RANDOMIZATION HELPS COMPUTING A MINIMUM SPANNING TREE UNDER UNCERTAINTY*

NICOLE MEGOW†, JULIE MEIßNER‡, AND MARTIN SKUTELLA‡

Abstract. Given a graph with "uncertainty intervals" on the edges, we want to identify a minimum spanning tree by querying some edges for their exact edge weights which lie in the given uncertainty intervals. Our objective is to minimize the number of edge queries. It is known that there is a deterministic algorithm with best possible competitive ratio 2 (Erlebach et al. [7]). Our main result is a randomized algorithm with expected competitive ratio $1 + 1/\sqrt{2} \approx 1.707$, solving the long-standing open problem whether an expected competitive ratio strictly less than 2 can be achieved (Erlebach and Hoffmann [4]). We also present novel results for various extensions, including arbitrary matroids and more general querying models.

Key words. Online Algorithms, Competitive Analysis, Minimum Spanning Tree, Randomized Algorithms

AMS subject classifications. 68W20, 68W27, 68R10

1. Introduction.

Uncertainty in the input data is an omnipresent issue in most real world planning processes. The quality of solutions for optimization problems with uncertain input data crucially depends on the amount of uncertainty. More information, or even knowing the exact data, allows for significantly improved solutions (see e. g., [17]). In general, it is impossible to fully avoid uncertainty. Nevertheless, it is sometimes possible to obtain exact data, but this may involve certain exploration cost in time, money, energy, bandwidth, etc. A classical application are estimated user demands that can be specified by undertaking a user survey, but this is an investment in terms of time and/or cost. Other applications include approximate positions of mobile devices, whose exact position can be determined at a cost, or insufficient information on existing infrastructure for telecommunication network planning, where a field measurement can reveal the capacity of an existing connection.

In this paper we are concerned with fundamental combinatorial optimization problems with uncertain input data that can be explored at a certain cost. We mainly focus on the minimum spanning tree (MST) problem with uncertain edge weights. In a given graph, we know initially for each edge only an interval containing the edge weight. The true value is revealed upon request (we say query) at a given cost. The task is to determine a minimum-cost adaptive sequence of queries to find a minimum weight spanning tree. In the basic setting, we only need to guarantee that the obtained spanning tree is minimal and we do not need to compute its actual weight, i. e., there might be tree edges whose weight we never query, as they appear in an MST independent of their exact weights. We measure the performance of an algorithm by competitive analysis. For any realization of edge weights, we compare the query cost of an algorithm with the optimal query cost. This is the cost for verifying an MST for a given fixed realization.

*Submitted to the editors 05.08.2016. An extended abstract of this work appeared in Proceedings of the 23rd European Symposium on Algorithms [15].

Funding: This research was carried out in the framework of Matheon supported by Einstein Foundation Berlin and by the German Science Foundation (DFG) under contract ME 3825/1.

†Department of Mathematics and Computer Science, University of Bremen, Germany (nicole.megow@uni-bremen.de)

‡Department of Mathematics, Technische Universität Berlin, Germany ({jmeiss,skutella}@math.tu-berlin.de)

As our main result we develop a randomized algorithm that improves upon the competitive ratio of any deterministic algorithm. This solves an important open problem in this area [4]. We also present the first algorithms for non-uniform query costs and generalize the results to matroids with uncertain weights, in both settings matching the best known competitive ratios for MST with uniform query cost.

Related work. The huge variety of research streams dealing with optimization under uncertainty reflects its importance for theory and practice. The major fields are online optimization [3], stochastic optimization [2], and robust optimization [1], each modeling uncertain information in a different way. Typically these models do not provide the possibility to influence when and how uncertain data is revealed. Kahan [12] was probably the first to study algorithms for explicitly exploring uncertain information in the context of finding the maximum and median of a set of values known to lie in given uncertainty intervals. Erlebach and Hoffmann recently compiled a survey of the research on this uncertainty model [4].

The MST problem with uncertain edge weights was introduced by Erlebach et al. [7]. Their deterministic Algorithm U-RED achieves competitive ratio 2 for uniform query cost when all uncertainty intervals are open intervals or trivial (i. e., containing one point only). They also show that this ratio is optimal and can be generalized to the problem of finding a minimum weight basis of a matroid with uncertain weights [6]. According to [4] it remained a major open problem whether randomized algorithms can beat competitive ratio 2. The offline problem of finding the optimal query set for a given realization of edge weights can be solved optimally in polynomial time [5].

Further problems studied in this uncertainty model include finding the k-th smallest value in a set of uncertainty intervals [9, 11, 12] (also with non-uniform query cost [9]), caching problems in distributed databases [16], computing a function value [13], and classical combinatorial optimization problems, such as shortest path [8], finding the median [9], and the knapsack problem [10].

A generalized exploration model was proposed in [11]. The OP-OP model reveals, upon an edge query, a refined open or trivial subinterval and might, thus, require multiple queries per edge. It is shown in [11] that Algorithm U-RED can be adapted and still achieves competitive ratio 2. The restriction to open intervals is not crucial as slight model adaptions allow to deal with closed intervals [11, 12].

While most works aim for minimal query sets to guarantee exact optimal solutions, Olsten and Widom [16] initiate the study of trade-offs between the number of queries and the precision of the found solution. They are concerned with caching problems. Further work in this vein can be found in [8, 9, 13].

Our contribution. After presenting lower bounds and structural insights in Section 3, we affirmatively answer the question whether randomization helps to minimize query cost in order to find an MST. In Section 4 we describe an algorithm framework underlying both our randomized algorithm as well as the result for non-uniform query cost. In Section 5 we present this randomized algorithm with tight competitive ratio 1.707, thus beating the best possible competitive ratio 2 of any deterministic algorithm.

One key observation is that the minimum spanning tree problem under uncertainty can be interpreted as a generalized online bipartite vertex cover problem. A similar connection for a *given* realization of edge weights was established in [5] for the related MST verification problem. In our case, new structural insights allow for a preprocessing which suggests a unique bipartition of the edges for *all* realizations simultaneously. Our algorithm borrows and refines ideas from a recent water-filling algorithm for the online bipartite vertex cover problem [18].

In Section 6 we consider the more general non-uniform query cost model in which each edge has an individual query cost. We observe that this problem can be reformulated within a different uncertainty model, called OP-OP, presented in [11]. The 2-competitive algorithm in [11] is a pseudo-polynomial 2-competitive algorithm for our problem with non-uniform query cost. We design new direct and polynomial-time algorithms that are 2-competitive and 1.707-competitive in expectation using the framework from Section 4. To that end, we employ a new strategy carefully balancing the query cost of an edge and the number of cycles it occurs in.

In Section 7 we consider the problem of computing the exact value of an MST under uncertain edge weights. While previous algorithms (U-RED [7], our algorithms) iteratively try to identify the largest-weight edge on a cycle, we now attempt to detect minimum-weight edges separating the graph into two components. This yields an algorithm based on cuts that is optimal for computing the exact value of an MST under uncertain edge weights. Interestingly, the same algorithm solves the original problem (only computing the MST, not its exact value) and achieves the same, best-possible competitive ratio 2, as the cycle-based algorithm presented in [7].

In Section 8 we observe in a broader context that these two algorithms can be interpreted as the best-in and worst-out greedy algorithm on matroids.

Finally, we show in Section 9 that in the adapted model where a query reveals only a subinterval and thus several queries per edge might be necessary, randomization does not allow for an improvement over worst-case competitive ratio 2.

Furthermore, we prove in Section 10 that finding an approximate MST under uncertainty also does not improve the worst-case bounds on the competitive ratio.

2. Problem Definition and Notation.

Initially we are given a weighted, undirected, connected graph $G = (V, E)$, with $|V| = n$ and $|E| = m$. Each edge $e \in E$ comes with an *uncertainty interval* A_e and possibly a *query cost* c_e. The uncertainty interval A_e constitutes the only information about e's unknown weight $w_e \in A_e$. We assume that an interval with lower limit L_e and upper limit U_e is either trivial, i.e., $A_e = [L_e, U_e], L_e = w_e = U_e$, or it is open, $A_e = (L_e, U_e), L_e < U_e$. We refer to such an instance of our problem as an *uncertainty graph.* A *realization* $\mathcal{R}$ *of edge weights* $(w_e)_{e \in E}$ for an uncertainty graph is *feasible*, if all edge weights w_e, $e \in E$, lie in their corresponding uncertainty intervals, i.e., $w_e \in A_e$.

The task is to find a minimum spanning tree (MST) in the uncertainty graph G for an a priori unknown, feasible realization $\mathcal{R}$ of edge weights. To that end, we may query any edge $e \in E$ at cost c_e and obtain its exact weight w_e according to $\mathcal{R}$. The goal is to design an algorithm that constructs a sequence of queries that determines an MST at minimum total query cost. For a realization $\mathcal{R}$ of edge weights, a set of queries $Q \subseteq E$ is *feasible*, if an MST can be determined given the exact edge weights for edges in Q only; that is, given w_e for $e \in Q$, there is a spanning tree which is minimal for any realization of edge weights $w_e \in A_e$ for $e \in E \setminus Q$. We denote this problem as *MST with edge uncertainty* and say *MST under uncertainty* for short.

Note that this problem does not necessarily involve computing the actual MST weight. We refer to the problem variant in which the actual MST weight must be computed as *computing the MST weight under uncertainty.* We also briefly consider the problem in which the query to an edge e does not necessarily reveal the exact edge weight, but may reveal a new uncertainty interval, as subinterval of the previous one instead. Here the input is a sequence of intervals $A_e^1 \supseteq A_e^2 \supseteq \dots$ and the query set is a multiset of the edges.

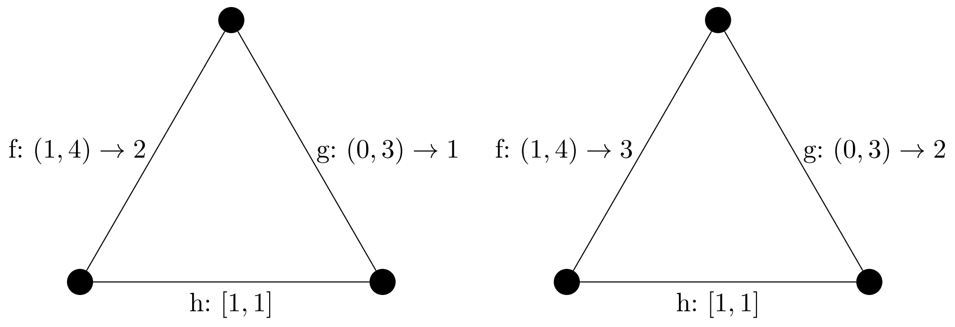

FIGURE 1. *Lower Bound Example with Realization $\mathcal{R}_1$ (left) and Realization $\mathcal{R}_2$ (right). The edge labels "$e : (L_e, U_e) \to w_e$" give edge e's uncertainty interval (L_e, U_e) as well as its (a priori unknown) weight w_e in a particular realization.*

We also consider the generalization of the problem to matroids. Given a ground set of elements X and a family of independent sets $\mathcal{I} \subseteq 2^X$, we call a matroid $M = (X, \mathcal{I})$ with an uncertainty interval A_x for each element $x \in X$ instead of the element's weight an *uncertainty matroid*. We define a query and its cost exactly as for the MST problem and refer to the problem of finding a minimum weight matroid base in an uncertainty matroid using a minimal number of queries as *Matroid base under uncertainty*.

We evaluate our algorithms by standard competitive analysis. An algorithm is *c-competitive* if, for any realization $(w_e)_{e \in E}$, the solution query cost is at most c times the optimal query cost for this realization. The optimal query cost is the minimum query cost that an offline algorithm (knowing the realization of edge weights) must pay to verify an MST. The *competitive ratio* of an algorithm ALG is the infimum over all c such that ALG is c-competitive. For randomized algorithms we compare the expected query cost to the optimal query cost. Competitive analysis addresses the problem complexity evolving from the uncertainty in the input, possibly neglecting any computational complexity. However, we note that all our algorithms run in polynomial time unless explicitly stated otherwise.

In the last section, we briefly consider approximation algorithms for *MST under uncertainty*. An algorithm is α-approximate if, for any realization $(w_e)_{e \in E}$, the query set Q found by the algorithm identifies a spanning tree of weight at most α times the weight of an MST. The approximation ratio of an algorithm ALG is the infimum over all α, such that ALG is α-approximate.

3. Lower Bounds, Intuition and Structural Insight.

The basis for understanding the behavior of uncertainty intervals and queries is their interplay on a cycle. This simple graph structure showcases both, lower bound examples and insights about the structure of a feasible query set. Consider a triangle with edge weights such that one edge is in any MST and the other two have overlapping uncertainty intervals (cf. Figure 1). We cannot decide which of the two edges is in the MST without querying at least one of them. Any deterministic algorithm decides to query either edge f or edge g first. If it decides to query edge f first, the algorithm has competitive ratio 2 for the realization $\mathcal{R}_1$, where the weight of edge f lies in the uncertainty interval of edge g. As the weight of edge g is not in the uncertainty interval of edge f, the optimal query set is $\{g\}$. Symmetrically the realization $\mathcal{R}_2$ reveals competitive ratio 2

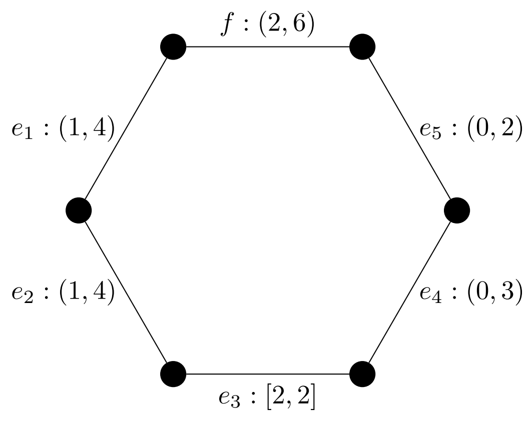

FIGURE 2. *Cycle with edge f and edges e_1, e_2, e_4 as additional candidates for being maximal.*

for all algorithms that query edge g first. Thus, as was already observed in [7], no deterministic algorithm can achieve a competitive ratio smaller than 2.

Next we consider randomized algorithms for the instance given in Figure 1. Each algorithm queries edge f with a certain probability first. We compute the expected competitive ratio for the two realizations $\mathcal{R}_1, \mathcal{R}_2$ parametrized by this probability. It is easy to observe that the best randomized algorithm queries both edges with probability 1/2 and has expected competitive ratio 1.5. This surprisingly easy example yields the best known lower bound on the competitive ratio for randomized algorithms. This lower bound was independently observed by Erlebach and Hoffmann [4].

We can already observe important problem features when considering a more general cycle (cf. Figure 2). To verify an MST on a cycle, we only need to identify an edge of maximal weight. Then there is an MST that does not contain this edge, and we can delete it [14]. We call such an edge *maximal.* An edge f with the largest upper limit U_f is a natural candidate for being maximal. We first observe that for each such edge f, unless we query it, we cannot prove it is contained in an MST, as it has the largest upper limit.

OBSERVATION 1. *Given a cycle C, where no edge is known to be maximal, let f be some edge with largest upper limit U_f. Let $\mathcal{R}$ be a feasible realization of edge weights, for which f is in an MST, then f is contained in any feasible query set for $\mathcal{R}$.*

We furthermore observe two different possibilities for proving that an edge f with largest upper limit U_f is in no MST. If edge f has the unique largest lower limit, the only other edges that are candidates for being maximal are the ones whose uncertainty interval overlaps with that of edge f. In Figure 2 these are the edges e_1, e_2, e_4. To find a maximal edge in the cycle we can query edge f to prove its edge weight is larger than the upper limit of all other edges. We can also query instead all edges with overlapping uncertainty interval and show their edge weight does not exceed f's lower limit. If edge f does not have the unique largest lower limit, the latter option is not feasible. Thus f must be in any feasible query set. This observation is strengthened and generalized in the next section in Lemma 6.

OBSERVATION 2. *Let f be some edge with largest upper limit U_f on a cycle C which does not have a maximal edge.*

(i) For any realization $\mathcal{R}$, every feasible query set contains edge f or all edges in C

whose uncertainty interval overlaps that of edge f.

(ii) Unless edge f has the unique largest lower limit L_f in C, it is in every feasible query set for any realization $\mathcal{R}$.

Structural Insight. The algorithm we design starts out with a minimum spanning tree for the particular realization where the weight of each edge is set to its lower limit. All other edges are considered in order of increasing lower limit and the algorithm iteratively tries to add an edge to the current spanning tree, thus closing a cycle. By construction, this cycle is closed by its edge with largest lower limit. The following structural insight shows that we can preprocess any instance such that this edge also has the largest upper limit in the cycle. In particular, we can apply Observation 2 to this edge.

Given an uncertainty graph $G = (V, E)$, consider the following two MSTs for extreme realizations. The *lower limit tree* $T_L \subseteq E$ is an MST for the realization w^L, in which all edge weights of edges with non-trivial uncertainty interval are close to their lower limits, more precisely $w_e^L = L_e + \varepsilon$ for infinitesimally small $\varepsilon > 0$. Symmetrically, the *upper limit tree* $T_U \subseteq E$ is an MST when the same edges have weight $w_e^U = U_e - \varepsilon$.

THEOREM 3. *Given an uncertainty graph with trees T_L and T_U, any edge $e \in T_L \setminus T_U$ with $L_e \neq U_e$ is in every feasible query set for any feasible realization.*

Proof. Given an uncertainty graph, let h be an edge in $T_L \setminus T_U$ with non-trivial uncertainty interval. Assume all edges apart from h have been queried and thus have fixed weight w_e. As edge h is in T_L, we can choose its edge weight such that edge h is in any MST. We set $w_h = L_h + \varepsilon$ and choose ε so small, that all edges with at least the same weight in w^L now have a strictly larger edge weight. Symmetrically, if we choose the edge weight w_h sufficiently close to the upper limit U_h, no MST contains edge h. Consequently we cannot decide whether edge h is in an MST without querying it. ∎

Any edge in the set $T_L \setminus T_U$ with non-trivial uncertainty interval is in every feasible query set and thus can be queried in a preprocessing procedure before the start of the algorithm. This can be done by repeatedly computing T_L and T_U as MSTs of the realizations w^L and w^U and querying all edges in $T_L \setminus T_U$. After at most m repetitions this difference contains only edges with trivial uncertainty interval. The existence of edges e with $L_e < U_e$ in $T_L \setminus T_U$ increases the size of every feasible query set, in particular also the optimal query set, and hence decreases the competitive ratio of an instance. If we furthermore choose the same ordering for identical trivial edges for T_L and T_U, then there are no non-trivial edges in $T_L \setminus T_U$. Thus, when analyzing the worst-case competitive ratio of an algorithm, we can restrict to instances for which $T_L = T_U$.

ASSUMPTION 4. *We restrict to uncertainty graphs for which $T_L = T_U$.*

4. A New Algorithm Framework.

We design an algorithmic framework for *MST under uncertainty*, which allows to plug in several different algorithmic cores. It is the basis for both, our randomized algorithm and our algorithm for non-uniform query costs. The algorithm FRAMEWORK is an adaption of the deterministic algorithm for the problem presented in [7] in the sense that it also relies on the cycle characterization of MSTs: Every edge not in a particular minimum spanning tree is maximal in the cycle it closes when added to the MST.

Given an uncertainty graph $G = (V, E)$ our algorithm FRAMEWORK starts with a lower limit tree T_L. We can view this as a first candidate for an MST we want to verify. We consecutively try to add the other edges $f_1, \dots, f_{m-n+1} \in R := E \setminus T_L$ to it in order of increasing lower limit; in case of ties we prefer the edge with the smaller upper limit. In every iteration $i = 0, \ldots, m-n+1$ we maintain a minimum spanning tree verified for the already considered edge set $E_i := T_L \cup \{f_1, \ldots, f_i\}$; that is, we maintain a nested chain of subsets $\emptyset = Q_0 \subseteq Q_1 \subseteq \cdots \subseteq Q_{m-n+1}$ such that $Q_i \subseteq E_i$ is a feasible query set for E_i. When we try to add edge f_i to the current spanning tree in iteration i, we consider the cycle C_i it closes and query edges until we find a maximal edge on C_i. Once we find such an edge, we delete it, as there is an MST not containing this edge. Then we start a new iteration and take the next edge of the sequence into account. A formal description of this procedure is given further below in Algorithm 1.

This algorithmic structure allows us to prove two lemmas about any feasible query set and thus, in particular, the optimal feasible query set. The first lemma shows that any feasible query set for the entire uncertainty graph $G = (V, E)$ also verifies a minimum spanning tree for the subgraph $G_i = (V, E_i)$. This crucially relies on the fact that we add edges ordered by increasing lower limit.

Lemma 5. *Let $i \in \{0, \ldots, m-n+1\}$. Given a feasible query set Q for the uncertainty graph $G = (V, E)$, then the set $Q|_{E_i} := Q \cap E_i$ is a feasible query set for $G_i = (V, E_i)$.*

Proof. For some fixed realization of edge weights, let T be a minimum spanning tree of G certified by the feasible query set Q. We construct a minimum spanning tree T' of G_i by solely using information provided by the query set $Q|_{E_i}$.

We first argue that there is a minimum spanning tree of G_i that contains every edge in $T \cap E_i$: Consider an edge $e \in T \cap E_i$ and let $U \subset V$ be the subset of nodes in one of the two connected components obtained by deleting e from T. Since Q is a feasible query set, it certifies that e has minimal weight among all edges in E connecting U to its complement $V \setminus U$. As a consequence, query set $Q|_{E_i}$ certifies that e's weight is minimal among all edges in E_i connecting U and $V \setminus U$.

We proceed to deal with edges in $E_i \setminus T$ by distinguishing two cases. The first case is that adding edge $e \in E_i \setminus T$ to $T \cap E_i$ closes a cycle C. Then, adding edge e to tree T closes the same cycle C. Thus, as Q is a feasible query set, it certifies that edge e is maximal on C. Moreover, since $C \subseteq E_i$, query set $Q|_{E_i}$ obviously suffices as a certificate. We can therefore discard every such edge e.

The second case is, that adding edge $e \in E_i \setminus T$ to the spanning tree T closes a cycle C containing some edge $f \notin E_i$. The feasible query set Q certifies that e's weight is lower bounded by the weight of any edge on C including edge f. Notice that $L_e \leq L_f$ due to our ordering of edges by increasing lower limit (and $U_e \leq U_f$ in case that $L_e = L_f$). Thus, in order to certify that e's weight is lower bounded by f's weight, its exact weight w_e must be known.

Summarizing, the query set $Q|_{E_i}$ certifies that edges in $T \cap E_i$ can be included into T', certain edges can be safely discarded, and the exact weights of all remaining edges in E_i are known. The gathered information clearly suffices to find a minimum spanning tree T' and, as a consequence, $Q|_{E_i}$ is indeed a feasible query set for G_i. ☐

In the next lemma we give a precise characterization of the edges which a feasible query set contains. This characterization is similar to the so-called 'Witness Set Lemma', that is used in [7] for the deterministic algorithm.

Lemma 6. *For some realization of edge weights, let T be a verified MST of the graph $G_i = (V, E_i)$ and let C be the cycle closed by adding edge f_{i+1} to T. Furthermore, let h be some edge with the largest upper limit in C and $g \in C \backslash h$ be an edge* *with $U_g > L_h$. Then any feasible query set for $G_{i+1} = (V, E_i \cup \{f_{i+1}\})$ contains h or g. Moreover, if A_g is contained in A_h, any feasible query set contains edge h.*

Proof. Consider a minimum spanning tree T' for G_{i+1}. We distinguish two cases depending on edge h being in the tree T' or not. If $h \in T'$, any feasible query set must identify an edge of larger weight on the cycle C. Edge h has the maximal upper limit U_h among all edges in C and thus it must be queried for that purpose. Hence, in this case, edge h is in any feasible query set.

If edge h is not in the tree T', then h must have maximal weight in C. In particular, a query set must verify that h's weight is lower bounded by g's weight. The uncertainty intervals of these two edges overlap, and thus any feasible query set contains at least one of the two edges. Moreover, if g's uncertainty interval is contained in that of edge h, querying g does not reveal any information about the ordering of the two edge weights. Hence, h must be contained in any feasible query set in this case. ∎

The key to our framework is the structural insight resulting in Assumption 4, which we then use to apply Lemma 6 in the analysis of our algorithm. We show that any edge f in the algorithm, which is added to the current spanning tree to close a cycle, has the largest upper limit in this cycle. By Assumption 4, $T_L = T_U$ and thus no edge f_i is in T_U. This means, if the candidate MST does not change during the algorithm and stays T_L, each edge f_i has the largest upper limit in the cycle it closes with the tree. If the tree changes, an edge f_i replaces some edge $e \in T_L$ that is on the cycle C which f_i closes with the tree. Then, f_i's weight, and the weight or upper limit of all other edges C must be smaller than the upper limit of the deleted edge e. Thus, all following cycles closed by edges $f_j, j > i$, which contained the deleted edge e now contain other edges from C and the upper limits on the cycles never increase. Consequently, these edges f_j also have the largest upper limit in the cycle they close with the tree and we can apply Lemma 6 to them.

This means that any feasible query set contains either edge f_i or all edges with uncertainty interval overlapping that of edge f_i. Moreover, by Observation 1, if edge f_i is in the tree T_i (the MST we verified for G_i), then edge f_i must have been queried. Consequently, all edges that are not in the lower limit tree and, in a later iteration, occur as edge g in Lemma 6 have already been queried. We can thus restrict to consider those edges as g-edges that are in T_L. We call them *neighbors* of f_i and let the *neighbor set* $X(f_i)$ contain all edges $e \in C_i \cap T_L$ that have an overlapping uncertainty interval $U_e > L_{f_i}$.

COROLLARY 7. *Given an uncertainty graph G and a realization of edge weights, let T_L be its lower limit tree. Let T be a verified MST of the graph $G_i = (V, E_i)$ and let C be the cycle closed by adding edge f_{i+1} to T. Furthermore, let $X(f_{i+1}) \subseteq C \cap T_L$ be the neighbor set. Then any feasible query set contains f_{i+1} or $X(f_{i+1})$.*

Furthermore, Assumption 4 yields that after querying edge f_i or the neighbor set, the conditions for the second part of Lemma 6 are always fulfilled.

LEMMA 8. *Given a cycle C on which we have queried edge f with the largest lower limit or all its neighbors $X(f)$ and still no edge is known to be maximal. Then any edge $e \in C$ with largest upper limit on the cycle (which may now be different from edge f) is in any feasible query set.*

Proof. We distinguish two cases and show for both that we can apply Lemma 6: If edge f was queried but is still not known to be maximal, its edge weight lies in the uncertainty interval of e, as edge f has the largest lower limit. If all neighbors of f

were queried, we have $e = f$. This is because by Assumption 4 edge f has the largest upper limit on C. Furthermore, the edge weight of one of f's neighbors lies in A_f, as f is not known to be maximal. For both cases edge e is in any feasible query set by Lemma 6. ∎

Hence we can extend the framework by the following two steps on a cycle C_i without an edge that is known to be maximal. First we call an algorithm Core which somehow decides between querying edge f_i and its neighbor set $X(f_i)$. If this query does not identify a maximal edge, we continue querying edges in the cycle in order of decreasing upper limit. A formal description of our algorithm is given in Algorithm 1.

Algorithm 1 Framework

```
Input: An uncertainty graph $G = (V, E)$.
Output: A feasible query set $Q$.
 1: Determine a tree $T_L$ and set the temporary graph $\Gamma$ to $T_L$.
 2: Index the edges in $R := E\backslash T_L$ by increasing lower limit $f_1, \dots, f_{m-n+1}$.
 3: Initialize $Q = \emptyset$.
 4: for $i = 1$ to $m - n + 1$ do
 5:     Add edge $f_i$ to the temporary graph $\Gamma$ and let $C_i$ be the unique cycle closed.
 6:     Let the neighbor set $X(f_i)$ be the set of edges $g \in T_L \cap C_i$ with $U_g > L_{f_i}$.
 7:     if $X(f_i)$ is not empty then
 8:         use algorithm Core to decide between querying $f_i$ and $X(f_i)$.
 9:     while no edge in the cycle $C_i$ is known to be maximal do
10:         Query the unqueried edge $e \in C_i \setminus Q$ with maximum $U_e$ and add it to the
            query set $Q$.
11:     Delete a maximal edge from $\Gamma$.
12: return The query set $Q$.
```

As pointed out above, the algorithm maintains a verified MST for a subset of the edges of increasing size. At the end of the algorithm the tree is verified for the complete edge set E and thus Q is a feasible query set. The Framework terminates, as in each iteration of the while loop an edge is queried. As soon as all edges on a cycle have been queried, we have certainly identified a maximal edge.

Any edge that is queried within Framework outside algorithm Core is in any feasible query set by Lemma 8. Thus the competitive ratio of an algorithm is solely determined by the query strategy of algorithm Core.

Relation to Vertex Cover. It was already observed by Erlebach and Hoffmann [6] that *MST under uncertainty* has a close relation to the vertex cover problem. They show that for a fixed realization we can design a bipartite vertex cover graph using the relation of Lemma 6. Then any feasible query set contains a vertex cover of this graph. We generalize the use of this relation to a complete problem instance. We create a bipartite vertex cover graph online along the execution of the Framework and thus prove a connection to the online bipartite vertex cover problem.

In the online bipartite vertex cover problem one side of the bipartite graph is given (consisting of the so-called *offline vertices*) and the vertices of the other side appear online one by one together with their incident edges. In any iteration we have to maintain a feasible vertex cover of the revealed graph.

For an instance of *MST under uncertainty* we generate the graph as follows: All edges of the lower limit tree T_L form the offline vertices of the vertex cover graph.

During an execution of the algorithm FRAMEWORK we add the edge $f_i \in R$ to the temporary graph Γ such that it closes a unique cycle C_i. Upon adding edge f_i in the algorithm, we add a corresponding vertex to the vertex cover graph and connect this new vertex to all vertices corresponding to edges in $C_i \cap T_L$ with overlapping uncertainty interval. Thus the set of neighbors of the new vertex corresponds to the neighbor set $X(f_i)$.

Observe that the vertex cover graph we create depends on the realization of the edge weights. We determine the maximal edge for every cycle C_i and delete it. This determines which cycle is closed next and thus the next incidences in the vertex cover graph. Thus we need to create the vertex cover graph online and cannot do it a priori.

5. Randomized Algorithm.

In this section we describe a randomized algorithm for *MST under uncertainty* that achieves competitive ratio $1 + 1/\sqrt{2} \approx 1.707$. Our algorithm RANDOM employs the algorithm FRAMEWORK presented in the previous section and makes use of its vertex cover interpretation for the algorithm core. We decide how to resolve cycles, by maintaining an edge potential for each edge $e \in T_L$ describing the probability to query it. The edge potentials are increased in every cycle we consider throughout the algorithm. To determine the increase, we carefully adapt a water-filling scheme presented in [18] for online bipartite vertex cover. This scheme considers all edges queried in the CORE algorithm, but not those queried in the FRAMEWORK. This is the reason that our algorithm does not achieve the same competitivity ratio as for online bipartite vertex cover. In this section we assume uniform query cost $c_e = 1$, $e \in E$, and explain the generalization to non-uniform query costs in Section 6.

Our algorithm CORE OF RANDOM is the decision procedure which edges to query on a cycle C_i in the FRAMEWORK. We maintain an edge potential $y_e \in [0,1]$ for all edges $e \in T_L$ which is initially set to 0. We query an edge if its potential exceeds the query bound b, which we draw uniformly at random from $[0,1]$ before we start the algorithm FRAMEWORK. Thus we can interpret the potential as the probability that edge e is queried.

We identify the following goals for the algorithm design: First, edges in the neighbor set of f_i should be queried with high probability, as they can occur in further neighbor sets later. Second, if an edge e in the neighbor set is queried with probability y_e, edge f_i must be queried at least with probability $1 - y_e$ to ensure feasibility. And third, in expectation we cannot query more than $1 + \alpha$ edges per iteration to achieve competitive ratio $1 + \alpha$. Here α is a fixed parameter that is determined later in the analysis. Formally we achieve these goals by distributing no more than potential α among the neighbor set $X(f_i)$. We distribute the potential among all neighbors such that they reach an equal level $t(f_i) \in [0,1]$ which is as large as possible. This means when we increase y_e to $\max\{t(f_i), y_e\}$ for all neighbors $e \in X(f_i)$, the total potential increase sums up to at most α. Now we compare this threshold $t(f_i)$ to the query bound b to decide which edges to query. If b is the larger of the two, we query edge f_i and otherwise we query all neighbors, the edges in $X(f_i)$.

The join of the two algorithms FRAMEWORK and CORE OF RANDOM together with the preceding random choice of b and initially setting $y_e := 0$, $e \in T_L$, forms the algorithm RANDOM. This algorithm has competitive ratio $1 + 1/\sqrt{2} \approx 1.707$ for *MST under uncertainty*, if we choose the parameter α to be $1/\sqrt{2}$.

For the proof of this performance we use an amortized analysis over all cycles closed during the run of the algorithm. We consider a fixed realization of edge weights and a corresponding optimal query set Q^*. We denote the potential of an edge $e \in T_L$

Algorithm 2 CORE OF RANDOM

```
Input: A cycle $C_i$ of the algorithm FRAMEWORK with its edge $f_i$, neighbor set $X(f_i)$,
    as well as the edge potentials $y_e = y_e^i$ and the query bound $b$.
Output: A feasible query set $Q \subseteq C_i$.
1: Maximize the threshold $t(f_i) \leq 1$ s.t. $\sum_{e \in X(f_i)} \max\{0, t(f_i) - y_e\} \leq \alpha$.
2: Increase edge potentials $y_e := \max\{t(f_i), y_e\}$ for all edges $e \in X(f_i)$.
3: if $t(f_i) < b$ then
4:     Add edge $f_i$ to the query set $Q$ and query it.
5: else
6:     Add all edges in $X(f_i)$ to the query set $Q$ and query them.
7: return The query set $Q$.
```

at the start of iteration i by y_e^i and use y_e to denote the edge potential after the last iteration of the algorithm. We will relate the expected number of queries of RANDOM to the total edge potential we distribute. For this, we first bound the potential distributed to edges in $T_L \setminus Q^*$ by the number of edges in $R \cap Q^*$ times our parameter α (where $R = E \setminus T_L$).

LEMMA 9. *Given an instance of* MST under uncertainty *together with a realization of edge weights, the edge potentials after an execution of* RANDOM*, and any feasible query set Q^*, it holds that*

$$\sum_{e \in T_L \setminus Q^*} y_e \leq \alpha \cdot |R \cap Q^*|.$$

Proof. For any edge $e \in T_L \setminus Q^*$, Corollary 7 states that all neighboring edges $f \in R$ with $e \in X(f)$ must be in the optimal query set Q^*. The potential y_e is the sum of the potential increases caused by edges $f \in R$ with $e \in X(f)$. As in each iteration of the algorithm the total increase of potential is bounded by α, we have

$$\begin{aligned}\sum_{e \in T_L \setminus Q^*} y_e &= \sum_{e \in T_L \setminus Q^*} \sum_{\substack{i: f_i \in Q^*,\\ e \in X(f_i)}} \max\left\{t(f_i) - y_e^i, 0\right\} \\ &\leq \sum_{i: f_i \in R \cap Q^*} \sum_{e \in X(f_i)} \max\left\{t(f_i) - y_e^i, 0\right\} \\ &\leq \sum_{i: f_i \in R \cap Q^*} \alpha = \alpha \cdot |R \cap Q^*|.\end{aligned}$$

This concludes the proof. ∎

Similarly, we can bound the sum over $1 - t(f_i)$ of all edges $f_i \in R \setminus Q^*$. We will see in the proof of the competitive ratio that $1 - t(f_i)$ is the probability for an edge $f_i \in R \setminus Q^*$ to be queried in RANDOM.

LEMMA 10. *Given an instance of* MST under uncertainty *together with a realization of edge weights, thresholds $t(f_i)$ determined in* CORE OF RANDOM*, and any feasible query set Q^*, it holds that*

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

Proof. For an edge $f_i \in R\backslash Q^*$ with $t(f_i) < 1$ we distribute exactly potential α among its neighbors $X(f_i)$ in Lines 1 and 2 of the Algorithm Core of Random. By Corollary 7, $X(f_i)$ is part of the optimal query set Q^*. We consider the share of the total potential increase each neighbor receives and distribute the term $1 - t(f_i)$ according to these shares. Hence,

$$\begin{aligned}\sum_{i: f_i \in R\backslash Q^*} (1 - t(f_i)) &= \sum_{i: f_i \in R\backslash Q^*} \frac{1 - t(f_i)}{\alpha} \sum_{e \in X(f_i)} \max\{t(f_i) - y_e^i, 0\} \\ &= \sum_{e \in T_L \cap Q^*} \sum_{\substack{i: f_i \in R\backslash Q^*,\\ e \in X(f_i)}} \frac{1 - t(f_i)}{\alpha} \left(y_e^{i+1} - y_e^i\right). \end{aligned} \tag{1}$$

In the last equation we have used $y_e^{i+1} = \max\{t(f_i), y_e^i\}$. We consider the inner sum in (1) and bound the summand from above by an integral from y_e^i to y_e^{i+1} of the function $\frac{1-z}{\alpha}$. This yields a valid upper bound, as the function is decreasing in z and $t(f_i) = y_e^{i+1}$, unless $y_e^{i+1} - y_e^i = 0$. Hence,

$$\sum_{\substack{i: f_i \in R\backslash Q^*,\\ e \in X(f_i)}} \frac{1 - t(f_i)}{\alpha} \left(y_e^{i+1} - y_e^i\right) \leq \sum_{\substack{i: f_i \in R\backslash Q^*,\\ e \in X(f_i)}} \int_{y_e^i}^{y_e^{i+1}} \frac{1-z}{\alpha} \mathrm{d}z \leq \int_0^1 \frac{1-z}{\alpha} \mathrm{d}z = \frac{1}{2\alpha}.$$

Now we use this bound in Equation (1) and conclude

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

This conclude the proof. ∎

Using these two bounds we can calculate the competitive ratio of the Algorithm Random.

Theorem 11. *For $\alpha = \frac{1}{\sqrt{2}}$, Random has competitive ratio $1 + \frac{1}{\sqrt{2}} (\approx 1.707)$.*

Proof. Consider a fixed realization and an optimal query set Q^*, as before. We first note that by Lemma 8 all edges queried in the Algorithm Framework are in Q^*. Now we observe that the increase of potentials in the algorithm depends on the cycles that are closed and thus on the realization, but not on the queried edges. In particular, the edge potentials are chosen independently of the query bound b in the algorithm. Therefore an edge $e \in T_L\backslash Q^*$ is queried with probability $P(y_e \geq b) = y_e$ and an edge $f_i \in R\backslash Q^*$ is queried with probability $P\big(t(f_i) < b\big) = 1 - t(f_i)$. Hence, we can bound the total expected query cost by

$$\mathbb{E}\left[|Q|\right] \leq |Q^*| + \sum_{e \in T_L\backslash Q^*} y_e + \sum_{i: f_i \in R\backslash Q^*} (1 - t(f_i)).$$

Applying Lemmas 9 and 10 to this equation yields total expected query cost

$$\mathbb{E}\left[|Q|\right] \leq |Q^*| + \alpha \cdot |R \cap Q^*| + \frac{1}{2\alpha} \cdot |T_L \cap Q^*|.$$

Choosing $\alpha = 1/\sqrt{2}$ yields the desired competitive ratio $1 + 1/\sqrt{2}$ for Random.

We consider the introductory example described in Figure 1 for realization $\mathcal{R}_2$, to show that this analysis is tight. Random distributes potential α to edge g and thus

Algorithm 3 CORE OF NON-UNIFORM RANDOM

```
Input: A cycle $C_i$ of the algorithm FRAMEWORK with its edge $f_i$, neighbor set $X(f_i)$,
    as well as the edge potentials $y_e$ and the query bound $b$.
Output: A feasible query set $Q \subseteq C_i$ and a maximal edge.
1: Maximize the threshold $t(f_i) \leq 1$ s.t. $\sum_{e \in X(f_i)} c_e \cdot \max\{0, t(f_i) - y_e\} \leq \alpha \cdot c_{f_i}$.
2: Increase edge potentials $y_e := \max\{t(f_i), y_e\}$ for all $e \in X(f_i)$.
3: if $t(f_i) < b$ then
4:     Add edge $f_i$ to the query set $Q$ and query it.
5: else
6:     Add all edges in $X(f_i)$ to the query set $Q$ and query them.
7: return The query set $Q$.
```

queries g first with probability α and f first with probability $1-\alpha$. As the realization has the structure $L_f < w_g < U_g \leq w_f$ we need two queries if we query edge g first and one query otherwise. Thus the expected number of queries is $2\alpha + 1 - \alpha$, which is $1+\alpha$. The optimal query set has size 1, hence RANDOM has expected competitive ratio $1+\alpha$ for this instance. ◻

6. Non-uniform Query Cost.

We now turn to the problem *MST under uncertainty* in which each edge $e \in E$ has associated an individual query cost c_e. W.l.o.g. we assume $c_e > 0$, for all $e \in E$, since querying all other edges does not increase the total query cost. We adapt our algorithm RANDOM (Sec. 5) to handle non-uniform query costs achieving the same competitive ratio $1 + 1/\sqrt{2}$ and then show how to derive a deterministic 2-competitive algorithm from it.

Before showing the main results, we remark that the problem can also be transformed into the OP-OP model [11]. This model allows multiple queries per edge and each query returns an open or trivial subinterval (point). Given an uncertainty graph, we model the non-uniform query cost $c_e \in \mathbb{Z}_{>0}$, $e \in E$, in the OP-OP model as follows: querying an edge e returns the same interval for $c_e - 1$ queries and returns the exact edge weight upon the c_e-th query. Then the 2-competitive algorithm for the OP-OP model [11] has a running time depending on the query cost of our original problem.

THEOREM 12. *There is a pseudo-polynomial, deterministic, 2-competitive algorithm for* MST under uncertainty *with non-uniform query cost.*

6.1. Randomization for Non-uniform Query Costs.

We generalize the Algorithm CORE OF RANDOM (Sec. 5) to the non-uniform query costs model. The adaptation is similar to one for the weighted online bipartite vertex cover problem in [18]. For each edge $f_i \in E \setminus T_L$ with query cost c_{f_i} we now distribute at most $\alpha \cdot c_{f_i}$ new potential to its neighborhood $X(f_i)$. We obtain Algorithm 3, CORE OF NON-UNIFORM RANDOM, by replacing Line 1 of CORE OF RANDOM (Algorithm 2) by:

$$\text{maximize } t(f_i) \leq 1 \text{ s.t. } \sum_{e \in X(f_i)} c_e \cdot \max\{t(f_i) - y_e, 0\} \leq \alpha \cdot c_{f_i} \text{ holds.} \tag{2}$$

We can apply exactly the same analysis as presented in Section 5 to prove the competitive ratio of this algorithm. There are non-uniform cost variants of the two lemmas bounding the potential of the edges in T_L and the query probability of edges in R.

LEMMA 13. *Given an instance of* MST under uncertainty *together with a realization of edge weights, the edge potentials after an execution of* RANDOM *adapted* *by* (2)*, and any feasible query set* Q^**, it holds that*

$$\sum_{e \in T_L \setminus Q^*} c_e \cdot y_e \le \alpha \sum_{i: f_i \in R \cap Q^*} c_{f_i}.$$

LEMMA 14. *Given an instance of* MST under uncertainty *together with a realization of edge weights, the thresholds* $t(f_i)$ *determined according to* (2)*, and any feasible query set* Q^**, it holds that*

$$\sum_{i: f_i \in R \setminus Q^*} c_{f_i} \cdot \big(1 - t(f_i)\big) \le \frac{1}{2\alpha} \sum_{e \in T_L \cap Q^*} c_e.$$

Using the same line of arguments as in the proof of Theorem 11, we can derive the following theorem.

THEOREM 15. *For the non-uniform query cost setting our algorithm* RANDOM *adapted according to* (2) *achieves competitive ratio* $1 + \frac{1}{\sqrt{2}}$.

6.2. Balancing Algorithm. Our polynomial-time algorithm BALANCE applies the algorithm FRAMEWORK together with an adaption of the previously described algorithm CORE OF NON-UNIFORM RANDOM to the deterministic setting. We call this new core algorithm CORE OF BALANCE. Erlebach et al. [7] prove that no deterministic algorithm can achieve competitive ratio less than 2, even in the uniform cost case. Thus we set the parameter α to 1. The goals for the algorithm design are the same as before. We prefer to query the neighbor set of f_i, as these edges may appear in several neighbor sets. However, we cannot query the neighbor set, if the additional cost exceeds c_{f_i} to ensure the competitive ratio.

As before we achieve these goals by maintaining an edge potential y_e for each edge $e \in T_L$. We reinterpret it as representing the share of the query cost of edge e for which we have already accounted. As the optimal solution needs to contain either edge f_i or all edges in $X(f_i)$, its cost increases exactly by the smaller of the two costs. We query edge f_i, if its query cost is smaller than the not yet covered cost of the neighbors. This is equivalent to a threshold $t(f_i) < 1$. In this case edge f_i covers an additional cost share of size c_{f_i} in the neighbor set and we increase the edge potentials accordingly. Otherwise all neighbors $e \in X(f_i)$ are queried.

Algorithm 4 CORE OF BALANCE

```
Input: A cycle $C_i$ of the algorithm FRAMEWORK with its edge $f_i$, neighbor set $X(f_i)$
    as well as the edge potentials $y_e$.
Output: A feasible query set $Q \subseteq C_i$.
1: Maximize the threshold $t(f_i) \le 1$ s.t. $\sum_{e \in X(f_i)} c_e \cdot \max\{0, t(f_i) - y_e\} \le c_{f_i}$.
2: Increase edge potentials $y_e := \max\{t(f_i), y_e\}$ for all $e \in X(f_i)$.
3: if $t(f_i) < 1$ then
4:     Add edge $f_i$ to the query set $Q$ and query it.
5: else
6:     Add all edges in $X(f_i)$ to the query set $Q$ and query them.
7: return The query set $Q$.
```

Similar to the proof of the competitive ratio of Algorithm RANDOM we can divide the algorithm's query set into different parts and bound them separately to prove that Algorithm BALANCE is 2-competitive.

THEOREM 16. *Algorithm* BALANCE *has competitive ratio* 2*, which is best-possible.*

Proof. For some realization, let Q^* denote an optimal query set. Consider the query set Q computed by BALANCE and let $R := E \setminus T_L$. Then we can split the query set Q into three parts $Q \cap Q^*$, $(T_L \cap Q) \setminus Q^*$ and $(R \cap Q) \setminus Q^*$. For all edges $e \in T_L \cap Q$ we have $y_e = 1$, hence,

$$\begin{aligned}\sum_{e\in Q} c_e &= \sum_{e\in Q\cap Q^*} c_e + \sum_{e\in(T_L\cap Q)\setminus Q^*} c_e + \sum_{i:f_i\in(R\cap Q)\setminus Q^*} c_{f_i}\\ &\leq \sum_{e\in Q^*} c_e + \sum_{e\in T_L\setminus Q^*} c_e \cdot y_e + \sum_{i:f_i\in R\setminus Q^*} c_{f_i}\,.\end{aligned}$$

The first term can be trivially bounded by the cost of Q^*. For the edges in $R \setminus Q^*$, we charge their full query cost in terms of potential to the edges in the neighbor set. We denote the edge potential at the start of iteration i by y_e^i and denote the edge potential after the last iteration of the algorithm by y_e. By Corollary 7 we know that $X(f_i) \subseteq Q^*$ for $f_i \notin Q^*$. Thus we can reformulate

$$\sum_{i:f_i\in R\setminus Q^*} c_{f_i} = \sum_{i:f_i\in R\setminus Q^*}\ \sum_{e\in X(f_i)} c_e\left(y_e^{i+1} - y_e^i\right) \leq \sum_{e\in T_L\cap Q^*} c_e \cdot y_e \leq \sum_{e\in T_L\cap Q^*} c_e\,.$$

For all edges in $T_L \setminus Q^*$, we apply Lemma 13 with $\alpha = 1$. Thus we get in total

$$\begin{aligned}\sum_{e\in Q} c_e &\leq \sum_{e\in Q^*} c_e + \sum_{e\in T_L\setminus Q^*} c_e \cdot y_e + \sum_{i:f_i\in R\setminus Q^*} c_{f_i}\\ &\leq \sum_{e\in Q^*} c_e + \sum_{i:f_i\in R\cap Q^*} c_{f_i} + \sum_{e\in T_L\cap Q^*} c_e\\ &= 2\sum_{e\in Q^*} c_e\,.\end{aligned}$$

This factor of 2 is best possible for deterministic algorithms, even in the special case of uniform query costs (cf. Section 3, [7]). ∎

7. Computing the MST Weight under Uncertainty. In this section we give an optimal polynomial-time algorithm for computing the exact MST weight in an uncertainty graph. As a key to our result, we algorithmically utilize the well-known characterization of MSTs through the *cut property* – in contrast to previous algorithms for *MST under uncertainty* which relied on the *cycle property* (cf. RANDOM, BALANCE, and U-RED [7]).

In our Algorithm CUT-WEIGHT, we consider a spanning tree Γ and iteratively delete its edges. In each iteration, we consider the cut which is defined by the two halves of the tree and query edges in increasing order of lower limits until we have identified and queried a *minimal* edge in the cut. That means an edge which is in an MST for any feasible realization. Then we exchange the tree edge with the minimal edge.

THEOREM 17. *The algorithm* CUT-WEIGHT *finds the optimal query set for* MST weight under uncertainty *in polynomial-time.*

Proof. We show for every edge we query, that it is in any feasible query set. Assume there is an edge g which contradicts this. Then, let T be the MST which does not contain this edge. We query edge g in the algorithm, when it has the smallest

Algorithm 5 CUT-WEIGHT

```
Input: Uncertainty graph $G = (V, E)$.
Output: A feasible query set $Q$.
 1: Find a spanning tree $\Gamma$ and let $Q := \emptyset$.
 2: Index the edges of $\Gamma$ by $e_1, e_2, \ldots, e_{n-1}$.
 3: for $i = 1$ to $n - 1$ do
 4:   Delete $e_i$ from $\Gamma$.
 5:   Let $S$ be the cut containing all edges in $G$ between the two components of $\Gamma$.
 6:   while $S$ does not contain a minimal edge with trivial uncertainty interval do
 7:     Choose $g \in S$ such that $L_g = \min\{L_e | e \in S\}$.
 8:     Query $g$ and add it to $Q$.
 9:   Add a minimal edge in $S$ to $\Gamma$.
10: return The query set $Q$.
```

lower limit in a cut S. At least one edge $f \in S$ is in the MST T and $T \setminus f \cup g$ is also a spanning tree. As the cut S does not contain a minimal edge when g is chosen in CUT-WEIGHT, edge f has current upper limit $U'_f > L_g$. As we also have $L_g \leq L_f$, this means if the edge weight of g is sufficiently close to its lower limit, we can exchange g with edge f and reduce the weight of the tree T. Thus edge g must be in the feasible query set to ensure the spanning tree is minimal.

The query set the algorithm computes is feasible, as it verifies any edge that is chosen for the MST is minimal in a cut. The algorithm queries all edges of the MST, as any edge finally in the tree was a minimal edge with trivial uncertainty interval for some cut in the algorithm. It terminates, because in each iteration of the while loop one edge is queried. At the latest, when all edges in a cut have been queried, we find a minimal edge. It runs in polynomial time, as we query one edge in each iteration and there is a polynomial number of edges. ∎

It may seem surprising that the cut-based algorithm solves the problem optimally, whereas cycle-based algorithms do not. However, there is an intuitive explanation. The cycle-based algorithms identify the edge of *maximum weight* on a cycle, which is not in the tree. Informally speaking, they have a bias to query edges not in the MST. In contrast, CUT-WEIGHT considers cuts in the graph and identifies the *minimum weight* edge in each cut, which characterizes an MST.

8. Matroid Base under Uncertainty.

We consider a natural generalization of *MST under uncertainty*: given an *uncertainty matroid*, i.e., a matroid with a ground set of elements with unknown weights, find a minimum weight matroid base. Erlebach et al. [6] show that the algorithm U-RED [7] can be applied to uncertainty matroids with uniform query cost and yields again a competitive ratio of 2. Similarly, our algorithms RANDOM and BALANCE can be generalized to matroids with non-uniform cost, and CUT-WEIGHT can determine the total weight of a minimum weight matroid base.

THEOREM 18. *There are deterministic and randomized online algorithms with competitive ratio* 2 *and* $1 + 1/\sqrt{2} \approx 1.707$*, respectively, for* Matroid base under uncertainty *with non-uniform query cost.*

THEOREM 19. *There is an algorithm that determines an optimal query set for* Matroid base weight under uncertainty *and computes the exact weight of the base.*

In a matroid with known weights we can find a minimum weight base using greedy algorithms; we distinguish between *best-in* greedy and *worst-out* greedy algorithms (cf. [14]). They are dual in the sense that both solve the problem on a matroid and each takes the role of the other on the corresponding dual matroid.

The best-in greedy algorithm adds elements in increasing order of weights as long as the system stays independent. We present a best-in greedy algorithm, Cycle-Alg, for uncertainty matroids by merging ideas from the Algorithms Random and U-RED2 in [6]. The worst-out greedy algorithm deletes elements in decreasing order of weights as long as a basis is contained in what remains. We show how to adapt our Algorithm Cut-Weight in Section 7 to a worst-out greedy algorithm, Cut-Alg, for uncertainty matroids.

Proposition 20. *The Algorithms* Cycle-Alg *and* Cut-Alg *are dual to each other in the sense that they solve the same problem on a matroid and its dual.*

8.1. Cycle Algorithm. Our Algorithm Cycle-Alg is inspired by our Algorithm Framework as well as the algorithm for uncertainty matroids in [6]. To design a greedy algorithm, we avoid the preprocessing step of the framework and thus do not rely on Assumption 4. We start with a matroid basis and greedily decide for all other elements, if they improve the basis weight or not. Analogous to the MST case we define a lower limit matroid base B_L as a basis for the realization w^L, in which all weights of elements with non-trivial uncertainty interval are close to their lower limit, more precisely $w_x = L_x + \varepsilon$ for infinitesimally small $\varepsilon > 0$.

Given an uncertainty matroid $M = (X, \mathcal{I})$, our Algorithm Cycle-Alg starts with a minimal lower limit basis B_L. We can view this as a first candidate for a minimum weight basis we want to verify. We consecutively add the other elements $f_1, \dots, f_{m-n+1}$ to it in order of increasing lower limit; in the case of ties we prefer the element with the smaller upper limit. In every iteration we maintain a minimum weight basis verified for the already considered element set $X_i := B_L \cup \{f_1, \dots, f_i\}$ with corresponding family of independent sets $\mathcal{I}_i := \{I \cap X_i | I \in \mathcal{I}\}$, i.e., the matroid $M_i := (X_i, \mathcal{I}_i)$. For each element we add, we consider the minimal dependent set C that is now contained. We query elements from C until we identify a maximal element in this set, by each time choosing an element with maximal upper limit f from C and an element $g \in C \backslash f$ with overlapping uncertainty interval. Note that here element f is not necessarily the just added element f_i in the first iteration of the while loop, as our uncertainty matroid may not fulfill the equivalent of Assumption 4 for matroids.

Algorithm 6 Cycle-Alg

```
Input: An uncertainty matroid $M = (X, \mathcal{I})$.
Output: A feasible query set $Q$.
 1: Determine lower limit basis $B_L$; set the temporary basis $\Gamma$ to $B_L$.
 2: Index all elements in $R := X \setminus B_L$ by increasing lower limit $f_1, f_2, \dots, f_{m-n+1}$.
 3: Initialize $Q := \emptyset$.
 4: for $i = 1$ to $m - n + 1$ do
 5:     Add element $f_i$ to the temporary basis $\Gamma$ and let $C$ be the occurring minimal
        dependent set.
 6:     while $C$ does not contain a maximal element do
 7:         Choose $f \in C$ s.t. $U_f = \max\{U_e | e \in C\}$.
 8:         Choose $g \in C \backslash \{f\}$ with $U_e > L_f$.
 9:         Add elements $f$ and $g$ to the query set $Q$ and query them.
10:     Delete the maximal element $x$ from $\Gamma$.
11: return The query set $Q$.
```

The query set the algorithm computes is feasible, as it verifies any element that is deleted is maximal in a dependent set. It terminates, as in each iteration of the while loop at least one element is queried. When all elements in a set C have been queried, we always find a maximal element.

Observation 21. Cycle-Alg *is a best-in greedy algorithm for* Matroid base under uncertainty*.*

The structure of Cycle-Alg is very similar to our Algorithm Framework. In particular, we once again maintain a partial solution, i.e., a minimum weight basis verified for a subset of the elements, and extend it by an additional element in every iteration. Hence it is not surprising, that we can reprove Lemmas 5 and 6 for the uncertainty matroid setting.

Lemma 22. *Given a feasible query set Q for an uncertainty matroid $M = (X, \mathcal{I})$, then the set $Q|_{X_i} := Q \cap X_i$ is a feasible query set for $M_i = (X_i, \mathcal{I}_i)$.*

Lemma 23. *Let B be a verified minimum weight basis of the uncertainty matroid $M_i = (X_i, \mathcal{I}_i)$ and let C be the minimal dependent set contained in $B \cup f_{i+1}$. Furthermore, let h be an element with the largest upper limit in C and $g \in C \backslash h$ be an element with $U_g > L_h$. Then any query set verifying a minimum weight basis for $M_{i+1} = (X_{i+1}, \mathcal{I}_{i+1})$ contains h or g.*

In particular, if A_g is contained in A_h, any feasible query set contains element h.

Theorem 24. Cycle-Alg *is 2-competitive for* Matroid base under uncertainty*.*

Proof. The query set Q Cycle-Alg computes is built iteratively. In each step we consider an element pair f, g and query the previously not queried part of it. This means we can partition Q into subsets of size at most two and allocate an algorithm iteration to each of them. By Lemma 22, any feasible query set must verify a minimum weight basis in that iteration. Furthermore, Lemma 23 yields that any feasible query set contains at least one element from the allocated query subset. Using the fact that any query subset has size at most two, this yields that Q is at most twice as large as any feasible query set. ∎

8.2. Cut Algorithm. Our dual Algorithm Cut-Alg is a modification of the Algorithm Cut-Weight we present in Section 7. We choose a particular upper limit basis B_U to start the algorithm. Let B_U be a minimal basis for the realization w^U, in which all weights of elements with non-trivial uncertainty interval are close to their upper limit, more precisely $w_x = U_x - \varepsilon$ for infinitesimally small $\varepsilon > 0$. We can view this as a first candidate for a minimum weight basis we want to verify. We choose to delete the basis elements $g_1, \dots, g_n$ from it in order of decreasing upper limit; in the case of ties we prefer the element with the larger lower limit. For each element we delete, we consider the set $S \subseteq X$ of all elements that would complete a basis. We query elements from S until we identify a minimal element in this set. We decide which elements to query by choosing an element with smallest lower limit g from S and an element $f \in S \backslash g$ with overlapping uncertainty interval. As before an element is called *minimal*, if it is in a basis for any realization.

The query set computed by the algorithm is feasible as it verifies any element in Γ is minimal in a set S and at least one element from the set S is contained in every basis. It terminates as in each iteration of the while loop an element is added or queried. When all elements in a set S have been queried, we always find a minimal element.

Algorithm 7 CUT-ALG

```
Input: An uncertainty matroid $M = (X, \mathcal{I})$.
Output: A feasible query set $Q$.
 1: Determine an upper limit basis $B_U$ and set the temporary basis $\Gamma$ to $B_U$.
 2: Index all elements of $B_U$ by decreasing upper limit $g_1, g_2, \dots, g_n$.
 3: Initialize $Q := \emptyset$.
 4: for $i = 1$ to $n$ do
 5:    Delete element $g_i$ from $\Gamma$.
 6:    Let $S \subset X$ contain all elements $x$ such that $\Gamma \cup \{x\}$ contains a basis.
 7:    while $S$ does not contain a minimal element do
 8:       Choose $g \in S$ s.t. $L_g = \min\{L_e | e \in S\}$.
 9:       Choose $f \in S \backslash \{g\}$ with $L_f < U_g$.
10:       Add elements $f$ and $g$ to the query set $Q$ and query them.
11:    Add a minimal element of $S$ to $\Gamma$.
12: return The query set $Q$.
```

OBSERVATION 25. CUT-ALG *is a worst-out greedy algorithm for* Matroid base under uncertainty.

We claim that CUT-ALG is dual to our algorithm CYCLE-ALG in the sense that it behaves exactly as CYCLE-ALG does on the dual matroid. For a given uncertainty matroid M, the dual matroid M^* has the same element set and the set of independent sets contains all sets whose complement contains a basis. Thus a basis of the dual matroid is exactly the complement of a basis of the original matroid.

We will consider the dual matroid with the *inverted weight function.* With this notion we mean for any element $x \in X$ with weight w_x and uncertainty interval (L_x, U_x) we consider the uncertainty interval $(-U_x, -L_x)$ and weight $-w_x$ for the dual matroid.

We first prove that CUT-ALG computes a query set verifying a minimum weight basis of the dual matroid for the inverted weight function.

THEOREM 26. CUT-ALG *computes a* 2*-competitive query set* Q *verifying a minimum weight matroid base for the dual matroid* $M^* = (X, \mathcal{I}^*)$ *with inverted weight function.*

Proof. CUT-ALG starts out with an upper limit basis $X \setminus B_U$ of the matroid M. According to the inverted weight function, this is a lower limit basis of M^*. In the algorithm we sort the elements of B_U by decreasing upper limit. This is the same order as sorting by increasing lower limit for the inverted weight function. The set S we choose is a minimal dependent set, i.e., a cycle, in the dual matroid M^*. Exactly as required in Lemma 23, we choose the two elements we query such that one has the largest upper limit, i.e., the smallest lower limit according to the inverted weight function, and the other has an overlapping uncertainty interval. Thus, for any edge pair we add to the query set Q, the optimal query set contains at least one of the two.

Therefore CUT-ALG computes a query set Q that verifies a minimum weight basis of M^* and has at most twice the size of any query set verifying such a basis. ∎

THEOREM 27. CUT-ALG *is* 2*-competitive for* Matroid base under uncertainty.

Proof. We need to prove that the query set Q computed by the Algorithm CUT-ALG verifies a minimum weight matroid base and has at most twice the size of any feasible query set fulfilling this property. First we observe that CUT-ALG verifies a minimum weight matroid basis of the dual matroid M^* with inverted weight function.

The complement of a basis of the dual matroid is a basis of the original matroid M. Hence, the algorithm verifies a basis of the matroid M of maximum weight for the inverted weight function. This however, means it verifies a basis of M of minimum weight according to the original weight function.

The line of arguments above shows that any set verifying a minimum weight matroid base of M^* for the inverted weight function also verifies a minimum weight matroid base of M for the original weight function and vice versa. Hence, the family of feasible query sets is the same for both problems. As the computed query set Q is at most twice the size of a feasible query set for a minimum weight matroid base of M^*, it also has at most twice the size of a feasible query set verifying a minimum weight matroid base of M with inverted weight function. ∎

9. Queries Returning Intervals.

We consider the extension of our query model, in which a query to an edge may return an open subinterval of the current uncertainty interval instead of a point. In this model, several queries to one edge might be necessary. The model was first analyzed in [11] under the name OP-OP model, meaning the original uncertainty intervals are open intervals or points and the query output as well. They show for *MST under uncertainty* that the deterministic 2-competitive algorithm by Erlebach et al. [7] extends to the OP-OP model without any loss in the competitive ratio. We analyze the OP-OP model for randomized algorithms and show the surprising fact, that no improvement over competitive ratio 2 is possible using randomization.

We consider a randomized problem instance $(G, \mathcal{R}, p)$, that is an uncertainty graph G together with a family of feasible realizations $\mathcal{R}$ and a probability distribution p on these realizations. We use $R \sim_p \mathcal{R}$ to denote a realization R drawn from $\mathcal{R}$ according to p. For such a randomized instance, we show that for no deterministic algorithm the expected ratio of ALG/OPT is less than 2. Applying a variant of Yao's Principle [3,19], this yields that no randomized algorithm has competitive ratio smaller than 2.

THEOREM 28 (Variant of Yao's Principle [3, Thm. 8.5]). *Let $\mathcal{A}$ denote the class of all deterministic algorithms and let $\mathcal{F}$ be the family of all randomized instances $(G, \mathcal{R}, p)$ for a minimization problem. Then any randomized algorithm has a competitive ratio c for which holds*

$$c \geq \min_{ALG \in \mathcal{A}} \mathbb{E}_{R \sim_p \mathcal{R}} \left[\frac{ALG(G, R)}{OPT(G, R)} \right] \qquad \forall (G, \mathcal{R}, p) \in \mathcal{F}.$$

Consider the uncertainty graph depicted in Figure 3, with two edges f and g joining the same pair of vertices and uncertainty intervals $A_f = (1, 3)$ and $A_g = (0, 2)$. For a fixed parameter $n \in \mathbb{Z}_{>0}$ we first define the family $\mathcal{R}_n$ of $2n$ feasible realizations and then give a probability distribution on them. Let realization R_j reveal for $j-1$ queries to edge f uncertainty interval $(1, 3)$ and for the j-th query interval $[2, 2]$. Here the uncertainty interval of edge g stays $(0, 2)$ for n queries and turns to the trivial uncertainty interval $[1, 1]$ upon the $n+1$-st query. Symmetrically let realization R_{-j} reveal edge weight $[1, 1]$ upon the j-th query to edge g and edge weight $[2, 2]$ with the $n+1$-st query to edge f. Then the optimal strategies for realizations R_j and R_{-j} on G are to query edge f or respectively edge g repeatedly for j times and thus make j queries in total.

We define a randomized instance $(G, \mathcal{R}_n, p)$ by giving a distribution p over the realizations in $\mathcal{R}_n$. Let each of the $2n$ realizations R_j and $R_{-j}, j = 1, \ldots, n$, occur with probability $P(R_j) = P(R_{-j}) = 1/2n$ in the distribution p.

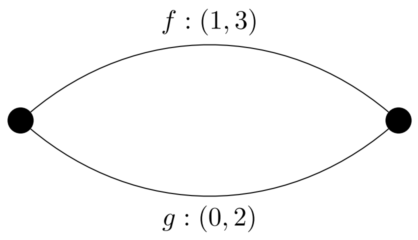

FIGURE 3. *Lower Bound Example for OP-OP Randomized*

Consider the algorithm ALG, that alternates between querying edge f and edge g. If we denote by f^i, g^i the i-th query to edges f and g, the query sequence of the algorithm is: $f^1, g^1, f^2, g^2, \dots, f^n, g^n$. We compute the competitive ratio of ALG and then show its performance is best-possible.

LEMMA 29. *The Algorithm ALG defined above has competitive ratio at least* 2.

Proof. Consider the randomized instance $(G, \mathcal{R}_n, p)$ for $n \in \mathbb{Z}_{>0}$ defined above. We show algorithm ALG has competitive ratio 2 when n tends to infinity. The algorithm ALG needs $2j - 1$ queries when realization R_j occurs, as it queries edge f for the j-th time after querying both edges $j-1$ times. For realization R_{-j} it needs $2j$ queries. The optimal query set has size j for both realizations R_j and R_{-j}. Thus the competitive ratio for the randomized instance $(G, \mathcal{R}_n, p)$ is:

$$\mathbb{E}_{R\sim_p \mathcal{R}_n}\left[\frac{\text{ALG}(G,R)}{\text{OPT}(G,R)}\right] = \sum_{j=1}^{n} P(R_j)\frac{2j-1}{j} + \sum_{j=1}^{n} P(R_{-j})\frac{2j}{j} = 2 - \frac{1}{2n}\sum_{j=1}^{n}\frac{1}{j}.$$

The sum expresses the harmonic number H_n, which has growth less than $1/n$, and thus we get

$$\mathbb{E}_{R\sim_p \mathcal{R}_n}\left[\frac{\text{ALG}(G,R)}{\text{OPT}(G,R)}\right] = 2 - \frac{1}{2n}\cdot H_n \overset{n\to\infty}{\longrightarrow} 2.$$

This proves Algorithm ALG has competitive ratio at least 2 on the randomized instance $(G, \mathcal{R}_n, p)$ and thus also in general. ☐

LEMMA 30. *No algorithm performs better than ALG on the randomized problem instance $\mathcal{R}_n$ with probability distribution p.*

Proof. We observe first, that any algorithm obeying the principle that one edge is queried for the i-th time only after the other edge has been queried for $i-1$ times has the same competitive ratio as ALG, as the family of realizations $\mathcal{R}_n$ and the probability distribution p are symmetric in f and g.

Now consider an algorithm ALG_1 not obeying this principle. Its query sequence contains edges f and g each n times in an arbitrary order. By definition it has a point in the query sequence, where the number of queries to edge f and to edge g differs by at least 2. Then the query sequence also contains two consecutive queries whose query numbers differ by at least 2. Without loss of generality assume their order is g^y, f^x and $y \geq x + 2$ for two integers x and y. We define a new algorithm ALG_2 and show that it has strictly smaller competitive ratio. Let ALG_2 be the algorithm where we switch these two queries g^y, f^x and that thus contains the sequence f^x, g^y.

The number of queries of ALG_1 and ALG_2 coincides for all realizations $R \notin \{R_x, R_{-y}\}$. Using the linearity of expected values, this means the difference of the two competitive ratios simplifies to

$$\mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}_2(G,R)}{\mathrm{OPT}(G,R)}\right] - \mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}_1(G,R)}{\mathrm{OPT}(G,R)}\right]$$
$$= \mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}_2(G,R)-\mathrm{ALG}_1(G,R)}{\mathrm{OPT}(G,R)}\right] = \frac{P(R_{-y})\cdot 1}{\mathrm{OPT}(G,R_{-y})} - \frac{P(R_x)\cdot 1}{\mathrm{OPT}(G,R_x)}.$$

The number of queries for realization R_x is one larger for Algorithm ALG_1 than for ALG_2. For realization R_{-y} it is the other way around. Hence, we get

$$\mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}_2(G,R)}{\mathrm{OPT}(G,R)}\right] - \mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}_1(G,R)}{\mathrm{OPT}(G,R)}\right] = \frac{1}{2n}\left(\frac{1}{y}-\frac{1}{x}\right) = \frac{x-y}{2nxy} < 0.$$

This yields that the performance of ALG_2 is strictly better than the performance of ALG_1. Any algorithm that queries f and g alternatingly has competitive ratio 2 and any other algorithm is not best-possible. Thus algorithm ALG with competitive ratio 2 is best-possible for the randomized instance $(G, \mathcal{R}_n, p)$. ∎

We apply Yao's Principle (Theorem 28) to Lemma 30 to prove our claim.

Theorem 31. *There is no randomized algorithm for* MST under uncertainty *in the OP-OP model with competitive ratio $c < 2$.*

Corollary 32. *The 2-competitive deterministic algorithm presented by Gupta et al. [11] has best-possible competitive ratio, even among randomized algorithms.*

10. Approximation. We return to the model, in which a single query to an edge reveals its exact weight. We consider an approximate variant in which we relax the requirement that an online algorithm must guarantee an exact MST and allow it to compute an α-approximate MST instead. More precisely, an algorithm is α-approximate if its query set $Q \subseteq E$ identifies one spanning tree that has weight at most α times the weight of an MST for any realization of edge weights $w_e \in A_e$ for $e \in E \setminus Q$. As before, we evaluate an algorithm's performance by competitive analysis. Note, we only relax the verification requirement for the algorithm, not for the optimum we compare with. The optimal query set still needs to verify an exact MST in the uncertainty graph. We show that despite this significant relaxation of the verification requirements for the algorithm, no performance improvement is possible—for any approximation factor α.

Theorem 33. *For any $\alpha > 1$, there is no α-approximate algorithm for* MST under uncertainty *with competitive ratio $c < 2$. Furthermore, there is no randomized α-approximate algorithm for* MST under uncertainty *with competitive ratio $c < 1.5$.*

Proof. Consider the uncertainty graph displayed in Figure 4 for a fixed approximation ratio $\alpha > 1$. Any deterministic algorithm queries either edge f or edge g first. For each of the two choices, we give a realization which does not give enough information to determine an α-approximate MST without a second query, whereas an optimal algorithm can compute the exact MST with a single query. This yields a lower bound of 2 on the competitive ratio.

Realization $\mathcal{R}_1$ has weights $w_f = 1, w_g = 1/(2\alpha)$. Then the optimal query set is $\{g\}$ and has size one. However, any algorithm which queries edge f first, cannot verify an α-approximate MST after one query. The algorithm has not yet queried

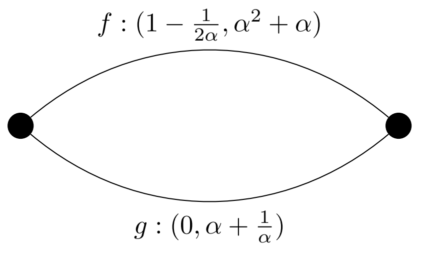
$f : (1 - \frac{1}{2\alpha}, \alpha^2 + \alpha)$
$g : (0, \alpha + \frac{1}{\alpha})$

FIGURE 4. *Lower Bound Example for α-approximate* MST under uncertainty

edge g but has to choose an α-approximate MST for all possible edge weights of edge g. However, edge f is not an α-approximate MST for $w_g = 1/(2\alpha)$ and edge g is not an α-approximate MST for $w_g = \alpha + 1/(2\alpha)$. Thus the algorithm also needs to query edge g to verify an α-approximate MST and consequently uses twice as many queries as the optimal algorithm.

Symmetrically we consider the realization $\mathcal{R}_2$, where edge $\{f\}$ is the optimal query set and the edges have weights $w_f = \alpha^2 + 1/\alpha, w_g = \alpha$. Here, an algorithm querying edge g first cannot verify an α-approximate MST, as edge f is not an α-approximate MST for $w_f = \alpha^2 + 1/\alpha$ and edge g is not an α-approximate MST for $w_f = 1 - 1/(3\alpha)$. Hence, again the algorithm needs two queries to find an α-approximate MST while an optimal algorithm needs only one query to find the MST.

Any randomized algorithm chooses the algorithm fg with some probability p and gf otherwise. This means it needs $2p + (1-p)$ queries in expectation for realization $\mathcal{R}_1$ and $p + 2(1-p)$ queries in expectation for realization $\mathcal{R}_2$. The maximum of these two terms is minimized for $p = 1/2$. Then this algorithm needs 1.5 queries in expectation for each of the two realizations. As before, the optimal query set has size 1 for both realizations, yielding a lower bound 1.5 on the competitive ratio. ∎

Acknowledgments. We thank the anonymous referees for numerous helpful comments and insights that improved the presentation of the paper.

REFERENCES

[1] A. Ben-Tal, L. El Ghaoui, and A. S. Nemirovski, *Robust Optimization*, Princeton Series in Applied Mathematics, Princeton University Press, 2009.
[2] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*, Springer Series in Operations Research, Springer, 1997.
[3] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.
[4] T. Erlebach, *Query-competitive algorithms for computing with uncertainty*, Bulletin of the EATCS, 116 (2015).
[5] T. Erlebach and M. Hoffmann, *Minimum spanning tree verification under uncertainty*, in Proceedings of WG, 2014, pp. 164–175.
[6] T. Erlebach, M. Hoffmann, and F. Kammer, *Query-competitive algorithms for cheapest set problems under uncertainty*, Theor. Comput. Sci., 613 (2016), pp. 51–64.
[7] T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihalák, and R. Raman, *Computing minimum spanning trees with uncertainty*, in Proceedings of STACS, 2008, pp. 277–288.
[8] T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy, *Computing shortest paths with uncertainty*, Journal of Algorithms, 62 (2007), pp. 1–18.
[9] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom, *Computing the median with uncertainty*, SIAM Journal on Computing, 32 (2003), pp. 538–547.
[10] M. Goerigk, M. Gupta, J. Ide, A. Schöbel, and S. Sen, *The robust knapsack problem with queries*, Computers & OR, 55 (2015), pp. 12–22.

[11] M. Gupta, Y. Sabharwal, and S. Sen, *The update complexity of selection and related problems*, Theory Comput. Syst., 59 (2016), pp. 112–132.
[12] S. Kahan, *A model for data in motion*, in Proceedings of STOC, 1991, pp. 267–277.
[13] S. Khanna and W. C. Tan, *On computing functions with uncertainty*, in Proceedings of PODS, 2001, pp. 171–182.
[14] B. Korte and J. Vygen, *Combinatorial optimization*, vol. 21, Springer, 2012.
[15] N. Megow, J. Meissner, and M. Skutella, *Randomization helps computing a minimum spanning tree under uncertainty*, in Proceedings of ESA, 2015, pp. 878–890.
[16] C. Olston and J. Widom, *Offering a precision-performance tradeoff for aggregation queries over replicated data*, in Proceedings of VLDB, 2000, pp. 144–155.
[17] P. Patil, A. P. Shrotri, and A. R. Dandekar, *Management of uncertainty in supply chain*, Int. J. of Emerging Technology and Advanced Engineering, 2 (2012), pp. 303–308.
[18] Y. Wang and S. C.-W. Wong, *Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm*, in Proceedings of ICALP, 2015, pp. 1070–1081.
[19] A. C.-C. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in Proceedings of the 17th Annual Symposium on Foundations of Computer Science (FOCS), 1977, pp. 222–227.