Minimum Spanning Tree under Explorable Uncertainty in Theory and Experiments*

Jacob Focke[1], Nicole Megow[2], and Julie Meißner[3]

1 **Department of Computer Science, University of Oxford, Oxford, UK**
jacob.focke@cs.ox.ac.uk
2 **Department of Mathematics and Computer Science, University of Bremen, Bremen, Germany**
nicole.megow@uni-bremen.de
3 **Institute of Mathematics, Technical University of Berlin, Berlin, Germany**
jmeiss@math.tu-berlin.de

Abstract

We consider the minimum spanning tree (MST) problem in an uncertainty model where uncertain edge weights can be explored at extra cost. The task is to find an MST by querying a minimum number of edges for their exact weight. This problem has received quite some attention from the algorithms theory community. In this paper, we conduct the first practical experiments for MST under uncertainty, theoretically compare three known algorithms, and compare theoretical with practical behavior of the algorithms. Among others, we observe that the average performance and the absolute number of queries are both far from the theoretical worst-case bounds. Furthermore, we investigate a known general preprocessing procedure and develop an implementation thereof that maximally reduces the data uncertainty. We also characterize a class of instances that is solved completely by our preprocessing. Our experiments are based on practical data from an application in telecommunications and uncertainty instances generated from the standard TSPLib graph library.

1998 ACM Subject Classification G.2.1 Combinatorial Algorithms, F.2.1 Computations on Discrete Structures

Keywords and phrases MST, explorable uncertainty, competitive ratio, experimental algorithms

Digital Object Identifier 10.4230/LIPIcs.SEA.2017.22

1 Introduction

Uncertain data is a common issue in many real-world optimization problems. While it is clear that uncertain data can not be completely avoided, improved or exact data can often be explored at an additional cost. Classical approaches to optimization under uncertainty such as robust, stochastic, and online optimization do not capture this possibility. Uncertainty exploration takes a different approach by taking into account the exploration of uncertain data at extra cost. Here, the goal is to quantify the trade-off between an investment in more precise data and the resulting quality for the solution to the optimization problem. A major research line in this context asks for the minimum exploration cost to find an optimal solution. In a sense, this is the opposite of robust optimization that aims for the best solution with zero exploration cost.

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

Applications for optimization under explorable uncertainty can be found, e.g., in telecommunication or infrastructure network design, where either construction cost estimates can be improved through expert advice, or approximate connection lengths can be ensured through field measurements. Other optimization problems that allow uncertainty exploration are user demand estimates that can be improved through user surveys, and weather predictions that can be enhanced by both, additional measurements and more computational power.

In this paper, we consider the minimum spanning tree (MST) problem in the uncertainty exploration model. For each edge, we are given an uncertainty interval in which the exact edge weight lies. We can query each edge to find out its exact weight. The goal is to minimize the number of queries needed until we find a minimum spanning tree. As a performance measure we use the competitive ratio, that is, the worst-case ratio between the number of queries needed by an algorithm and the minimum number of queries required when given the exact data in advance. Precise definitions follow at the end of this section.

The MST problem, as one of the most fundamental and practically relevant combinatorial optimizations problems, has been investigated intensively in the uncertainty exploration model from the theoretical perspective. Several algorithms with provable worst-case guarantees are known [8, 15]. In this paper, we compare these algorithms theoretically, conduct the first practical experiments, and compare the theoretical and practical behavior of the algorithms. Furthermore, we investigate a preprocessing that was proposed in [15]. We develop an implementation thereof, for which we guarantee that it maximally reduces the data uncertainty. This is not only theoretically interesting but also practically relevant, as it reduces the data uncertainty that remains when starting an (arbitrary) algorithm.

We run our experiments on two different data sets. The first set of data is from a telecommunication service provider. It describes a problem that appears when expanding a cable network to a new roll-out area. First, the facility locations are chosen, that need to be connected. The exact connection costs between the facilities are unknown and can only be explored through costly field measurements. We find the best MST under uncertainty for these instances. We complement this practical data by a second data set, which we generate based on graphs available in the well-known graph library TSPLib [16].

Related Work. Optimization under uncertainty is an important, well-studied topic in theory and practice. The major lines of research are robust optimization [2], online optimization [4], and stochastic optimization [3], each modeling uncertain information in a different way. The first model where uncertain information can be explicitly explored at a fixed cost was studied by Kahan [14]. He investigated finding the maximum and median of a set of values known to lie in given uncertainty intervals. The recent survey by Erlebach and Hoffmann [6] gives a nice overview on research in the uncertainty exploration model. Various problems have been studied, including the k-th smallest value in a set of uncertainty intervals [14, 13, 10] and classical combinatorial optimization problems, such as shortest path [9], finding the median [10], the MST problem [8, 15, 11], the cheapest set problem [7] and the knapsack problem [12]. The latter work on the knapsack problem seems to be the only one that contains computational experiments conducted in this field.

The MST problem with uncertain edge weights was introduced by Erlebach et al. [8]. Their deterministic algorithm achieves an optimal competitive ratio of 2. A simplification of this algorithm that omits a repetitive restart by preserving the competitive ratio was given in [15]. Also the existence of a dual algorithm is observed. The main contribution in [15] is a randomized algorithm with expected competitive ratio of $1 + 1/\sqrt{2} \approx 1.707$ whereas the best-known lower bound is 1.5. The offline problem of finding the optimal query set for a given realization of edge weights can be solved in polynomial time [5].

Our Contribution. We theoretically compare the three algorithms for MST under uncertainty, make practical experiments and showcase similarities and differences between theoretical and practical observations. For the algorithms we define the best-possible preprocessing in Section 3 and characterize a class of instances which it solves completely. We observe exactly this structure in one of the practical data sets. Our experiments show that the average competitive ratio is small and the total number of queries as well. The comparison of theory and practice in Sections 2 and 5 shows that the competitive ratio and the variance in the size of an optimal solution are far from the worst-case. While theoretically there are instances on which the two deterministic algorithms show opposing behavior, in our experiments their performance is almost identical for all instances. We show that there are instances on which the two deterministic algorithms perform better than the randomized one. Surprisingly, we observe this behavior for the telecommunication data. For the TSPLib data the opposite happens and the randomized algorithm has a significantly smaller competitive ratio. Conducting the first experiments, we saw that the implementation hurdle is small and our run times are reasonably small, even though we did not optimize on it.

Problem Definition and Notation. Given an undirected, connected graph (V, E) with $|V| = n$ and $|E| = m$, we associate with each edge $e \in E$ an *uncertainty interval* A_e. This interval constitutes the only information about e's unknown weight $w_e \in A_e$. Such an interval is either *trivial*, $A_e = [L_e, U_e], L_e = U_e$, or it is *non-trivial*, i. e., $A_e = (L_e, U_e)$, with lower limit L_e and upper limit $U_e > L_e$. Closed, non-trivial intervals cannot be allowed as they lead to a non-constant competitive ratio [8]. We call an edge *trivial* if it has a trivial uncertainty interval, *non-trivial* otherwise. Let $\mathcal{A}$ be the set of uncertainty intervals for E. Then an instance of our problem is an *uncertainty graph* $G = (V, E, \mathcal{A})$ together with a *realization* $\mathcal{R}$ *of edge weights* $(w_e)_{e \in E}$ which 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 realization $\mathcal{R}$ of edge weights. We may query any edge $e \in E$ and obtain its exact weight w_e according to $\mathcal{R}$. The goal is to determine an MST through a sequence of queries that is as short as possible. The resulting set of queries $Q \subseteq E$ is *feasible*, if there is a spanning tree which is an MST for every realization of edge weights $w_e \in A_e$ for $e \in E \setminus Q$ and the weights w_e defined by $\mathcal{R}$ for $e \in Q$. We call this problem *MST under uncertainty*.

We evaluate our algorithms by standard competitive analysis. An algorithm is *c-competitive* if, for any realization $\mathcal{R} = (w_e)_{e \in E}$, the number of queries is at most c times the optimal query number. For a fixed realization $\mathcal{R}$, the optimal number of queries describes the minimum size of a query set that verifies an MST. The *competitive ratio* of an algorithm is the infimum over all c such that the algorithm is c-competitive. For randomized algorithms we compare the expected number of queries to the optimal number of queries.

2 Introduction of Algorithms and Theoretical Comparison

In this section we discuss known algorithms for MST under uncertainty. The first (deterministic) algorithm Ured was introduced by Erlebach et al. [8]. It achieves the best-possible competitive ratio of 2. Subsequently, two deterministic algorithms Cycle and Cut with competitive ratio 2 were given by Megow et al. [15]. Originally, they were presented for the more general problem of computing a minimum weight matroid basis in the uncertainty exploration model. Applying Cycle to computing an MST, it can be interpreted as a variant of Ured without repeated restarts and, thus, we consider here only the simplified variant. The randomized algorithm Random with competitive ratio 1.707 and a preprocessing were given also in [15]. Here the best known lower bound is 1.5.

In our paper we consider the three algorithms Cycle, Cut and Random, which we briefly describe below; pseudo-code can be found in the appendix. We apply the preprocessing on the input to *all* three algorithms. Details on the preprocessing follow in Section 3.

Deterministic algorithm Cycle. The algorithm Cycle is a worst-out greedy algorithm that is based on the following MST characterization: The largest-weight edge in a cycle is not in any MST. It starts out with a candidate minimum spanning tree and then iteratively considers the other edges. Each additional edge defines a cycle together with the candidate tree. On this cycle, the two edges with largest upper limit are queried repeatedly, until we either verify the additional edge has largest weight or we find an edge of larger weight on the cycle. In the latter case we improve the tree by exchanging the two edges.

Deterministic algorithm Cut. Cut is the dual algorithm to Cycle, that is defined by matroid duality. It uses that the minimum-weight edge in a cut is in an MST. Like in the previous algorithm, Cut starts with a candidate MST, but iteratively considers the tree edges. Deleting a tree edge defines a cut. On this cut we repeatedly query the two edges with smallest lower limit, until we either verify that the tree edge has the smallest weight in the cut or find an edge of smaller weight to replace the candidate tree edge.

Randomized algorithm. The randomized algorithm Random crucially needs a preprocessed instance with the following structural property: For any cycle appearing in the algorithm, any feasible query set contains either the edge with largest upper limit e or all edges with overlapping interval, i.e., whose uncertainty interval contains the lower limit L_e. The preprocessing, which we discuss in detail in Section 3, guarantees this property [15].

The algorithm Random uses the same structure as Cycle. Starting out with a candidate MST, it iteratively considers the remaining edges not in this tree. For each remaining edge, the algorithm inspects the (unique) cycle it closes in the MST to see which of its edges should be queried. The preprocessing yields that on each such cycle any feasible query set contains either the edge with the largest upper limit, say f, or all cycle edges whose intervals overlap with that of f. The algorithm either queries the largest edge or all overlapping edges at once. To balance this decision over several cycles closed during the algorithm, Random introduces a potential for each edge. In each cycle additional potential is distributed to all overlapping edges such that they reach an equal level. Depending on the resulting amount of potential, either these edges or the edge with largest upper limit are queried. This decision is randomized by comparing the potential to a randomly chosen uniform threshold.

2.1 Comparing the Deterministic Algorithms

We show that there are instances on which Cycle and Cut have an opposing performance, meaning that one algorithm is near-optimal and the other shows its worst-case performance. Intuitively, the instance is solved by querying the edges of a single cycle C and Cycle queries pairs of edges on C only. Cut, however, almost exclusively queries pairs with only one edge in C. The reverse holds for instances, in which it suffices to query the edges of a single cut.

Our graph class SP consists of a path of edges S and a set of parallel edges P, each of which closes a cycle with S. We give two realizations $\mathcal{R}_1$ and $\mathcal{R}_2$ in Figure 1.

For $\mathcal{R}_1$ the set S is the unique optimal query set and a query set is a feasible solution only if it contains S. The first cycle closed by the algorithm Cycle contains S and exactly one edge of P. It queries all edges on this cycle, which is a feasible solution of size $|S| + 1$. Cut on the other hand considers cuts of the form $P + \{s\}$ with a non-queried edge $s \in S$.

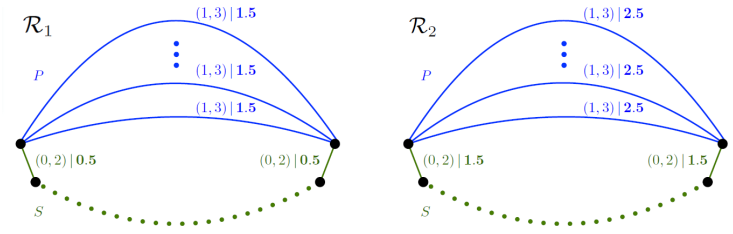

Figure 1 Different realizations for the class of uncertainty graphs SP lead to different extremes in the behavior of Cycle and Cut. Edge labels: $(L_e, U_e) \,|\, \mathbf{w_e}$.

There are $|S|$ such cuts. For each, Cut queries a pair of edges as long as there are non-queried edges left in P. Thus, it queries $|S| + \min\{|S|, |P|\}$ edges. By choosing S and P of appropriate cardinality we can achieve every performance ratio $q \in (1, 2]$ for Cut. In particular, for $|S| \leq |P|$ and $|S| \to \infty$, the performance ratio of Cycle approaches 1 and the ratio of Cut is 2.

The reverse holds for the realization $\mathcal{R}_2$. In this case a feasible query set has to contain P, Cut finds a solution of size $|P| + 1$, and Cycle queries $|P| + \min\{|S|, |P|\}$ edges.

▶ **Observation 1.** *For any rational $q \in (1, 2]$, there exists a graph in the class SP and a realization such that* Cycle *(*Cut*) is near-optimal whereas* Cut *(*Cycle*) yields a performance ratio of q.*

Thus, theoretically the query set sizes can vary greatly for Cycle and Cut. However, we do not observe this behavior for any of the instances in our experiments.

2.2 Comparing Randomized and Deterministic Algorithms

We show that Random can be optimal for worst-case instances of Cycle and Cut, and – somewhat surprisingly – the reverse is also possible.

Consider a family of cycles joined in one node, each with three edges f, g, h with uncertainty intervals $(1, 4)$, $(0, 3)$ and $[1, 1]$ respectively. Further, edge f has weight 3 and edge g has weight 1. Then, Random terminates with a single query of either f or g in each cycle, while Cycle and Cut query both f and g.

▶ **Observation 2.** *There are instances, for which* Random *finds an optimal solution, while* Cycle *and* Cut *achieve their worst-case ratio of* 2*.*

A similar instance evokes the reverse performance behavior. Consider a cycle C with k edges e_i with interval $(0, 3)$, one edge g with interval $(0, 4)$ and one edge f with interval $(1, 5)$. We choose the weights as $w_{e_i} = 2$ and $w_g = w_f = 3$. Then Cycle and Cut query only edges f and g, which is optimal, but Random yields its worst-case ratio $1 + 1/\sqrt{2}$ for $k \to \infty$.

▶ **Observation 3.** *There exists a family of uncertainty graphs, for which* Cycle *and* Cut *perform optimally, whereas* Random *asymptotically shows its worst case behavior.*

2.3 Variance of OPT

We investigate the variance of the optimal number of queries, Opt, under different realizations for a fixed input instance. We give an example instance in which small perturbations in the realization significantly change the value of Opt.

Algorithm 1 PREPROCESSING $(\prec_\ell, \prec_u)$

Input: An uncertainty graph $G = (V, E, \mathcal{A})$.
Output: A query set $Q \subseteq E$ and the two trees T_ℓ, T_u.

```
Q ← ∅.
Determine T_ℓ and T_u according to ≺_ℓ and ≺_u respectively using Prim's algorithm [1].
while T_ℓ \ T_u contains a non-trivial edge do
    Query all non-trivial edges in T_ℓ \ T_u, and add them to Q.
    Update T_ℓ and T_u.
return The query set Q and the two trees T_ℓ, T_u.
```

Consider a cycle C of length m consisting of an edge f with uncertainty interval $(1, 4)$ and $m-1$ identical edges $\{g_1, \ldots, g_{m-1}\} =: \mathcal{G}$ with uncertainty interval $(0, 3)$ and weight 2. If we set the weight of f to be 3, it suffices to query f and $\textsc{Opt} = 1$. On the other hand, if the weight of f is 2, all edges in C have to be queried and $\textsc{Opt} = m$. Interestingly, we do not observe this large variance in our experiments (see Section 5: The optimal solution).

▶ **Observation 4.** *For a fixed uncertainty graph* Opt *can vary greatly even for minor changes of the underlying realization.*

3 Preprocessing

Preprocessing aims at simplifying the input instance, that is, we identify and query edges that must be queried by any algorithm including the optimal one. Naturally, we want to query as many such edges as possible before starting the actual algorithm.

There is a characterization of (a subset of) such edges that relies on the following definition. Given an instance of MST under uncertainty, the *lower limit tree* T_ℓ is an MST for the realization w^ℓ, in which all edge weights of edges with non-trivial uncertainty interval are arbitrarily close to their lower limits, more precisely $w^\ell_e = L_e + \varepsilon$ for infinitesimally small $\varepsilon > 0$. The *upper limit tree* $T_u \subseteq E$ is an MST for the realization in which edges have weights arbitrarily close to their upper limit, that is, $w^u_e = U_e - \varepsilon$. Note, that the order relation of edges with identical lower (upper) limit is yet unspecified. By $\prec_\ell$ and $\prec_u$ we denote an arbitrary but fixed pair of total orderings of edges, w.r.t. which we obtain a lower limit tree T_ℓ and upper limit tree T_u respectively.

▶ **Theorem 5** ([15])**.** *Given an uncertainty graph with lower and upper limit trees T_ℓ, T_u, any non-trivial edge $e \in T_\ell \setminus T_u$ is in every feasible query set for any realization.*

The preprocessing in [15] iteratively computes the trees T_ℓ and T_u and queries the edges in the set $T_\ell \setminus T_u$ until this set contains only edges with trivial uncertainty interval; see Alg. 1. The choice of $\prec_\ell$ and $\prec_u$ and thus specifying the order relation of edges with identical lower (upper) limit raises the potential for good or bad choices. As an example, consider a graph of k identical two-edge cycles that are all joined in one node. Each cycle is of the form $C = \{e_1, e_2\}$, where all edges have the same lower limit L and upper limits $U_1 < U_2$. Then, for any ordering $\prec_u$ the upper limit tree T_u does not contain e_2 for each of the cycles. For the lower limit ordering $\prec_\ell$, all orderings are feasible. For the ordering $e_1 \prec e_2$, we have $T_\ell = T_u$ and the preprocessing does not query any edge. However, for the ordering $e_2 \prec e_1$ the two trees are disjoint and k edges are queried in the first iteration of the preprocessing.

Observing this significant impact, we define a specific pair of total orderings $\prec_L, \prec_U$ on the edges and we prove that the algorithm above, PREPROCESSING $(\prec_L, \prec_U)$, maximizes the total number of queries.

▶ **Definition 6** (Limit Orders and Trees). Let $G = (V, E, \mathcal{A})$ be an uncertainty graph and let $e_1, \ldots, e_m$ be an arbitrary but fixed labeling of the edges in E. Then we define two orderings for the edges in E.

Lower Limit Order: $e_i \prec_L e_j$, if $L_{e_i} < L_{e_j}$ or if $L_{e_i} = L_{e_j}$ and one of the following holds:

- **(i)** e_i trivial and e_j non-trivial
- **(ii)** $U_{e_i} > U_{e_j}$ and e_j non-trivial
- **(iii)** $U_{e_i} = U_{e_j}$ and $i < j$.

Upper Limit Order: $e_i \prec_U e_j$, if $U_{e_i} < U_{e_j}$ or if $U_{e_i} = U_{e_j}$ and one of the following holds:

- **(i)** e_j trivial and e_i non-trivial
- **(ii)** $L_{e_i} > L_{e_j}$ and e_i non-trivial
- **(iii)** $L_{e_i} = L_{e_j}$ and $j < i$.

We call the corresponding lower and upper limit trees T_L and T_U.

We show that Preprocessing $(\prec_L, \prec_U)$ queries all edges which are in $T_\ell \setminus T_u$ for any other pair of orderings $\prec_\ell, \prec_u$. As a first step, it is not hard to see that an edge e, which is contained in $T_\ell \setminus T_u$ for some fixed orderings $\prec_\ell$ and $\prec_u$, remains in this set independently from queries of edges other than e.

▶ **Lemma 7.** *An edge in $T_\ell \setminus T_u$ remains in the set $T_\ell \setminus T_u$ until it is queried.*

Proof. Let e be in $T_\ell \setminus T_u$. As long as e is not queried, its interval limits do not change. Querying other edges only increases their lower limits and decreases their upper limits. Hence, e stays in T_ℓ and remains excluded from T_u. ◀

Next we show that Preprocessing $(\prec_L, \prec_U)$ does not terminate while there is a non-trivial edge in $T_\ell \setminus T_u$. The proof of Lemma 8 considers an edge e in $T_\ell \setminus T_u$ and proves the statement separately for the three cases $e \in T_L$, $e \notin T_L$ and $e \notin T_U$ as well as $e \in T_U \setminus T_L$.

▶ **Lemma 8.** *If there is a non-trivial edge in $T_\ell \setminus T_u$, then there is also one in $T_L \setminus T_U$.*

Proof. Assume there is a non-trivial edge $e \in T_\ell \setminus T_u$, but $T_L \setminus T_U$ contains only trivial edges. We distinguish three cases. If e is in T_L, it is also in T_U. Then there is an edge h, which is in the cut in $T_U \setminus e$ and in the cycle in $T_u \cup e$. As it is in the cut, we have $U_h \geq U_e$. At the same time, the cycle shows $U_h \leq U_e$, such that the two upper limits must be equal. Then, the fact that h is in the cut, but not in T_U means $L_e \geq L_h$. If h is trivial, e must also be trivial, which contradicts our assumption. Otherwise, as we choose $h \notin T_U$ and $T_L \setminus T_U$ contains only trivial edges, edge h is also not in T_L. If h is not in the cut $T_L \setminus e$, there must be an edge g in $T_L \setminus T_U$ that is in the cut $T_U \setminus e$ and in the cycle in $T_L \cup h$. This edge g is trivial, larger than e in the ordering $\prec_U$ and smaller than h in the ordering $\prec_L$. This means together with the observations about the bounds of e and h we made above, that we have $U_g \geq U_e = U_h$ and $L_e \geq L_h \geq L_g$. Thus e and h are both trivial: a contradiction. Alternatively we consider h is in the cut $T_L \setminus e$, where only edges at least as large as e are contained. This means $L_e \leq L_h$ and consequently $L_e = L_h$. The edge h is in the cut $T_L \setminus e$ and in the cut $T_U \setminus e$, which means we have $e \prec_L h$ and $e \prec_U h$. However, this contradicts that the intervals of e and h are identical.

If e is not in T_L and not in T_U, then there is an edge h, which is in the cut $T_\ell \setminus e$ and in the cycle in $T_L \cup e$. As it is in the cut, we have $L_e \leq L_h$ and h non-trivial, and as it is in the cycle we have $L_e \geq L_h$. Thus, we have $L_e = L_h$ and $U_h \geq U_e$ because of the ordering $\prec_L$. We choose $h \in T_L$. As $T_L \setminus T_U$ contains only trivial edges, edge h is also in T_U. If h is not in

the cycle $T_U \cup e$, there must be an edge g in $T_L \setminus T_U$ that is in the cut $T_U \setminus h$ and in the cycle $T_L \cup e$. This edge g is trivial, larger than h in the ordering $\prec_U$ and smaller than e in the ordering $\prec_L$. This means together with the observations about the bounds of e and h we made above, that we have $U_g \geq U_h \geq U_e$ and $L_h = L_e \geq L_g$. Thus e and h are both trivial: a contradiction. Alternatively we consider h is in the cycle $T_U \cup e$, where only edges with upper limit at most as large as e are contained. This means $U_h \leq U_e$ and consequently $U_h = U_e$. The edge h is in the cycle $T_U \cup e$ and in the cycle $T_L \cup e$, which means we have $h \prec_L e$ and $h \prec_U e$. However, this contradicts that the interval of e and h is identical.

Finally, we consider $e \in T_U \setminus T_L$. Then there is an edge h in the cut $T_U \setminus e$ and in the cycle $T_L \cup e$. This means $h \in T_L \setminus T_U$ and thus trivial. Additionally we have $e \prec_U h, h \prec_L e$, which means $L_h \leq L_e \leq U_e \leq U_h$. However, this is a contradiction as e is non-trivial. ◀

Combined, this means Preprocessing $(\prec_L, \prec_U)$ queries every non-trivial edge in $T_\ell \setminus T_u$.

▶ **Theorem 9.** Preprocessing *$(\prec_L, \prec_U)$ queries the union over all edges queried by* Preprocessing *$(\prec_\ell, \prec_u)$ for all orderings $\prec_\ell, \prec_u$. Thus, it queries the maximum number of edges characterized by Theorem 5.*

Proof. We show by induction over the number of iterations that edges queried in Preprocessing $(\prec_\ell, \prec_u)$ are also queried in Preprocessing $(\prec_L, \prec_U)$. By Lemma 7 and 8, all edges queried in iteration 1 of Preprocessing $(\prec_\ell, \prec_u)$ are also queried in our specific preprocessing. Let e be an edge queried in iteration $i > 1$ of Preprocessing $(\prec_\ell, \prec_u)$. Let S be the set of all edges queried in the previous iterations. Then, by induction, the set S is queried by Preprocessing $(\prec_L, \prec_U)$. Assume edge e is not queried by Preprocessing $(\prec_L, \prec_U)$. We consider Preprocessing $(\prec_\ell, \prec_u)$ in iteration i and additionally query all edges which are queried by Preprocessing $(\prec_L, \prec_U)$. By Lemma 7 edge e is still in $T_\ell \setminus T_u$ for this new uncertainty graph. However, this is exactly the uncertainty graph at the end of Preprocessing $(\prec_L, \prec_U)$. Thus, the termination of the algorithm at this point contradicts Lemma 8. ◀

3.1 Instances Solved by the Preprocessing

The preprocessing is a modification of the input instance and intuitively it simplifies it by removing uncertainty. We note, however, that in theory it can lead to a worse algorithm performance for specific input. Nevertheless, in our experiments, the preprocessing generally improves the performance ratio of our algorithms. One class of our data sets is even solved exactly by the preprocessing alone. We generalize this observation and characterize a family of uncertainty graphs which can be completely solved by our preprocessing.

▶ **Proposition 10.** *For uncertainty graphs, in which every cycle contains only edges with identical lower limit or only edges with identical upper limit,* Preprocessing $(\prec_L, \prec_U)$ *finds an optimal solution.*

Proof. The proof is by contradiction. Assume Preprocessing $(\prec_L, \prec_U)$ terminates with T_L and T_U and did not find a feasible query set. Then the uncertainty graph has a cycle C on which it is unclear which edge has the largest weight. All but one edge of C are in T_L. Assume, that originally all edges on the cycle had the same upper limit. If there is only one edge f with largest upper limit, all other edges on the cycle are trivial. Since it is unclear, which edge has largest weight on C, f cannot also have the largest lower limit. Thus f is non-trivial and in $T_L \setminus T_U$, which is a contradiction. Otherwise, there are two non-trivial edges e and f on C with largest upper limit, $e \in T_L$ and $f \notin T_U$. This means we can define

an alternative ordering $\prec_u$ with $f \prec_u e$ and thus $e \notin T_u$. Thus, for the preprocessing with orderings $\prec_L$ and $\prec_u$ we have $e \in T_L \setminus T_u$. By Theorem 9 this means PREPROCESSING $(\prec_L, \prec_U)$ queries e, a contradiction to e being non-trivial.

A cycle with identical lower limits can be treated analogously. ◀

4 Experimental Data

First, note that there is an inherent difficulty with practical experiments for exploration uncertainty. For a practical application the uncertainty intervals might be known as well as the exact edge weights of the *queried* edges. To decide the optimal number of queries necessary, in general one needs the exact edge weights of further edges. However, in practice there is no reason to explore additional edges after the solution has been found. Thus, even though we have practical data we need to generate a part of the instance.

Telecommunication. For the telecommunication data we have 5 different graphs of varying size with up to 1000 nodes available to us. For each of them we have two different sets of uncertainty intervals. In the first set, the terrain data, we consider the building cost uncertainty that arises from different terrains. The cost of a connection is limited by the construction cost per meter cable through a field and the cost under a paved street times the length of the connection. We draw the exact edge weight *uniformly* distributed in the interval. In this uncertainty setting, exploring the exact weight of an edge represents the time or cost investment it takes to identify the terrain of a particular connection. The second data set we call existence data. This setting assumes that the terrain of the connection is known, but it is uncertain if existing infrastructure is available or not. As a result the interval ranges from almost no building cost due to existing infrastructure to a fixed building cost, which is roughly known in advance. The exact edge weight follows a *two-point distribution* close to the two endpoints of the interval. We maintain the ratio of 20% small weight to 80% large weight that is observed in practice.

TSPLib. We consider the 19 graphs for the symmetric traveling salesman problem TSP of the library TSPLib that have at most 100 nodes. They are usually used for TSP computations, but we compute their minimum spanning trees. The library contains the exact edge weights and we need to create corresponding uncertainty intervals. We choose the interval size proportional to the weight of each edge, which is a natural approach that we also observe in the telecommunication data. We experiment with the ratio between interval size and exact edge weight, let us call this ratio d, to generate difficult instances. As before, we consider intervals such that the realization is either uniformly distributed or two-point distributed at the two extremes. For an edge with weight w we draw the lower limit L uniformly at random in $((1-d)\cdot w, w)$ in the uniform case and set the upper limit U to $L + d \cdot w$. In the extremal case we choose the lower limit close to the edge weight w such that $L < w$ or we choose the upper limit U close to w with $w < U$ each with probability $1/2$. Then we choose the other limit accordingly. We computed the average competitive ratio of all three algorithms for the two distributions and various values for d between 0.001 and 0.5 for 190 uncertainty graphs; see Figure 6. As we are interested in a worst-case behavior, we choose for our experiments a uniform value $d = 0.065$ for which all algorithms have a rather large competitive ratio.

As one aspect of our experimental analysis, we investigate for a given graph the variance of certain parameters. We distinguish between the two data types: For the telecommunication

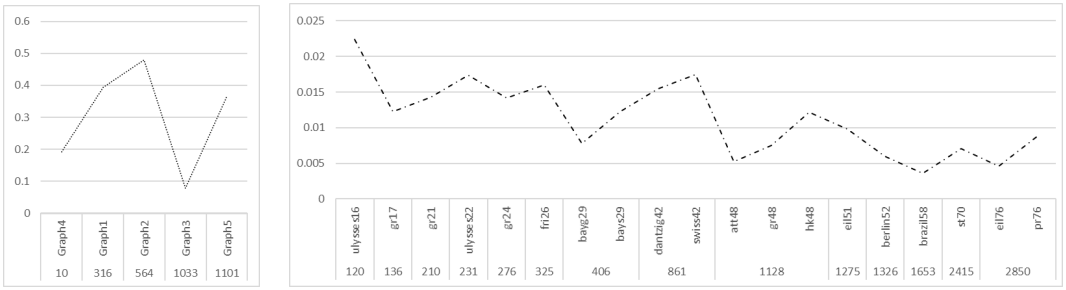

Figure 2 Size of the optimal solution OPT divided by the number of edges on the y-axis and the uncertainty graphs sorted by data set and increasing number of edges on the x-axis.

data the realization inside the uncertainty interval changes, while for the TSPLib data the location of the fixed length uncertainty interval around the also fixed realization changes.

5 Experimental Algorithm Analysis

For the detailed analysis we draw 100 uncertainty intervals/realizations for each graph in a data set, which yields 4800 instances in total. We perform our experiments with 20 repetitions of RANDOM per instance, as more repetitions did not alter the average performance. For each of the instances we compute the number of edges, the size of the query set in the preprocessing, the size of the optimal solution, the size of the query set for each of the three algorithms, the run time of the three algorithms as well as that of the preprocessing. For RANDOM we additionally compute the average number of edges on a cycle closed in the algorithm and the average number of edges on an algorithm cycle that have an uncertainty interval overlapping the one of the edge with largest upper limit. For the latter two parameters, we could not find a relation to the algorithm's performance. We summarize our experimental results in the following subsections. We make our code, the complete input and output data, and further analysis available at `http://www.coga.tu-berlin.de/fileadmin/i26/coga/MSTData.zip`.

5.1 The Optimal Solution

The size of the optimal solution OPT, that is, the minimum number of queries to find an MST, naturally grows with the size of the instance. To analyze a correlation, we consider the number of edges m as the instance size and determine the parameter OPT$/m$; see Figure 2. There are instances among the telecommunication data for which the ratio OPT$/m$ is as large as 0.5 and other ones where it is very small. Among this small number of instances the parameter behavior seems arbitrary. For the TSPLib data the ratio OPT$/m$ is a lot smaller and it decreases when m increases. Our theoretical analysis in Section 2.3 shows that for a single instance the behavior of this parameter can change between $1/m$ and 1. We do observe great variance for some instances of the telecommunication data, but very small variance for the TSPLib data. The variance is always far from the theoretical maximum variance.

5.2 Comparing Deterministic and Randomized Algorithms

For the telecommunication data the competitive ratio of all three algorithms has roughly the same average (cf. Figure 3). Averaging over all telecommunication instances CYCLE and CUT both have competitive ratio 1.18 and RANDOM has the slightly worse competitive ratio of

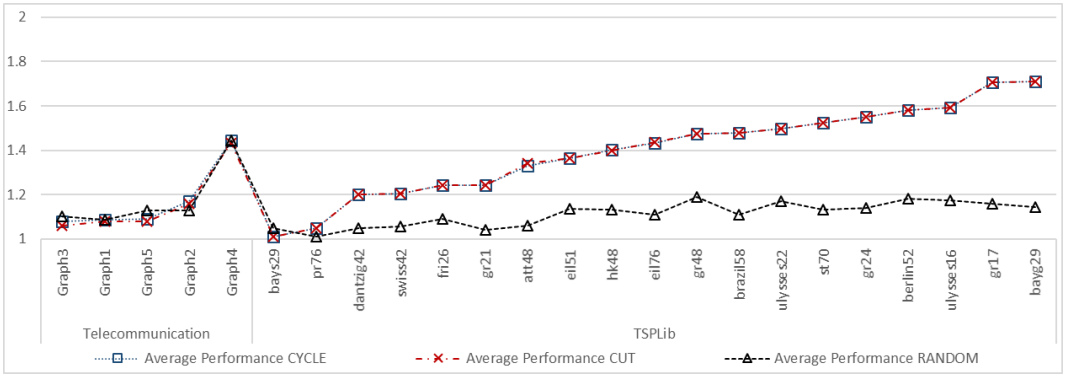

Figure 3 Average performance, i.e., the ratio of the algorithm query set size over the optimal query set size, for the three algorithms and the two data sets.

1.22. For the TSPLib data, the two deterministic algorithms have equal average competitive ratio 1.37, which is significantly larger than that for the telecommunication data. RANDOM has a notably smaller competitive ratio of 1.11 on average, that is even smaller than the ratio for the telecommunication data.

All average competitive ratios are far from their theoretical worst-case guarantee which is 2 for both deterministic algorithms and ≈ 1.707 for RANDOM. It is somewhat surprising, that despite the significant improvement of our randomized over the deterministic algorithms for the TSPLib data, there is no improvement for the telecommunication data. This means the usefulness of randomization depends on the considered data set. For the telecommunication data our way of randomization may even worsen the performance. This might seem counter-intuitive, but we give a theoretical explanation in Section 2.2.

5.3 Comparing the Deterministic Algorithms

In Section 2.1 we show that there can be a large difference between the performance ratio of CYCLE and CUT, even to the extreme case where one has ratio 1 and the other has ratio 2. However, as displayed in Figure 3, their average performance is identical for all graphs and both data sets. On an instance by instance comparison, the two ratios are equal for 98% of all instances we evaluate. The largest difference between performance ratios we observe in our experiments are seven instances with difference 0.33 and one with difference 0.7.

5.4 Variance in Performance

We compare the average performance of an algorithm to the worst performance among the best 25% of performances as well as the worst performance among the best 75% of all

show that the variance increases with the average performance ratio of an uncertainty graph and it is greater for the deterministic algorithms than for RANDOM. As the variance is equal for CYCLE and CUT, we only display the graph for CYCLE. For almost every graph individually, the variance between the different instances is very small.

5.5 Worst-Case Instances

Both deterministic algorithms have a competitive ratio of 2. In our experiments, this worst-case ratio is attained for some instances for which the optimal query number is at most 10. For the telecommunication data the worst-case is attained only on the pathological

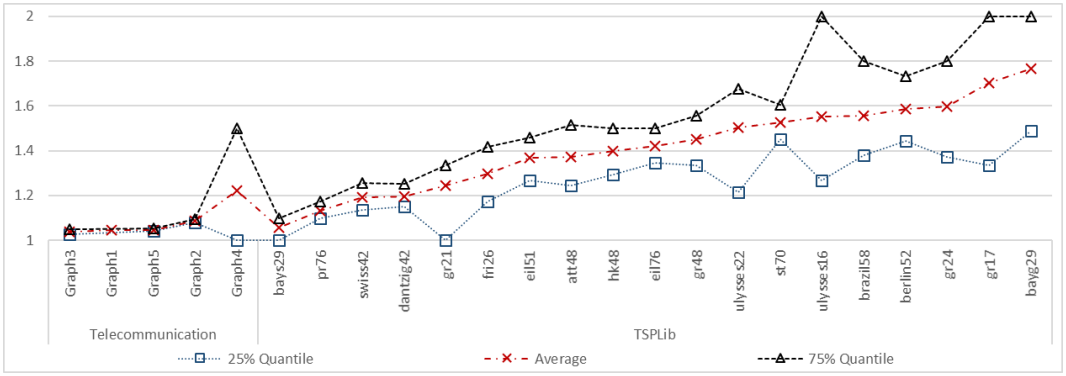

Figure 4 We show the variance of the average performance of CYCLE for each uncertainty graph by displaying the 25% Quantile, the average, and the 75% Quantile of the algorithm performance.

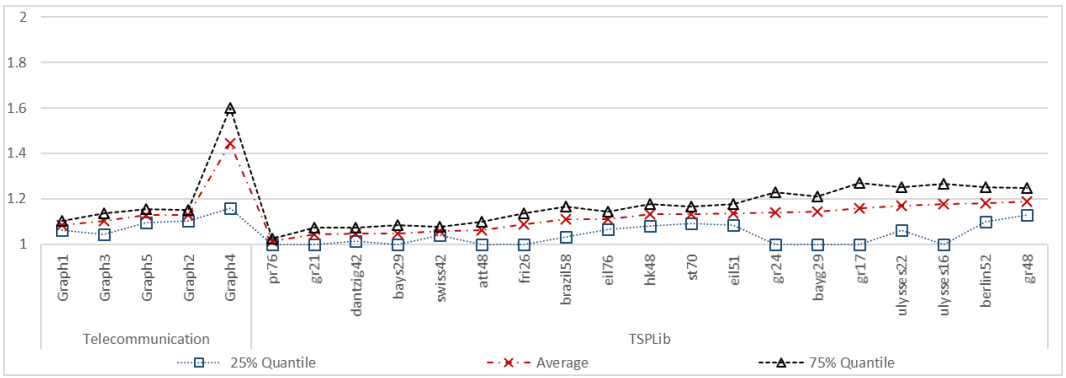

Figure 5 Variance of the average performance of RANDOM for each uncertainty graph with the 25% Quantile, the average, and the 75% Quantile of the algorithm performance.

example of Graph 4 consisting of a single cycle. However, the 7 smallest of the 19 graphs of the TSPLib data showcase instances with performance ratio 2. There are graphs, for which more than half of the instances showcase a worst-case ratio 2, but for others it is only a small percentage. The number of cases of ratio 2 roughly decreases with the number of edges in the graph. This is not symmetric to the case of performance ratio 1. There are more instances and more graphs for which there are instances which the algorithms solve optimally.

5.6 Comparing the Distributions

PREPROCESSING $(\prec_L, \prec_U)$ solves all telecommunication data instances with extreme distribution. We prove this theoretically in Section 3 and observe that this is due to the interval structure and not the distribution. In general, the share of instances solved by the preprocessing significantly increases from uniform to extreme distribution. For the TSPLib data the average share increases from 0.014 to 0.15 and for the telecommunication data it is 0.33 for the uniform distribution and 1 for the extreme one.

Additionally, we observe that the absolute number of queries almost always decreases, when changing from uniform to extreme distribution for all telecommunication instances and all algorithms. However, for the TSPLib data the behavior varies and for each graph there are instances where the uniform distribution has a smaller query number and others where the extreme distribution has a smaller query number.

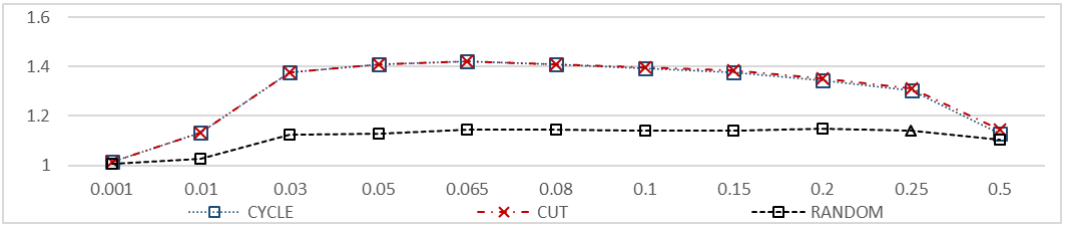

Figure 6 Average performance of the three algorithms for the TSPLib data for different values of the parameter d = *interval size over exact edge weight*.

5.7 Interval Size

To create the TSPLib data, we experimented with different interval sizes. Figure 6 shows that the algorithms' performance changes greatly with the chosen ratio d of interval size over edge weight. For very large parameter d, almost all intervals overlap and their edges must be queried. For very small d, however, only few intervals overlap and almost no queries are required. This is true for any algorithm, and thus, it explains why Cycle, Cut and Random have an average competitive ratio near to 1 for very small and very large d.

5.8 Running Time

We run our experiments on a Linux system with an AMD Phenom II X6 1090T (3.2 GHz) processor and 8 GB RAM. Together, the three algorithms take about 1200 milliseconds to compute. The preprocessing dominates the run time with a duration of 770 milliseconds on average. On a one-by-one comparison Cycle and Random have similar average run times of around 20 milliseconds, but Cut's average run time is around 350 milliseconds. As expected, the run time increases with the graph size. In total, our data set of 400 instances up to a size of 70 vertices or 3000 edges can be generated and solved in roughly four hours. As we did not optimize the implementation in terms of run time, we expect that also larger instances can be solved in reasonable time.

References

1 R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

2 A. Ben-Tal, L. El Ghaoui, and A. S. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, 2009.

3 J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research. Springer, 1997.

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

5 T. Erlebach and M. Hoffmann. Minimum spanning tree verification under uncertainty. In *Proceedings of WG*, pages 164–175, 2014. doi:10.1007/978-3-319-12340-0_14.

6 T. Erlebach and M. Hoffmann. Query-competitive algorithms for computing with uncertainty. *Bulletin of the EATCS*, Vol. 116, 2015.

7 T. Erlebach, M. Hoffmann, and F. Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Th. Comp. Sci.*, 613:51–64, 2016. doi:10.1016/j.tcs.2015.11.025.

8 T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihalák, and R. Raman. Computing minimum spanning trees with uncertainty. In *Proceedings of STACS*, pages 277–288, 2008. doi:10.4230/LIPIcs.STACS.2008.1358.

9 T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62:1–18, 2007. doi:10.1016/j.jalgor.2004.07.005.

10 T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32:538–547, 2003. doi:10.1137/S0097539701395668.

11 J. Focke. Comparative analysis of algorithms for minimum spanning tree under uncertainty. Master's thesis, Technische Universität Berlin, 2017.

12 M. Goerigk, M. Gupta, J. Ide, A. Schöbel, and S. Sen. The robust knapsack problem with queries. *Computers & OR*, 55:12–22, 2015. doi:10.1016/j.cor.2014.09.010.

13 M. Gupta, Y. Sabharwal, and S. Sen. The update complexity of selection and related problems. *Theory Comput. Syst.*, 59(1):112–132, 2016. doi:10.1007/s00224-015-9664-y.

14 S. Kahan. A model for data in motion. In *Proceedings of STOC*, pages 267–277, 1991. doi:10.1145/103418.103449.

15 N. Megow, J. Meißner, and M. Skutella. Randomization helps computing a minimum spanning tree under uncertainty. *Journal on Scientific Computing*, 2017.

16 G. Reinelt. TSPLIB – A traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991. doi:10.1287/ijoc.3.4.376.