertex of f_i must query an end vertex of each of the neighbor edges (by Lemma 2.5 and Remark 5.1). Hence the

only candidate for a size one query set apart from the end vertices of f_i is w. Thus V-RANDOM$_C$ queries w and the two end vertices of f_i in random order until a longest edge in C_i can be identified or all three vertices have been queried. If still know edge in C_i is known to have maximum length, we query the two remaining end vertices of edges in $X(f_i)$.

A detailed description of V-RANDOM$_C$ is displayed in Algorithm 8.

In the following we will introduce notation used in the description of V-RANDOM$_C$ and in the proof of the algorithm's performance.
Let $N_{f_i} := C_i(X(f_i))$ denote the subgraph of C_i which is induced by $X(f_i)$. We denote by a the number of vertices in N_{f_i}, i.e. $a := |\cup_{\{x,y\}\in X(f)} \{x, y\}|$ and define $b := \sum_K \left\lfloor \frac{|K|}{2} \right\rfloor$, where K sums over the connected components of N_{f_i}. An example is depicted in Figure 5.2.

Remark 5.4. Note that $b \geq \frac{a}{3}$ and that any solution which queries no end vertex of f_i must query at least b vertices.

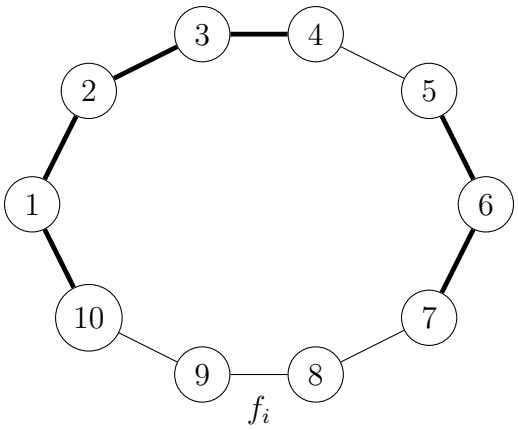


Figure 5.2: C_i with edge f_i and the edges in $X(f_i)$ (bold). Here $a = 8$ and $b = 3$. An optimal solution which avoids f_i must query at least three vertices to cover all edges in $X(f_i)$.

Remark 5.5. Scenario 1 is described by $a \geq 4$, Scenario 2 is described by $a = 2$ and Scenario 3 is described by $a = 3$. Note that $a = 1$ is impossible and $a = 0$ means that $X(f_i)$ is empty and thus f_i is always maximal in C_i.

Moreover we will use the following notation for the technical implementation of the algorithm. Let $\mathcal{S}_4 =: \{\sigma_1^{(4)}, \sigma_2^{(4)}, ..., \sigma_{24}^{(4)}\}$ be the set of permutations
$\sigma_j^{(4)} : \{1, 2, 3, 4\} \to \{1, 2, 3, 4\}$. We assume that the indices j are chosen such that the permutations are ordered lexicographically, e.g. $\sigma_1^{(4)} = (1, 2, 3, 4)$, $\sigma_2^{(4)} = (1, 2, 4, 3)$, ...

Let $\sigma_j^{(3)}$ with $j = 1, ..., 6$ be defined analogously.
For a vertex label $l : V \to \{1, ..., n\}$ and a subset $A \subset V$ we denote by $r : A \to \{1, ..., |A|\}$ the order preserving bijection, i.e. $r(u) < r(v) \iff l(u) < l(v)$ for all $u, v \in A$.

Now we will prove that V-RANDOM$_C$ achieves competitive ratio 3.

Proposition 5.2. *For instances of V-MST-U with uniform query costs where no two cycles share a vertex with non-trivial uncertainty set, the algorithm V-RANDOM$_C$ achieves competitive ratio at most 3.*

Proof. Let C_i be the cycle closed in iteration i. For an instance which consists only of C_i, let Q_i^* denote an optimal vertex query set and $OPT_i := |Q_i^*|$. As cycles in G do not share non-trivial vertices, the disjoint union Q^* of the Q_i^* is an optimal solution for G and thus $OPT = \sum_{i=1}^{m-n+1} OPT_i$. Moreover, this structure guarantees that $X(f_i)$ is independent of both the choice of queries in previous iterations and the outcome of those queries. We aim to show that we expect at most $3 \cdot OPT_i$ queries in iteration i.

First consider Scenario 1, where $a \geq 4$. We distinguish three cases:

Case 1: $|Q_i^*| \geq 2$ and Q_i^* contains at least one end vertex of f_i,

Case 2: $Q_i^* = \{x\}$, where x is an end vertex of f_i,

Case 2: $Q_i^* \cap f_i = \emptyset$.

For now we assume that with probability p_i we start by querying the end vertices of f_i and with probability $1 - p_i$ we start by querying all vertices in N_{f_i}. We will argue that $p_i = 1$ is the optimal choice, i.e. that the maximum of the three expected competitive ratios in Case 1, Case 2 and Case 3 is minimized for $p_i = 1$.

Case 1: With probability p_i we start by querying f_i. The edge f_i shares at least one vertex with Q_i^* and after having queried f_i, we go on by querying witness sets of size at most three. Hence the competitive ratio is at most 3. If we start by querying vertices in N_{f_i}, we might end up querying all end vertices of edges in $X(f_i)$ and both end vertices of f_i. As $|Q_i^*| \geq 2$ the competitive ratio is bounded by $\frac{a+2}{2}$. Thus the expected competitive ratio $ECR_1(p_i)$ in Case 1 is bounded by

$$ECR_1(p_i) \leq 3p_i + \frac{a+2}{2}(1 - p_i) = 2p_i - \frac{a}{2}p_i + \frac{a}{2} + 1. \tag{5.1}$$

Case 2: With probability p_i we start by querying f_i which yields competitive ratio 2. With probability $1 - p_i$ we start by querying vertices in N_{f_i} and might need to query

input : An instance of V-MST-U with uniform query costs and graph $G = (V, E)$ such that no two cycles share a vertex with non-trivial uncertainty set
output: A feasible query set Q

Draw p uniformly at random from $[0, 1]$;
Let $l : V \to \{1, ..., n\}$ be an arbitrary but fixed labeling of the vertex set;
Let $\mathcal{S}_3 =: \{\sigma_1^{(3)}, \sigma_2^{(3)}, ..., \sigma_6^{(3)}\}$ and $\mathcal{S}_4 =: \{\sigma_1^{(4)}, \sigma_2^{(4)}, ..., \sigma_{24}^{(4)}\}$;
Preprocess the instance such that $T_L = T_U$ and set $\Gamma := T_L$;
Index the edges $f_1, ..., f_{m-n+1}$ in $R := E \setminus T_L$ arbitrarily ;
Initialize $Q = \emptyset$;
for $i \leftarrow 1$ **to** $m - n + 1$ **do**
    Let x_1 and x_2 denote the end vertices of f_i;
    Add f_i to Γ and let C_i be the unique cycle closed;
    Let $X(f_i)$ be the set of edges $g \in T_L \cap C_i$ with $U_g > L_{f_i}$;
    Compute $a := |\cup_{\{x,y\} \in X(f_i)} \{x, y\}|$;
    if $a \leq 1$ **then**
        Delete f_i from Γ
    else if $a = 2$ **then**
        `// Query all end vertices in` $X(f_i) \cup \{f_i\}$ `in random order`
        Let w_1, w_2 denote the end vertices of the edge in $X(f_i)$;
        Let $k := |\{x_1, x_2, w_1, w_2\}| \in \{3, 4\}$;
        Let $r : \{x_1, x_2, w_1, w_2\} \to \{1, ..., k\}$ be the order preserving bijection w.r.t l;
        Let $j \in \{1, ..., k!\}$ be such that $\frac{j-1}{k!} < p \leq \frac{j}{k!}$ if $p > 0$ else $j := 1$;
        Query the vertices in $\{x_1, x_2, w_1, w_2\}$ in order of increasing $\sigma_j^{(k)} \circ r$ until a maximum weight edge in C_i can be identified.
    else if $a = 3$ **then**
        `// Query all candidates for a size 1 query set first and in random order`
        Let w denote the vertex incident to two neighbors;
        Let $r : \{x_1, x_2, w\} \to \{1, 2, 3\}$ be the order preserving bijection w.r.t l;
        Let $j \in \{1, ..., 6\}$ be such that $\frac{j-1}{6} < p \leq \frac{j}{6}$ if $p > 0$ else $j := 1$;
        Query the vertices in $\{x_1, x_2, w\}$ in order of increasing $\sigma_j^{(3)} \circ r$ until a maximum weight edge in C_i can be identified or all three vertices were queried;
        if *No edge in C_i is always maximal* **then**
            Query the remaining end vertices of edges in $X(f_i)$.
    else
        Use Algorithm V-DET$_C$ to find a maximum weight edge in C_i
    Delete an always maximal edge in C_i from Γ

Algorithm 8: The algorithm V-RANDOM$_C$ for V-MST-U with uniform query costs in graphs where no two cycles intersect in non-trivial vertices

input : A cycle C_i of the algorithm V-RANDOM$_C$ with its edge f_i, neighbor set $X(f_i)$, as well as the current query set Q and the uncertainty sets A_v for $v \in V(C_i)$
output: A query set Q

```
Add both end vertices of f_i to Q and query them;
while no edge in the cycle C_i is always maximal do
    Let g be an edge with largest upper limit in X(f_i);
    if w_{f_i} ∈ A_g then
        Query the end vertices of g and add them to Q.
    else
        Let h be an edge with largest upper limit in X(f_i) \ {g};
        Query the end vertices of g and h and add them to Q.
```

Algorithm 9: The subroutine V-DET$_C$

all end vertices of edges in $X(f_i)$ and both end vertices of f_i. Thus the competitive ratio is bounded by $a+2$. Hence the expected competitive ratio $ECR_2(p_i)$ in Case 2 is at most

$$ECR_2(p_i) \leq 2p_i + (a+2)(1-p_i) = -ap_i + a + 2. \tag{5.2}$$

Case 3: With probability p_i we start by querying f_i and go on by querying witness sets of size at most two (see Remark 5.3). Hence with probability p_i the competitive ratio is at most

$$\frac{2 + 2 \cdot OPT_i}{OPT_i} = 2 + \frac{2}{OPT_i} \leq 2 + \frac{2}{b},$$

by Remark 5.4. With probability $1-p_i$ we start by querying vertices in N_{f_i} which yields a competitive ratio of at most $\frac{a}{b} \leq 3$, also by Remark 5.4. Thus the expected competitive ratio $ECR_3(p_i)$ in Case 3 is bounded by

$$ECR_3(p_i) \leq (2 + \frac{2}{b})p_i + 3(1-p_i) = \frac{2}{b}p_i - p_i + 3. \tag{5.3}$$

Now we show that $\max\{ECR_1(p_i), ECR_2(p_i), ECR_3(p_i)\}$ is minimized for $p_i = 1$. As $a \geq 4$ implies that $b \geq 2$, we have that $2 + \frac{2}{b} \leq 3$ and $\frac{a+2}{2} \geq 3$. Hence

$$(2 + \frac{2}{b})p_i + 3(1-p_i) \leq 3p_i + \frac{a+2}{2}(1-p_i),$$

which means that the value in Equation 5.3 is at most the value in Equation 5.1. Thus it is enough to look at the values in 5.1 and 5.2. Note that $a \geq 4$ implies that $2 - \frac{a}{2} \leq 0$. This means that the value in Equation 5.1 is minimized for $p_i = 1$. The same holds for Equation 5.2 because $-a \leq 0$. For $p_i = 1$ the value in Equation 5.1 equals

$$2 - \frac{a}{2} + \frac{a}{2} + 1 = 3$$

and the value in Equation 5.2 equals

$$-a + a + 2 = 2.$$

Thus, if $a \geq 4$ we obtain a competitive ratio of at most 3 if we deterministically query f_i and afterwards proceed as in our analogue of the framework of RANDOM.

Now we consider Scenario 2, i.e. we assume $a = 2$. If an optimal query set Q_i^* for C_i has size at least 2 then the competitive ratio is at most $\frac{a+2}{2} = 2$, even if we query all vertices of edges in the neighbor set and both end vertices of f_i. If $Q_i^* = \{v\}$ is a singleton then it is equally likely that v is the first, the second, the third or the fourth vertex to be queried (if f_i and the edge in the neighbor set do not share an end vertex − otherwise v is queried after one, two or three queries with probability $\frac{1}{3}$ each.) Hence the expected competitive ratio is at most $\frac{1+2+3+4}{4} = 2.5$.
Thus, if $a = 2$ we obtain a competitive ratio of at most 2.5 if we query all of the at most four vertices of edges in $X(f_i) \cup \{f_i\}$ in an order which we pick uniformly at random.

Finally we turn to Scenario 3 and assume $a = 3$. If Q_i^* has size at least 2 then the competitive ratio is again at most $\frac{a+2}{2} = 2.5$. If $Q_i^* = \{v\}$ then v could either be an end vertex of f_i or the vertex shared by the two edges in the neighbor set. Thus it is equally likely that v is queried first, second or third, which means that the competitive ratio is bounded by $\frac{1+2+3}{3} = 2$. Thus, if $a = 3$ we achieve a competitive ratio of at most 2.5 if we query the three vertices which are candidates for a size one query set first (in an order which we pick uniformly at random) and only query the remaining vertices of edges in $X(f_i) \cup \{f_i\}$ if a maximum weight edge in C_i cannot yet be identified. The cases $a = 3$ and $a = 2$ are illustrated in Figure 5.3.

Let c denote the number of queries made in the preprocessing.
Then,

$$\frac{\mathbb{E}[ALG]}{OPT} = \frac{c + \mathbb{E}[\sum_{i=1}^{m-n+1} ALG_i]}{OPT} = \frac{c + \sum_{i=1}^{m-n+1} \mathbb{E}[ALG_i]}{OPT} \leq \frac{c + \sum_{i=1}^{m-n+1} 3 \cdot OPT_i}{0.5c + \sum_{i=1}^{m-n+1} OPT_i} \leq 3.$$

□

Remark 5.6. The algorithm V-RANDOM$_C$ runs in polynomial time. The uncertainty interval of the associated edge instance needs to be computed for m edges. For the cases where $a \leq 3$ we only perform constant time operations. The analysis of V-DET$_C$ is similar to the one of BALANCE (see Remark 2.4), except that the computation of edge potentials is not needed. Hence, V-RANDOM$_C$ needs $O(mn)$ time without preprocessing and $O(m^2 \log(n))$ time with the preprocessing included.

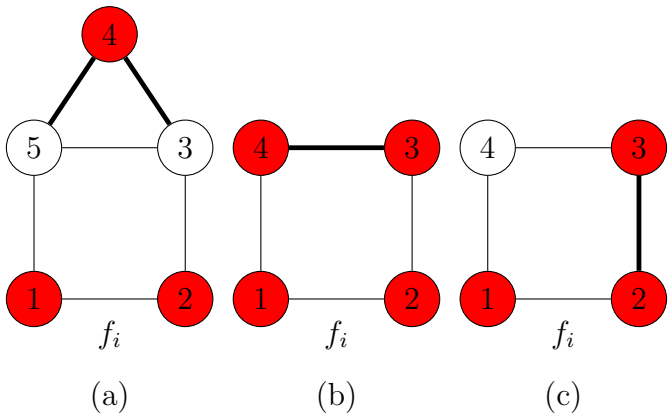


Figure 5.3: C_i with $b = 1$, where edges in $X(f_i)$ are bold. Vertices which are displayed in red are possible candidates for an optimal query set Q_i^* of size one. In (a) we have $a = 3$ such that the end vertices of f_i and the vertex in the intersection of the two neighbor edges are possible candidates for a size one query set. In (b) and (c) we have $a = 2$. Depending on whether f_i and the edge in the neighbor set share a vertex, there are either four or three candidates for a size one query set.

Remark 5.7. The algorithm V-RANDOM$_C$ is defined for uniform query costs and cannot be adapted for the non-uniform case in a straightforward way because it relies on witness sets, a concept which only makes sense for uniform query costs.

Let us see how the algorithm V-RANDOM$_C$ proceeds when applied to the following small instance of V-MST-U:

Example 5.1. Consider the instance as displayed in Figure 5.4 below.
Let $\epsilon > 0$ be small. All vertices, except for the vertices in $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ have trivial uncertainty sets. The length of edges between trivial vertices is ϵ and the distance between a trivial vertex and the boundary of an adjacent non-trivial vertex is ϵ too. Assume that the realisation of vertex positions is such that 3, 4, 5, 6 and 7 lie in the center of their uncertainty intervals and the locations of 1 and 2 are such that the length of $\{1, 2\}$ is $10 + 2\epsilon$. Moreover the vertices 8, 9 and 11 are located at distance at most ϵ from the far left end of their uncertainty sets, while 10 is located at distance at most ϵ from the far right end of A_{10}.
In the associated edge instance, the upper limits of all edges except for $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$, $\{6, 7\}$, $\{8, 9\}$ and $\{10, 11\}$ are at most $7 + \epsilon$. Thus these edges have to lie in an MST with certainty. In the associated edge instance we have $A_{\{1,2\}} = (10, 14)$, $A_{\{3,4\}} = (1, 11)$, $A_{\{5,6\}} = A_{\{6,7\}} = (6, 12.5)$, $A_{\{8,9\}} = (7, 11)$ and $A_{\{10,11\}} = (4, 8)$. The associated edge instance does not need preprocessing, as we already have $T_L = T_U = E \setminus \{\{1, 2\}, \{8, 9\}\}$. Assume that $p = 0.4$. For the non-trivial vertices we pick the label l as indicated by the black numbers in Figure 5.4. The trivial vertices are labelled

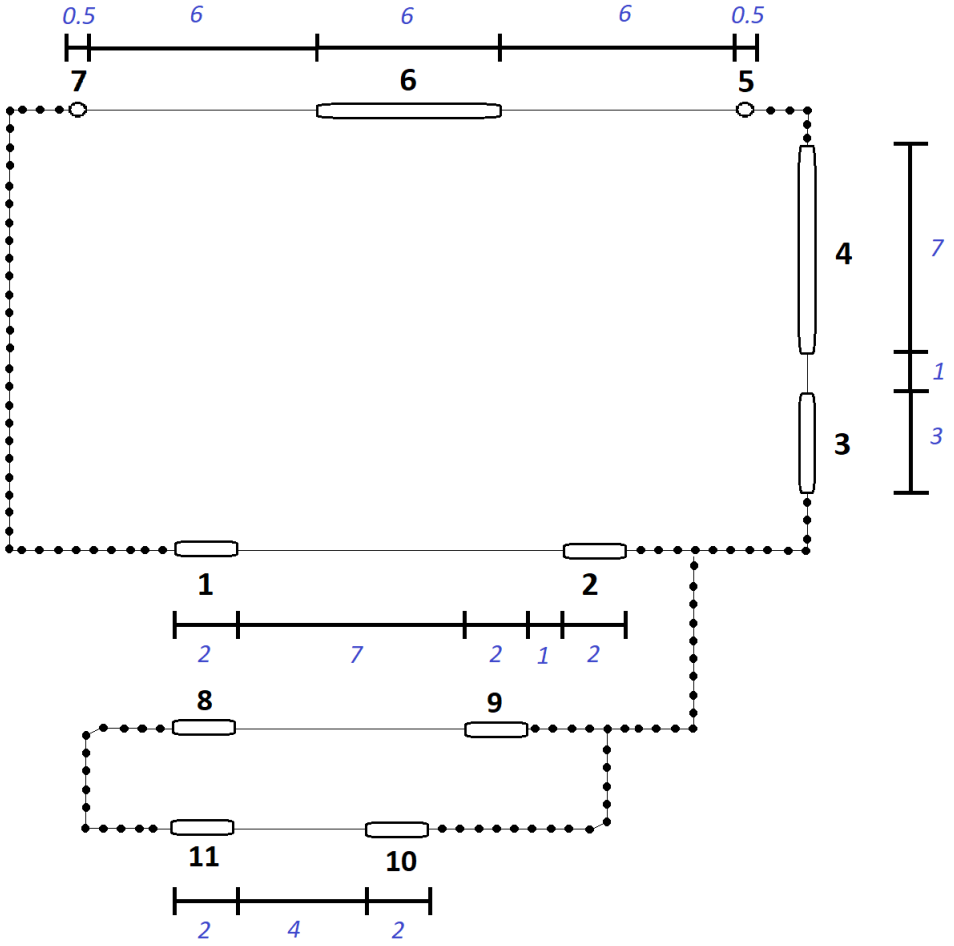


Figure 5.4: An instance of V-MST-U. Blue numbers correspond to distances, black numbers correspond to vertex labels of vertices with non-trivial uncertainty sets.

arbitrarily, their labels do not matter. Remember that we assume that the permutations are ordered lexicographically, e.g. $\sigma_1^{(4)} = (1,2,3,4)$, $\sigma_2^{(4)} = (1,2,4,3)$... We set $f_1 = \{1,2\}$, $f_2 = \{8,9\}$ and $Q := \emptyset$.

i=1 In the first iteration, we have $C_1 = (1,2,...,3,4,...,5,6,7,...)$ and $a = 5$. Hence we apply the deterministic subroutine V-DET$_C$. The edge $\{1,2\}$ is queried and it turns out that $\bar{w}_{\{1,2\}} = 10 + 2\epsilon$, which lies in the uncertainty sets of all three neighbor edges. Now $\{5,6\}$ is an edge with largest upper limit such that its end vertices are queried and added to $Q := \{5,6\}$. It turns out that $\bar{w}_{\{5,6\}} = 9.25$ and $A_{\{6,7\}}$ is set to $(9, 9.5)$. Now $\{3,4\}$ has the maximum upper limit among edges in C_1 and both end vertices are queried, i.e. $Q := \{3,4,5,6\}$. The edge $\{1,2\}$ is removed from Γ. Note

that an optimal solution queries 6 to see that the lengths of $\{5,6\}$ and $\{6,7\}$ both lie in $(9, 9.5)$ and 4, which reveals that the length of $\{3,4\}$ must lie in $(4.5, 7.5)$.

i=2 In the second iteration, we have $C_2 = (8, 9, ..., 10, 11, ...)$ and $a = 2$. As $\{10,11\}$ and $\{8,9\}$ are not adjacent, we have $k := 4$ and $r(8) = 1$, $r(9) = 2$, $r(10) = 3$ and $r(11) = 4$. As $\frac{9}{24} = 0.375$ and $\frac{10}{24} \approx 0.42$, we set $j = 10$. Note that $\sigma_{10}^{(4)} = (2,3,4,1)$. Then $\sigma_{10}^{(4)}(r(8)) = \sigma_{10}^{(4)}(1) = 2$, $\sigma_{10}^{(4)}(r(9)) = \sigma_{10}^{(4)}(2) = 3$, $\sigma_{10}^{(4)}(r(10)) = \sigma_{10}^{(4)}(3) = 4$ and $\sigma_{10}^{(4)}(r(11)) = \sigma_{10}^{(4)}(4) = 1$. Hence we start by querying 11 ($Q := \{3,4,5,6,11\}$) such that the uncertainty set of $\{10,11\}$ is updated to $A_{\{10,11\}} = (6-\epsilon, 8-\epsilon) \subset A_{\{8,9\}}$. Thus we cannot yet identify a maximum weight edge in C_2. Now we query 8 ($Q := \{3,4,5,6,8,11\}$), which yields that $\bar{w}_{\{8,9\}} \in (9-\epsilon, 11-\epsilon)$, i.e. $\{8,9\}$ has maximum weight in C_2. Note that an optimal solution for C_2 would have queried 8 only.
Hence, for this instance of V-MST-U the competitive ratio achieved by V-RANDOM$_C$ is $\frac{4+2}{2+1} = 2$, if $p = 0.4$.

5.2 Computing the MST Weight under Uncertainty

Another variant of the Minimum Spanning Tree Problem under Uncertainty is *MST Weight under Uncertainty* (W-MST-U) which not only asks for the edge set of an MST but also for the precise weight of an MST. W-MST-U was introduced by [15] who provided a query optimal algorithm that runs in polynomial time. The algorithm CUT-WEIGHT relies on the following well-known property of minimum spanning trees:

Proposition 5.3. *T is the edge set of a minimum spanning tree if and only if for every $e \in T$ it holds that if C is one of the two connected components in $T - e$, then e has minimum weight in the cut set $\delta(C)$.*

The algorithm CUT-WEIGHT starts with an arbitrary spanning tree Γ and iteratively deletes an edge $e \in \Gamma$. In each iteration, CUT-WEIGHT queries the edges in the cut which is defined by the two connected components of $\Gamma - e$ in order of increasing lower limits until a minimum weight edge f in the cut can be identified and its weight is known. Finally, Γ is updated by exchanging f and e. Algorithm 10 displays a formal description of CUT-WEIGHT.

Remark 5.8. The algorithm CUT-WEIGHT runs in $O(mn)$ time. Computing a spanning tree requires at most $O(m \log(n))$ time. Prior to the for-loop we sort the edges with respect to increasing lower limit and update the sorting whenever an edge is queried ($O(\log(n))$ time per query). In each iteration of the for-loop we need to find the cut edges ($O(m)$ time) and check whether the cut edge with smallest lower limit is trivial (which requires constant time.) The time needed for querying edges and

input : An instance of W-MST-U with graph $G = (V, E)$, uncertainty sets A_e and query costs q_e, $e \in E$
output: A feasible query set Q

```
1 Determine a spanning tree Γ;
2 Index the edges of Γ by f_1, ..., f_{n-1};
3 Initialize Q = ∅;
4 for i ← 1 to n − 1 do
5 |   Delete f_i from Γ and let S_i be the cut containing all edges between the
  |    two components of Γ;
6 |   while S_i does not contain a minimal edge with trivial uncertainty
  |    interval do
7 |   |   Choose g ∈ S_i s.t. L_g = min{L_e | e ∈ S_i};
8 |   |   Query g and add it to Q
9 |   Add a minimal edge in S_i to Γ
```

Algorithm 10: The algorithm CUT-WEIGHT for W-MST-U with general query costs (adapted from [15], page 1232)

updating the sorting sums up to $O(m \log(n))$ time in total. Hence, n iterations yield a run-time of at most $O(mn)$.

[15] showed that every edge queried during the execution of CUT-WEIGHT lies in any feasible query set for the W-MST-U instance:

Theorem 5.3. ([15], page 1232) *The algorithm CUT-WEIGHT outputs an optimal solution for W-MST-U.*

An intuitive motivation for choosing a cut-based algorithm over a cycle-based algorithm (such as BALANCE or U-RED), is that a cycle-based algorithm is biased to query edges outside the MST while CUT-WEIGHT identifies edges with minimum weight in a cut which ultimately lie inside the MST.

Example 5.2. The following example shall demonstrate how CUT-WEIGHT works. We will use again the instance as given in Example 2.3. We pick
$\Gamma := \{\{1,2\},\{1,7\},\{6,7\},\{4,6\},\{3,4\},\{5,6\}\}$ as a feasible spanning tree to start with and initialize $Q := \emptyset$.

i=1: In the first iteration we remove $\{1,2\}$ from Γ which results in the two components induced by the vertex sets $\{2\}$ and $\{1,3,4,5,6,7\}$. The cut set is given by $S_1 = \{\{1,2\},\{2,3\},\{2,5\}\}$. The edge $g := \{2,3\}$ has the minimum lower limit among the cut edges and is thus queried and then added to Γ. Hence $Q := \{\{2,3\}\}$.

i=2: Next we remove $\{1,7\}$ from Γ, which isolates the vertex 1. The cut set we obtain is $S_2 = \{\{1,2\},\{1,7\}\}$. The edge $\{1,7\}$ has the minimum lower limit among edges in S_2 and is trivial. Thus no query is made and $\{1,7\}$ is re-added to the tree.

i=3: When removing $\{6,7\}$ from Γ, we consider the cut set $S_3 = \{\{6,7\},\{1,2\}\}$ out of which $\{6,7\}$ is queried and added to the tree. $Q := \{\{2,3\},\{6,7\}\}$.

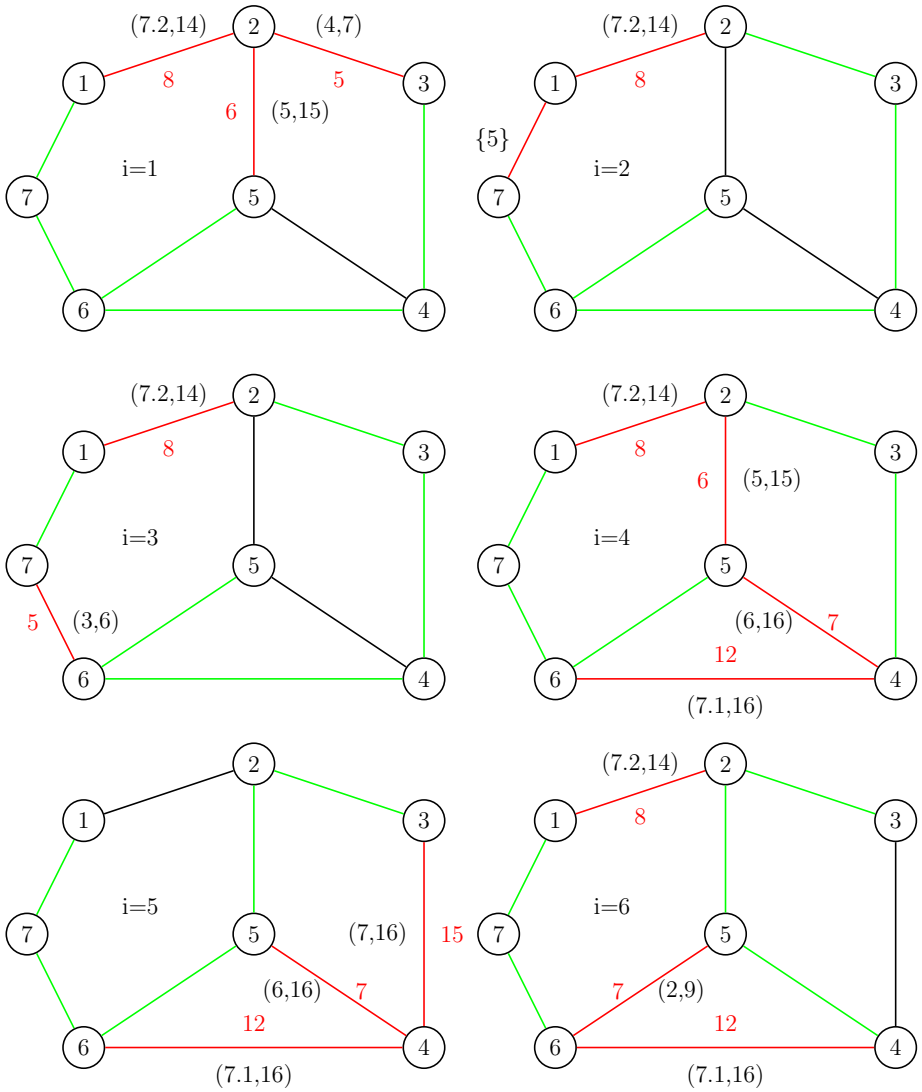


Figure 5.5: The six iterations of CUT-WEIGHT: Edges in green belong to Γ, while cut edges in S_i are red, $i = 1, ..., 6$.

i=4: The removal of $\{4,6\}$ leads to the components given by $\{1,5,6,7\}$ and $\{2,3,4\}$ and the cut set $S_4 = \{\{1,2\},\{4,6\},\{4,5\},\{2,5\}\}$. We query $\{2,5\}$ and obtain

$\bar{w}_{\{2,5\}} = 5$, which is less that the lower limit of any of the other edges in the cut. Thus we add $\{2,5\}$ to the tree and set $Q := \{\{2,3\},\{6,7\},\{2,5\}\}$.

i=5: By removing the edge $\{3,4\}$ from Γ, we isolate the vertex 4, hence S_5 consists of the edges incident to 4. We query $\{4,5\}$, which has the minimum lower limit among edges in the cut and see that it can be added to Γ. $Q := \{\{2,3\},\{6,7\},\{2,5\},\{4,5\}\}$.

i=6: In the last iteration we consider the cut edges in $S_6 = \{\{5,6\},\{1,2\},\{4,6\}\}$, which are obtained by removing $\{5,6\}$ from Γ. $\{5,6\}$ needs to be queried and is re-added to the spanning tree. $Q := \{\{2,3\},\{6,7\},\{2,5\},\{4,5\},\{5,6\}\}$.

Hence we found the minimum spanning tree
$\Gamma = \{\{2,5\},\{1,7\},\{6,7\},\{2,3\},\{4,5\},\{5,6\}\}$ as well as its weight 6+5+5+5+7+7=35 by querying the edges in the optimal query set $Q = \{\{2,3\},\{6,7\},\{2,5\},\{4,5\},\{5,6\}\}$.

5.3 The OP-OP Model

So far, queries were able to return the precise weight of the queried edge. In some settings however, it might be more realistic for a query to only reveal increasingly refined estimates of the edge weights.
Gupta et al. [12] consider the more general setting where a function $f(x_1, x_2, ...x_n)$ is to be computed while some of the x_i's are not fully specified but are known to lie in some interval. Again an algorithm can make queries about the x_i. If the input intervals are open (OP) and the queries return points (P), the model is referred to as OP-P. If the input consists of open intervals and queries reveal open subintervals of the original uncertainty sets, the model is called OP-OP.

A useful connection between the OP-P model and the OP-OP model is given by the following theorem, which was proved by [12]:

Theorem 5.4. ([12], page 6) *A witness algorithm for a problem under the OP-P model is a witness algorithm for the OP-OP version of the same problem.*

By Theorem 2.1, the algorithm U-RED extends to a 2-competitive algorithm for the OP-OP version of MST-U.
Again one might ask if randomization yields an improvement in terms of competitive ratio. Unlike in the OP-P version of MST-U, the answer to this question is no, as shown by [15]. They define an instance of the OP-OP version of MST-U such that no deterministic algorithm has expected ratio $\frac{ALG(G,R)}{OPT(G,R)}$ less than 2, where R is a realization drawn from a family $\mathcal{R}$ of feasible realization of edge weights according to a specific probability distribution p. Together with a variant of Yao's Principle (See Borodin and El-Yaniv [1], Theorem 8.5), this yields the following result shown by [15]:

Theorem 5.5. ([15], page 1238) *No randomized algorithm for MST-U under the OP-OP model can achieve competitive ratio $c < 2$.*

5.4 Approximate Minimum Spanning Trees

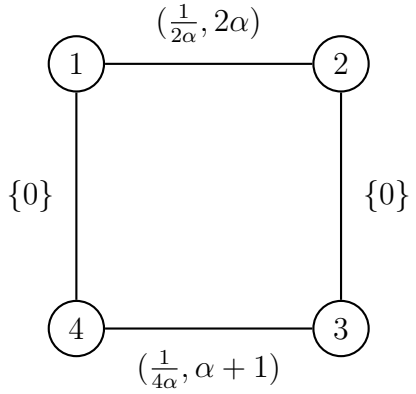
Another alternative of MST-U is to relax the necessity of an exact MST and aim for an α-approximate solution instead:

Definition 5.2. Given an instance of MST-U with graph $G = (V, E)$ and realization $\bar{w}_e$, $e \in E$, let T^* be the edge set of a minimum spanning tree. The α-approximate MST-U consists in finding the edge set of a spanning tree T such that $\sum_{e \in T} \bar{w}_e \leq \alpha \cdot \sum_{e \in T^*} \bar{w}_e$, while minimizing the query cost needed to find T. Note, that the requirement to find an MST is relaxed only for the algorithm, not for the optimum solution. An optimal query set still has to verify an exact MST.

However, this relaxation does not yield an improvement in terms of the known bounds for the competitive ratio, as proved in [15].

Theorem 5.6. ([15], page 1239) *For $\alpha > 1$, there is no c-competitive (randomized) algorithm for the α-approximate MST-U with $c < 2$ ($c < 1.5$).*

Proof. Let $\alpha > 1$ and consider the following instance as depicted in the figure below.



Note that $\{1, 4\}$ and $\{2, 3\}$ are in any MST. However, there might be an α-approximate spanning tree which does not contain both $\{1, 4\}$ and $\{2, 3\}$. Let Q be a query set and T' a spanning tree, such that querying the edges in Q verifies that T' is α-approximate. Now assume first that T' does not contain both $\{1, 4\}$ and $\{2, 3\}$. Let T be a spanning tree that contains both the trivial edges and let T^* be an MST.

Figure 5.6: Lower bound computation for the α-approximate MST-U. In this instance all query costs equal 1.

Then by querying edges in Q we know that $\bar{w}(T) \leq \bar{w}(T') \leq \alpha \cdot \bar{w}(T^*)$, where $\bar{w}$ denotes the sum of weights of edges in the respective tree. Thus, querying the edges in Q also guarantees that T is an α-approximate tree. Hence we can w.l.o.g. assume that a feasible query set Q aims to verify an α-approximate tree T which contains both edges of cost 0.

We first prove the deterministic bound. Assume a deterministic algorithm queries $\{1,2\}$ first. Consider the realization $\mathcal{R}_1$ with $\bar{w}_{\{1,2\}} = 1$, $\bar{w}_{\{3,4\}} = \frac{1}{3\alpha}$. As $1 \not\leq \alpha \cdot \frac{1}{2\alpha}$ and $\alpha + \frac{1}{2} \not\leq \alpha \cdot 1$, the algorithm has to query $\{3,4\}$ too to find an α-approximate spanning tree. An optimal solution, however, only queries $\{3,4\}$ to find an exact MST. Now we assume an algorithm queries $\{3,4\}$ first while the realization $\mathcal{R}_2$ is $\bar{w}_{\{1,2\}} = \frac{3\alpha+1}{2}$, $\bar{w}_{\{3,4\}} = 1$. Then the optimal solution only needs to query $\{1,2\}$ to find that $\{3,4\}$ is in an MST. The algorithm however, needs to query $\{1,2\}$ too because $1 \not\leq \alpha \cdot \frac{3}{4\alpha}$ and $\frac{3\alpha}{2} \not\leq \alpha \cdot 1$.
Now we turn to the case of randomized algorithms. A randomized algorithm queries edge $\{1,2\}$ first with probability p. If the realization is $\mathcal{R}_1$, then the expected query cost is $2p+(1-p)$. If the realization is $\mathcal{R}_2$, then the expected query cost is $p+2(1-p)$. Again the maximum expected query cost for these two instances is minimized by choosing $p = 0.5$. Hence no randomized algorithm can achieve a competitive ratio below 1.5. □

5.5 Minimum Matroid Base under Uncertainty

In this section we will see how the results for MST-U extend to the more general Minimum Weight Matroid Base Problem under Uncertainty (MMB-U).

Definition 5.3. A matroid is a pair $(E, \mathcal{I})$ with $\mathcal{I} \subset \mathcal{P}(E)$, such that

- $\emptyset \in \mathcal{I}$,
- If $I \in \mathcal{I}$ and $\tilde{I} \subset I$, then $\tilde{I} \in \mathcal{I}$,
- If $I, \tilde{I} \in \mathcal{I}$ and $|\tilde{I}| > |I|$, then $\exists\, e \in \tilde{I} \setminus I$, such that $I \cup \{e\} \in \mathcal{I}$.

We call a set $I \in \mathcal{I}$ independent and a set $C \in \mathcal{P}(E) \setminus \mathcal{I}$ dependent. A set $B \in \mathcal{I}$ is a base of the matroid if for all $e \in E \setminus B$ the set $B \cup \{e\}$ is not in $\mathcal{I}$. A minimal dependent set (with respect to inclusion) is called circuit.

Definition 5.4. For a matroid $(E, \mathcal{I})$ and a weight function $\bar{w} : E \to \mathbb{R}_{\geq 0}$, the Minimum Weight Matroid Base Problem (MMB) consists in finding a base $B \in \mathcal{I}$ of the matroid such that $\sum_{b \in B} \bar{w}_b$ is minimal.

For the Minimum Weight Matroid Base Problem under Uncertainty (MMB-U) we are given an uncertainty set A_e instead of each weight $\bar{w}_e$, which is guaranteed to lie inside A_e and can be determined by querying e at cost q_e. We assume each uncertainty set to be an open interval or a singleton. The goal is to find a minimum weight matroid base at minimum query cost.

MMB is a generalization of MST in the following sense: For a connected graph $G = (V, E)$, define the matroid $M := (E, \mathcal{I})$, where $\mathcal{I} := \{A \subset E \mid (V, A) \text{ is cycle-free}\}$.

Then B is a basis of M iff B is the edge set of a spanning tree. Hence the MST in G translates to the MMB in M. A matroid whose independent sets are the forests of a (not necessarily connected) graph is called *graphic*.
Algorithms which have been presented for the MST-U can be modified in order to work for MMB-U too: For the case where query costs are uniform, Erlebach et al. [6] show that the algorithm U-RED can be generalized to work for MMB-U as well and achieves competitive ratio 2. For general query costs, the algorithms BALANCE and RANDOM can also be adapted such that they work in the setting of matroids and achieve competitive ratio 2 and $1 + \frac{1}{\sqrt{2}}$ respectively, as shown by [15]. The algorithm CUT-WEIGHT which will be discussed in Section 5.2 and computes the exact weight of an MST in the setting of explorable uncertainty such that the set of edges which are queried has minimum cost also translates to the matroid case where is achieves competitive ratio 1 too (see [15]).

Megow et al. [15] present two additional algorithms for MMB-U with uniform query costs: the best-in greedy algorithm CYCLE and the worst-out greedy algorithm CUT.

5.5.1 The algorithm CYCLE

CYCLE merges ideas from BALANCE and U-RED. Thus we assume that a *lower (upper) limit basis* B_L (B_U) is defined analogously to a lower (upper) limit tree in Section 2.3.2, the ordering "$<$" of elements with respect to lower limits is an analogue of Definition 2.2 and an *always maximal element* in a circuit is defined in the same way as an always maximal edge in a cycle (see Definition 2.3). The framework of CYCLE is similar to the one of BALANCE or RANDOM. We start out with an initial basis, a lower limit basis B_L. Then we add the remaining elements f_i one after the other, in order of non-decreasing lower limits. Each time that we add such an element, we consider the circuit C_i which arises from adding f_i to the current basis. By querying elements until an always maximal element in C_i can be identified we make sure that we maintain a partial solution. The edges that we query in a specific circuit however, are not picked in the same way as in the algorithm BALANCE. Instead, CYCLE proceeds in a similar way as U-RED: It chooses an element f with maximum upper limit in C_i and and an element g in $C_i \setminus \{f\}$ such that $A_f \cap A_g \neq \emptyset$ and queries both. Note that we do not need to assume $B_L = B_U$ here, because we do not require that $f = f_i$. A precise description of CYCLE is displayed in Algorithm 11.

Megow et al. [15] show that the algorithm CYCLE achieves competitive ratio 2:

Theorem 5.7. ([15], page 1234) *The algorithm CYCLE outputs a feasible query set for MMB-U with uniform query costs and is 2-competitive.*

input : An instance of MMB-U with matroid $M = (E, \mathcal{I})$, uncertainty sets A_e, $e \in E$ and uniform query costs
output: A feasible query set Q

Determine B_L and set $\Gamma := B_L$;
Index the elements in $R := E \setminus B_L$ such that $f_1 \leq ... \leq f_k$;
Initialize $Q := \emptyset$;
for $i \leftarrow 1$ **to** k **do**
  Add f_i to Γ and let C_i be the occurring circuit;
  while C_i *does not contain an always maximal element* **do**
    Choose $f \in C_i$ s.t. $U_f = \max\{U_e | e \in C_i\}$;
    Choose $g \in C_i \setminus \{f\}$ with $U_g > L_f$;
    Add f and g to Q and query them
  Delete the maximum weight element $\bar{e}$ from Γ
Return Q

Algorithm 11: The algorithm CYCLE for MMB-U with uniform query costs (adapted from [15], page 1234)

Example 5.3. Consider the matroid $M = (\{a, b, c, d\}, \mathcal{I})$ with $\mathcal{I} = \{I \in \mathcal{P}(\{a, b, c, d\}) \mid |I| \leq 2\}$. Let $A_a = (2, 5)$, $A_b = (1, 4)$, $A_c = (2, 3)$ and $A_d = (3, 6)$ with weights $\bar{w}_a = 3$, $\bar{w}_b = 2$, $\bar{w}_c = 2.5$ and $\bar{w}_d = 4$. Note that all subsets of size 2 are bases of M. $B_L := \{a, b\}$ is a lower limit basis. Hence $f_1 = c$, $f_2 = d$ and we set $\Gamma := \{a, b\}$.

i=1: We add c to Γ. Then $C_1 = \{a, b, c\}$ is the circuit in Γ, $f := a$ has the largest upper limit in C_1 and $g := c$ is such that $A_c \cap A_a = (2, 3) \cap (2, 5) \neq \emptyset$. We query a and c. As $\bar{w}_c = 2.5 \in (1, 4) = A_b$, another iteration of the while-loop is required such that b is queried too. CYCLE removes a from Γ because it has largest weight in the circuit C_1, i.e. $\Gamma = \{b, c\}$, $Q := \{a, b, c\}$.

i=2: We add d to Γ. We have $C_2 = \{b, c, d\}$. Note that d is always maximal in C_2 because $L_d = 3 > 2 = \bar{w}_b$ and $L_d = 3 > 2.5 = \bar{w}_c$. Hence we know that $\Gamma := \{b, c\}$ is a minimum weight basis and output Q.

Note that an optimal solution would have queried a and b only because we need not query c to see that $\bar{w}_a > U_c$ and $L_d > U_c$. This means that CYCLE achieves competitive ratio 1.5 when applied to this instance if it picks $g := c$ in the first iteration. Had it picked $g := b$ in the first iteration it would have even found an optimal solution.

Remark 5.9. The matroid M in Example 5.3 is uniform (the independent sets are

precisely the sets of size at most 2) but not graphic.

5.5.2 The algorithm CUT

The algorithm CUT starts out with an upper limit basis B_U instead of a lower limit basis and deletes elements in order of decreasing upper limit. More precisely, we define the ordering of elements as follows:

Definition 5.5. For two elements $e, f \in E$ we say that $e >_u f$ if $U_e > U_f$ or $U_e = U_f$ and $L_e > L_f$. If $e >_u f$ or $e = f$ holds, we say that $e \geq_u f$.

Each time we delete an element g_i from the current basis, we consider the analogue of a cut set, namely the elements in $S_i = \{e \in E \mid \Gamma \cup \{e\} \text{ is a basis}\}$, where Γ denotes the basis found in the previous iteration minus the deleted element g_i. By making queries, we identify a minimum weight edge in S_i which is consequently picked to complete the basis. The algorithm CUT is displayed in Algorithm 12.

input : An instance of MMB-U with matroid $M = (E, \mathcal{I})$, uncertainty sets A_e, $e \in E$ and uniform query costs
output: A feasible query set Q

Determine B_U and set $\Gamma := B_U$;
Index the elements in B_U such that $g_1 \geq_u ... \geq_u g_n$;
Initialize $Q := \emptyset$;
for $i \leftarrow 1$ **to** n **do**
    Delete g_i from Γ;
    Let $S_i = \{e \in E \mid \Gamma \cup \{e\} \text{ contains a basis}\}$;
    while *we cannot identify a minimum weight element in* S_i **do**
        Choose $g \in S_i$ s.t. $L_g = \min\{L_e | e \in S_i\}$;
        Choose $f \in S_i \setminus \{g\}$ with $L_f < U_g$;
        Add f and g to Q and query them
    Add the minimum weight element $\bar{e}$ from S_i to Γ
Return Q

Algorithm 12: The algorithm CUT for MMB-U with uniform query costs (adapted from [15], page 1235)

CUT achieves competitive ratio 2, as shown by [15]:

Theorem 5.8. ([15], page 1236) *The algorithm CUT finds a feasible query set for MMB-U with uniform query costs and is 2-competitive.*

Example 5.4. We consider the same instance of MMB-U with matroid M as in Example 5.3. The choice of the upper limit basis $B_U = \{b, c\}$ is unique. Then $g_1 = b$ and $g_2 = c$. Set $\Gamma := \{b, c\}$ and $Q := \emptyset$.

i=1: First CUT removes b from Γ. As all subsets of size 2 form a basis, we have $S_1 = \{a, b, d\}$. The element b has smallest lower limit among elements of S_1, i.e. $g := b$. Assume CUT picks $f = d$, because $A_d \cap A_b = (3, 6) \cap (1, 4) \neq \emptyset$ and queries both edges. It becomes clear that b has smallest weight in S_1, i.e. $\Gamma := \{b, c\}$ and $Q := \{b, d\}$.

i=2: CUT removes c from Γ. This yields $S_1 = \{a, c, d\}$. Assume CUT picks $g := c$ and $f := a$. Then querying c and a yields that c needs to be re-included into Γ and we output $Q := \{a, b, c, d\}$.

Remember that $\{a, b\}$ is an optimal query set. This means that CUT can achieve its worst case competitive ratio of 2 when applied to this instance. Had CUT chosen $f := a$ in the first iteration, it would have found an optimal solution.

5.6 Special Spanning Trees

Instead of identifying a general minimum spanning tree, one could also ask to find a minimum spanning tree of a specified type at minimum query cost. However, for many types of special spanning trees, it is already NP-hard to find such a tree in a graph, e.g. Hamiltonian paths, spiders (see Gargano et al. [10]) or caterpillars (see Khosravani [14]).
A special spanning tree that can be found in polynomial time is a star tree.

Definition 5.6. The complete bipartite graph $K_{1,k}$ is called star tree S_k. If a graph G with n vertices contains S_{n-1} as a subgraph, then S_{n-1} is said to be a spanning star in G.

We define the *Minimum Spanning Star Problem under Explorable Uncertainty* (MSS-U) analogously to MST-U. However, it turns out that no algorithm for MSS-U can achieve constant competitive ratio.

Proposition 5.4. *There exists no algorithm for MSS-U which achieves constant competitive ratio.*

Proof. Consider a graph $G = (V, E)$ with $V = \{1, ..., n\}$ and
$E = \{\{1, v\} | v = 2, ..., n\} \cup \{\{n, v\} | v = 1, ..., n-1\}$. Then G has precisely two spanning stars with center 1 resp. n (all other vertices have degree 2.) Let $q_e = 1$ for all $e \in E$.

The uncertainty sets are defined as follows:

- $A_{\{1,n\}} = \{1\}$,
- $A_{\{1,k\}} = (0,1)$ for $k = 2, ..., n-1$ and
- $A_{\{n,k\}} = (0,n)$ for $k = 2, ..., n-1$.

Now assume the realization of edge weights is such that $\bar{w}_{\{j,n\}} = n-1$ for some arbitrary but fixed $j \in \{2, ..., n-1\}$ and $\bar{w}_e = 0.5$ for all $e \in E \setminus \{\{j,n\}, \{1,n\}\}$. Then an optimal solution queries only $\{j,n\}$ to know that the spanning star with center n has larger weight. Conversely, without querying $\{j,n\}$, it is impossible to tell which spanning star has minimum weight. For each possible order in which an algorithm queries edges in G, we can choose an instance as above with j such that $\{j,n\}$ is the last edge of type $\{k,n\}$ with $k \in \{2, ..., n-1\}$ to be queried by the algorithm. Hence no algorithm achieves competitive ratio better than $n-2$. For $n = 5$ the construction is illustrated in Figure 5.7. □

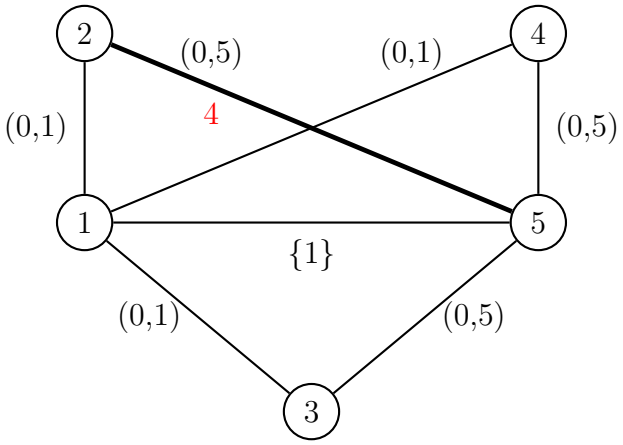


Figure 5.7: The weight of $\{2,5\}$ is displayed in red. All missing edge weights equal 0.5. An optimal solution only needs to query $\{2,5\}$ which has larger weight than the star with center 1 can have. A deterministic algorithm can not distinguish between the edges $\{2,5\}, \{3,5\}$ and $\{4,5\}$ and might have to query all three of them.

Conclusion

In this master's thesis we have seen different deterministic as well as randomized algorithms for the Minimum Spanning Tree Problem Under Explorable Uncertainty and other variants of the problem. The following table gives an overview over the performance of the algorithms as well as over existing lower bounds.

	BALANCE	U-RED	RANDOM	RANDOM$_C$	V-RANDOM$_C$	CUT-WEIGHT	BOUND DET.	BOUND RAND.
MST-U	2	2	$1+\frac{1}{\sqrt{2}}$	-	-	-	2	1.5
MST-U cactus	2	2	$1+\frac{1}{\sqrt{2}}$	1.5	-	-	2	1.5
V-MST-U	-	4	-	-	-	-	4	2.5
V-MST-U special case	-	4	-	-	3	-	4	2.5
MMB-U	2	2	$1+\frac{1}{\sqrt{2}}$	-	-	-	2	1.5
MMB-U Weight	-	-	-	-	-	1	1	1
MST-U Weight	-	-	-	-	-	1	1	1
Approx. MST-U	2	2	$1+\frac{1}{\sqrt{2}}$	-	-	-	2	1.5
Op-Op	-	2	-	-	-	-	2	2
MSS-U	-	-	-	-	-	-	no const.	no const.

Table 5.1: This table lists the results for uniform query costs. The results are identical for the non-uniform case, except for the algorithms U-RED and V-RANDOM$_C$ which cannot be applied in the latter case. The entry "-" is used whenever an algorithm cannot be applied to the specific problem. Note that algorithm names refer to the original algorithms for MST-U, as well as to adaptions of the original algorithms for different settings.

The MST-U in itself is a problem which still deserves further investigation. A major open question is whether there exist randomized algorithms with a smaller performance guarantee than RANDOM or whether the lower bound of 1.5 can be improved. Also note that there is no randomized algorithm for the V-MST-U so far and even for the special case where cycles intersect in trivial vertices only, there is a gap of 0.5 between the lower bound and the performance of V-RANDOM$_C$. Also for the deterministic case the version of V-MST-U with non-uniform query costs remains unsolved. Moreover, different models of the uncertainty exploration could be the subject of further investigations. So far, only adaptive models have been considered. Erlebach and Hoffmann [5] suggested a partly non-adaptive concept, where queries are made in rounds. Queries of the same round have to be made in parallel and depend solely on the query outcome of previous rounds. Different *round competitive models*, e.g. models where the objective function is the minimization of the number of rounds while

the number of queries per round is fixed or models where the goal is to minimize the number of queries while the number of rounds should not exceed a value k could be considered in future work on the topic.

Bibliography

[1] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[2] R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient update strategies for geometric computing with uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005.

[3] C. Dürr, T. Erlebach, N. Megow, and J. Meißner. Scheduling with Explorable Uncertainty. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[4] T. Erlebach and M. Hoffmann. Minimum spanning tree verification under uncertainty. In *Proceedings of the International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 164–175, 2014.

[5] T. Erlebach and M. Hoffmann. Query-competitive algorithms for computing with uncertainty. *Bulletin of the European Association for Theoretical Computer Science*, 116, 2015.

[6] T. Erlebach, M. Hoffmann, and F. Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Theoretical Computer Science*, 613:51–64, 2016.

[7] T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihalák, and R. Raman. Computing minimum spanning trees with uncertainty. In *Proceedings of Symposium on Theoretical Aspects of Computer Science*, pages 277–288, 2008.

[8] T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62(1):1–18, 2007.

[9] J. Focke, N. Megow, and J. Meißner. Minimum Spanning Tree under Explorable Uncertainty in Theory and Experiments. In *16th International Symposium on Experimental Algorithms (SEA 2017)*, volume 75 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[10] L. Gargano, M. Hammar, P. Hell, L. Stacho, and U. Vaccaroa. Spanning spiders and light-splitting switches. *Discrete Mathematics*, 285:83–95, 2004.

[11] M. Goerigk, M. Gupta, J. Ide, A. Schöbel, and S. Sen. The robust knapsack problem with queries. *Computers and Operations Research*, 55:12–22, 2015.

[12] M. Gupta, Y. Sabharwal, and S. Sen. The update complexity of selection and related problems. *Computing Research Repository*, abs/1108.5525, 2011.

[13] S. Kahan. A model for data in motion. *23rd Annual ACM Symposium on Theory of Computing (STOC'91)*, pages 267–277, 1991.

[14] M. Khosravani. Searching for Optimal Caterpillars in General and Bounded Treewidth Graphs. PhD Thesis, Department of Computer Science, University of Auckland, 2011.

[15] N. Megow, J. Meißner, and M. Skutella. Randomization helps computing a minimum spanning tree under uncertainty. *SIAM Journal on Computing*, 46(4):1217–1240, 2017.

[16] Y. Wang and S.-W. Wong. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 1070–1081, 2015.

List of Algorithms

List of Figures