Minimum Spanning Tree under Explorable Uncertainty in Theory and Experiments

JACOB FOCKE, University of Oxford, UK
NICOLE MEGOW, University of Bremen, Germany
JULIE MEIßNER, Technical University of Berlin, Germany

We consider the minimum spanning tree (MST) problem in an uncertainty model where interval edge weights can be explored to obtain the exact weight. The task is to find an MST by querying the minimum number of edges. This problem has received quite some attention from the algorithms theory community. In this article, 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 to optimality 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.

CCS Concepts: • **Mathematics of computing → Combinatorial optimization**; • **Theory of computation → Graph algorithms analysis**;

Additional Key Words and Phrases: Minimum spanning tree, uncertainty intervals, explorable uncertainty, query, competitive ratio, experiments

ACM Reference format:
Jacob Focke, Nicole Megow, and Julie Meißner. 2020. Minimum Spanning Tree under Explorable Uncertainty in Theory and Experiments. *J. Exp. Algorithmics* 25, 1, Article 1.14 (November 2020), 20 pages.
https://doi.org/10.1145/3422371

1 INTRODUCTION

Uncertain data is a common issue in many real-world optimization problems. While it is clear that uncertain data cannot be completely avoided, improved or exact data can often be obtained at an additional cost. Classical approaches to optimization under uncertainty such as robust, stochastic,

An extended abstract of this work was published at the Symposium of Experimental Algorithms (SEA) 2017 [11].
This research was carried out in the framework of MATHEON supported by Einstein Foundation Berlin and the German Science Foundation (DFG) under Contract No. ME 3825/1. Jacob Focke has received funding from the Engineering and Physical Sciences Research Council (Grant No. EP/M508111/1).
Authors' addresses: J. Focke, Department of Computer Science, University of Oxford, 15 Parks Rd, Oxford OX1 3QD, UK; email: jacob.focke@cs.ox.ac.uk; N. Megow, Department of Mathematics and Computer Science, University of Bremen, Bibliothekstraße 5, 28359 Bremen, Germany; email: nicole.megow@uni-bremen.de; J. Meißner, Institute of Mathematics, Technical University of Berlin, Str. des 17. Juni 136, 10587 Berlin, Germany; email: julie@schasler.com.

1084-6654/2020/11-ART1.14 $15.00
https://doi.org/10.1145/3422371

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. The question is how to balance 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.

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 article, 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 is contained. 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 made by an algorithm and the minimum number of queries required to verify an optimal solution for the underlying MST problem. 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 article, 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] as a procedure to determine and query edges that every algorithm has to query to obtain a solution. We develop an implementation thereof that 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. In this setting a number of chosen facility locations 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].

1.1 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 kth smallest value in a set of uncertainty intervals [10, 13, 14] and classical combinatorial optimization problems, such as shortest path [9], finding the median [10], the MST problem [8, 15], 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].

1.2 Our Contribution

We compare the three mentioned algorithms for MST under uncertainty theoretically as well as via practical experiments. We then showcase similarities and differences between theoretical and practical observations.

We also define an implementation of a preprocessing that identifies and queries a *maximal* number of edges that must be queried by any algorithm including the optimal one. In that sense, we find a best-possible preprocessing, and we also characterize a class of instances that is solved completely by the preprocessing. Interestingly, we observe exactly this structure in one of the practical data sets.

Our experiments show that on average the competitive ratio as well as the total number of queries in comparison to the size of the graph are small. The comparison of theory and practice in Sections 4 and 6 shows that the competitive ratio of the examined algorithms is far from the worst-case. We also observe that the influence of different realizations on the size of an optimal solution is substantially smaller than theoretically possible. While in theory 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, theoretically, there exist instances on which the two deterministic algorithms perform better than the randomized one. Surprisingly, we observe this behavior for a part of the telecommunication data. For the TSPLib data the contrary is true and the randomized algorithm has a significantly smaller competitive ratio. Conducting the first experiments, we point out that the implementation hurdle is small and our runtimes are reasonably small, even though we did not optimize on it.

1.3 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*, $A_e = [L_e, U_e]$ with $L_e = U_e$, 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 are not 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 an MST in the uncertainty graph G for an a priori unknown realization $\mathcal{R}$ of edge weights. An algorithm 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 using a set of queries that is as small as possible. An algorithm can include edges in this query set one at a time and use the information about the realization of added edges to determine the next query but it must not delete edges from the query set once added. The final set of queries $Q \subseteq E$ is *feasible*, if there is a spanning tree that 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 the problem of finding a feasible query set of minimum size *MST under uncertainty*.

We evaluate algorithms by standard competitive analysis. An algorithm is *c-competitive* if, for any uncertainty graph G and 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 PREPROCESSING

Generally, preprocessing aims at simplifying input instances. In our setting, this means that we identify and query edges that must be queried by *any* algorithm including the optimal one. To reduce the input instance to its core regarding uncertainty, we want to query as many such edges as possible before starting the actual algorithm. It shall be mentioned that one of our algorithms requires this preprocessing whereas the other two do not. For a reasonable comparison, we let all algorithms start from the same preprocessed instance.

A preprocessing routine has been proposed in [15] in the context of designing a randomized algorithm. It was used to guarantee a certain structural property of edges appearing in cycles in the algorithm. In this section, we discuss and refine this preprocessing such that it is guaranteed to query a maximal number of edges.

There is a characterization of (a subset of) such edges that relies on the following definitions. 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 ε-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 ε-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. Let $(\prec_\ell, \prec_u)$ be a pair of total orderings of the edges in E. We say $(\prec_\ell, \prec_u)$ is *feasible* if a minimum spanning tree with respect to $\prec_\ell$ is a lower limit tree and with respect to $\prec_u$ it is an upper limit tree, respectively.

Theorem 2.1 ([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 Algorithm 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 feasible pair of total orderings $\prec_L, \prec_U$ on the edges, and we prove that for this pair of orderings Preprocessing $(\prec_L, \prec_U)$ maximizes the total number of queries.

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.
1: $Q \leftarrow \emptyset$.
2: Determine T_ℓ and T_u according to $\prec_\ell$ and $\prec_u$, respectively, using Prim's algorithm [1].
3: **while** $T_\ell \setminus T_u$ contains a non-trivial edge **do**
4: Query all non-trivial edges in $T_\ell \setminus T_u$, and add them to Q.
5: Update T_ℓ and T_u.
6: **return** The query set Q and the two trees T_ℓ, T_u.

Definition 2.1 (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:

(1) e_i is trivial and e_j is non-trivial,
(2) $U_{e_i} > U_{e_j}$ and e_j is non-trivial,
(3) $U_{e_i} = U_{e_j}$ 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:

(1) e_j is trivial and e_i is non-trivial,
(2) $L_{e_i} > L_{e_j}$ and e_i is non-trivial,
(3) $L_{e_i} = L_{e_j}$ 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 that 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 2.1. *Let $(V, E, \mathcal{A})$ be an uncertainty graph and $(\prec_\ell, \prec_u)$ be a feasible pair of total orderings of E with corresponding lower and upper limit trees T_ℓ and T_u. When running PREPROCESSING $(\prec_L, \prec_U)$, 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 2.2 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 2.2. *Let $(V, E, \mathcal{A})$ be an uncertainty graph and $(\prec_\ell, \prec_u)$ be a feasible pair of total orderings of E with corresponding lower and upper limit trees T_ℓ and T_u. 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.

Case 1: $e \in T_L$. Then e is also in T_U. Consider the cycle in $T_u \cup \{e\}$. At least two edges in this cycle have to be in the cut $T_U \setminus \{e\}$, one of which is e itself. Thus, let h be an edge distinct from e, which

is both in the cut $T_U \setminus \{e\}$ 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 $L_e \geq L_h$ by our definition of $\prec_U$. Furthermore, since h is in the cycle, $U_h = U_e$, and e is non-trivial, h has to be non-trivial (for $\prec_u$ to be a feasible total ordering). Since h is non-trivial and $h \notin T_U$, we have $h \notin T_L$ (since $T_L \setminus T_U$ contains only trivial edges). Now consider the cycle in $T_L \cup \{h\}$ and the cut $T_U \setminus \{e\}$, which contains h. There has to exist at least one other edge g, which is distinct from h and both in the cycle in $T_L \cup \{h\}$ and the cut $T_U \setminus \{e\}$. We split the analysis depending on whether or not $g = e$.

- If $g = e$, then e is in the cycle in $T_L \cup \{h\}$ and therefore $e \prec_L h$. This means that $L_e \leq L_h$ and consequently $L_e = L_h$. Hence, e and h have identical uncertainty intervals. Since h is in the cut $T_U \setminus \{e\}$ it holds that $e \prec_U h$. However, if both $e \prec_L h$ and $e \prec_U h$ hold, then e and h cannot have identical uncertainty intervals, a contradiction.
- If $g \neq e$, then g is in T_L and not in T_U and is therefore trivial by the assumption that $T_L \setminus T_U$ contains only trivial edges. Furthermore, since it is in the cut it holds that $e \prec_U g$, and since it is in the cycle, we have $g \prec_L h$. Together with the observations about the bounds of e and h we made above, this means 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.

Case 2: $e \notin T_L \cup T_U$. In this case there is an edge $h \neq e$, 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 is 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 $\prec_L$. Since h is non-trivial and $h \in T_L$ edge h has to be in T_U (as $T_L \setminus T_U$ contains only trivial edges). As in the previous case there must be an edge g distinct from h that is both in the cycle $T_L \cup \{e\}$ and in the cut $T_U \setminus \{h\}$. Thus, analogous to the previous case:

- If $g = e$, then e and h have identical uncertainty intervals, which contradicts that both $h \prec_L e$ and $h \prec_U e$ hold.
- If $g \neq e$, then g is in $T_L \setminus T_U$ and therefore trivial. In addition it holds that $h \prec_U g$ as well as $g \prec_L e$. Together with the observations about the bounds of e and h this means 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.

Case 3: $e \in T_U \setminus T_L$. Then there is an edge h in the cut $T_U \setminus \{e\}$ and in the cycle $T_L \cup \{e\}$. This implies $h \in T_L \setminus T_U$ and therefore that h is trivial. Additionally, we have $e \prec_U h$ and $h \prec_L e$, 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 2.2. *The procedure Preprocessing $(\prec_L, \prec_U)$ queries the union of the edges queried by Preprocessing $(\prec_\ell, \prec_u)$ over all feasible pairs of orderings $(\prec_\ell, \prec_u)$. Thus, it queries the maximum number of edges characterized by Theorem 2.1.*

Proof. We show by induction over the number of iterations that edges that are queried in Preprocessing $(\prec_\ell, \prec_u)$ are also queried in Preprocessing $(\prec_L, \prec_U)$. By Lemmas 2.1 and 2.2, 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 that are queried by Preprocessing $(\prec_L, \prec_U)$. By Lemma 2.1 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 2.2. □

3 ALGORITHMS

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 designed for the more general problem of computing a minimum weight matroid basis in the uncertainty exploration model. The algorithm Cycle in the context of MST under uncertainty can be interpreted as a variant of Ured without repeated restarts. Thus, we consider in this work the implementation of Cycle as a simplified variant of Ured. The randomized algorithm Random with competitive ratio 1.707 and the preprocessing discussed in Section 2 were also introduced in [15]. The best-known lower bound for randomized algorithms is 1.5.

In this work, we consider the algorithms Cycle, Cut, and Random, which we describe below. Despite their original definition, we apply the preprocessing procedure from Section 2 to *all* three algorithms. To describe the algorithms, we use the definition of lower and upper limit trees, T_L and T_U, in Definition 2.1. Moreover, we say that an edge is *always maximal* in a cycle, if it has the largest upper limit and it either has a trivial uncertainty interval or no other upper limit exceeds its lower limit. Symmetrically, an edge is *always minimal* in a cut, if it has the smallest lower limit and it either has a trivial uncertainty interval or no other lower limit is less than its upper limit. For a cycle C with an edge f, we call the set of all edges $e \in C$ with $U_e > L_f$ the *neighborhood* $X(f)$ of edge f.

3.1 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. Edges are added in order of increasing lower limit where edges with smaller upper limit are preferred in the case of ties. 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. We give the pseudocode for Cycle in Algorithm 2.

ALGORITHM 2: Cycle

Input: An uncertainty graph G with $G = (V, E, \mathcal{A})$ and a realization $\mathcal{R}$.
Output: A feasible query set $Q \subseteq E$.

```
Execute the algorithm Preprocessing (≺_L, ≺_U), which returns T_L, T_U, Q.
Let f_1, ..., f_{m-n+1} be the edges from E \ T_L ordered by increasing lower limit.
for i = 1, ..., m − n + 1 do
    Add f_i to T_L and let C be the cycle closed by f_i.
    while ∄ always maximal edge in C do
        Choose f ∈ C s.t. U_f = max{U_e | e ∈ C}.
        Choose g ∈ C \ {f}, with largest upper limit U_g > L_f.
        Query g and f if they are non-trivial and add them to Q.
    Delete an always maximal edge from T_L.
return Q.
```

3.2 Deterministic Algorithm Cut

Cut is the dual algorithm to Cycle, which is defined by matroid duality. It uses that the minimum-weight edge in a cut is in any MST. Like in the previous algorithm, Cut starts with a candidate

MST, but iteratively considers the edges of this tree. Edges are considered in order of decreasing upper limit where edges with larger lower limit are preferred in the case of ties. Deleting an edge from the tree 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. Pseudocode for Cut is given in Algorithm 3.

ALGORITHM 3: Cut

Input: An uncertainty graph $G = (V, E, \mathcal{A})$ and a realization $\mathcal{R}$.
Output: A feasible query set $Q \subseteq E$.

```
Execute the algorithm Preprocessing (≺_L, ≺_U), which returns T_L, T_U, Q.
Let g_1, ..., g_{n-1} be the edges from T_U ordered by decreasing upper limit.
for i = 1, ..., n − 1 do
    Delete g_i from T_U and let S be the cut that is created.
    while ∄ always minimal edge in S do
        Choose g ∈ S with smallest lower limit L_g = min{L_e | e ∈ S}.
        Choose f ∈ S \ {g}, s.t. L_f < U_g.
        Query g and f if they are non-trivial, and add them to Q.
    Add an always minimal edge to T_U.
return Q.
```

3.3 Randomized Algorithm

The randomized algorithm Random uses the following structural property: In each cycle appearing in the algorithm, the edge with largest lower limit also has the largest upper limit. This is guaranteed by the preprocessing, which we discussed in detail in Section 2.

The algorithm Random has 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. Again edges are added to the tree in order of increasing lower limit where edges with smaller upper limit are preferred in the case of ties. 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 y_e of an edge e to a randomly chosen uniform threshold b. The corresponding pseudocode is given in Algorithm 4.

4 THEORETICAL ANALYSIS

4.1 Instances Solved and Instances Impaired by the Preprocessing

The preprocessing is a modification of the input instance and intuitively it simplifies the instance by removing uncertainty. We note, however, that in theory it can lead to a worse algorithm performance for specific input and algorithm. For instance, consider executing Cycle on the uncertainty graph shown in Figure 1 on the left. In the preprocessing step (Algorithm Preprocessing $(\prec_L, \prec_U)$) the edge e_3 is queried. Since e_3 is queried its updated uncertainty interval is then [3]. Depending on an unspecified tiebreaking rule Cycle might then query the pair of edges e_1 and e_2, querying a total of 3 edges. Now assume we execute Cycle without the preprocessing step. In its first iteration

Fig. 1. Preprocessing can impair the performance of Cycle (graph on the left). However, it can also improve its performance from worst-case to optimal (graph on the right). Edge labels: $(L_e, U_e) \mid \mathbf{w_e}$.

it would query e_2 and e_3, which is a feasible query set. In this example, the preprocessing might impair the competitive ratio of Cycle by a factor of 1.5.

Nevertheless, in our experiments, the preprocessing generally improves the performance ratio of the examined algorithms; see Section 6.2. One class of the data sets we use is even solved exactly by the preprocessing alone. In Proposition 4.1, we generalize this observation and characterize a family of uncertainty graphs that can be solved completely by the proposed preprocessing. This family also offers examples that demonstrate that the preprocessing can significantly improve algorithm performance. Consider executing Cycle on the uncertainty graph depicted in Figure 1 on the right. The preprocessing queries edge e_3 and finds an optimal set of queries, whereas Cycle without the preprocessing step would pair e_3 with either e_1 or e_2 and query 2 edges. This shows that the preprocessing can improve the performance of Cycle from its worst-case competitive ratio of 2 to finding an optimal solution instead. (One can find similar examples for the algorithm Cut.)

Proposition 4.1. *Preprocessing* $(\prec_L, \prec_U)$ *finds an optimal solution for uncertainty graphs in which every cycle contains only edges with identical lower limit or only edges with identical upper limit.*

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

ALGORITHM 4: Random

Input: An uncertainty graph $G = (V, E, \mathcal{A})$ and a realization $\mathcal{R}$.
Output: A feasible query set $Q \subseteq E$.

Execute the algorithm Preprocessing $(\prec_L, \prec_U)$, which returns T_L, T_U, Q.
Let $f_1, \ldots, f_{m-n+1}$ be the edges from $E \setminus T_L$ ordered by increasing lower limit.
Initialize $y_e = 0$, $\forall e \in E$, and choose b uniformly at random in $[0, 1]$.
for $i = 1$ to $m - n + 1$ **do**
  Add edge f_i to T_L and let C be the unique cycle closed by f_i.
  Let the neighbor set $X(f_i)$ be the set of edges $g \in T_L \cap C$ with $U_g > L_{f_i}$.
  if $X(f_i)$ is not empty **then**
    Maximize the threshold $t(f_i) \leq 1$ s.t. $\sum_{e \in X(f_i)} \max\{0, t(f_i) - y_e\} \leq 1 + 1/\sqrt{(2)}$.
    Increase edge potentials $y_e := \max\{t(f_i), y_e\}$ for all edges $e \in X(f_i)$.
    if $t(f_i) < b$ **then**
      Query f_i and add it to the query set Q.
    else
      Query all edges in $X(f_i)$ and add them to Q.
  while $\nexists$ always maximal edge in C **do**
    Query edge $e \in C \setminus Q$ with maximum U_e and add it to Q.
  Delete an always maximal edge from T_L.
return Q.

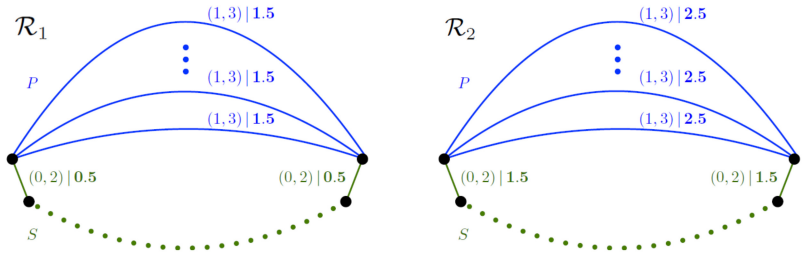


Fig. 2. Different realizations for the class of uncertainty graphs SP lead to different extremes in the behavior of Cycle and Cut. Edge labels: $(L_e, U_e)\,|\,\mathbf{w_e}$.

unclear which edge has the largest weight and for which 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, then 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 in C with largest upper limit. One of these edges, say f, is not in T_U. Then $f \notin T_L$, and since at most one edge in C is not in T_L, we have $e \in T_L$. 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 2.2 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. □

4.2 Comparing the Deterministic Algorithms

We describe a setting for which Cycle is near-optimal but Cut has competitive ratio of 2. The idea is to construct an instance that is solved by querying the edges of a single cycle C. Then, 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.

Let SP be an uncertainty graph that consists of a path of edges S and a set of parallel edges P, each of which closes a cycle with S. We give the uncertainty intervals and two realizations $\mathcal{R}_1$ and $\mathcal{R}_2$ in Figure 2.

Consider $\mathcal{R}_1$. Observe that any feasible query set must contain S as otherwise there remains a cycle on which the maximal edge is not known. Notice further, that it is sufficient to query the set S and, thus, S is the unique optimal query set. Now consider the algorithm Cycle. The first cycle closed by 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, however, considers cuts of the form $P + \{s\}$ with a non-queried edge $s \in S$. 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 4.1. *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.

4.3 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 4.2. *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 4.3. *There exists a family of uncertainty graphs, for which Cycle and Cut perform optimally, whereas Random asymptotically shows its worst-case behavior.*

4.4 Variation in the Size of an Optimal Solution

We investigate the influence of different realizations for a fixed input instance on the size of the optimal solution Opt. We give an example instance in which small perturbations in the realization significantly change the value of Opt.

Consider a cycle C of length m consisting of an edge f with uncertainty interval $(1,4)$ and $m-1$ identical edges $\mathcal{G} = \{g_1, \ldots, g_{m-1}\}$ with uncertainty interval $(0,3)$ and weight 2. If we set the weight of f to be 3, then it suffices to query f and Opt = 1. However, if the weight of f is 2, then all edges in C have to be queried and Opt = m. Interestingly, we do not observe this large variance in our experiments, see Section 6.1.

Observation 4.4. *For a fixed uncertainty graph, Opt can vary greatly even for minor changes of the underlying realization. Changing the weight of only one edge may lead to drastic changes in the optimal solution, from Opt = 1 to Opt = m.*

5 EXPERIMENTAL DATA

First, note that there is an inherent difficulty in running 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 are given five different graphs of varying size with up to 1,000 nodes. These graphs are relatively sparse as is demonstrated in Table 1. 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 equal to the construction cost per meter for installing the cable times the length of the connection. The construction cost is at least that of setting up a connection in a field and at most that of the installation underneath a paved street. Having determined the uncertainty intervals in that way, we have to generate the remainder of the instance by drawing the exact edge weights *uniformly* distributed in the corresponding intervals. In this uncertainty setting, exploring

Table 1. Basic Properties of Graphs in the Telecommunication Data Set

Graph	Graph 1	Graph 2	Graph 3	Graph 4	Graph 5
# Vertices	10	255	384	1002	896
# Edges	10	316	564	1033	1101
Density	0.2222	0.0098	0.0077	0.0021	0.0027

the exact weight of an edge represents the time or cost investment it takes to identify the terrain of a particular connection. We call the second data set existence data. This setting assumes that the terrain of the connection is known, but it is uncertain if existing cable 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 19 graphs for the symmetric traveling salesman problem TSP from the library TSPLib, which have between 16 and 76 nodes. They are usually used for TSP computations, but we compute their minimum spanning trees. Using this real-world data, we are given exact edge weights and need to generate the 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 instances for which preferably the competitive ratio of examined algorithms is large. 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 of d between 0.001 and 0.5 for a total of 380 uncertainty graphs per value of d (we draw 10 sets of uncertainty intervals for each of the 19 graphs and for each type of distribution); see Figure 13. As we are interested in a worst-case behavior, we choose $d = 0.065$ for our experiments as the investigated algorithms have a rather large competitive ratio for this choice of d.

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 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.

6 EXPERIMENTAL ANALYSIS

For the detailed analysis, we draw 100 uncertainty intervals/realizations for each graph in a data set, which yields 4,800 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 runtime of the three algorithms as well as that of the preprocessing. As a side note, we additionally compute the average number of edges on a cycle closed by Random and the average number of edges on an algorithm cycle that have an uncertainty interval overlapping the one of the edges with largest upper limit. For the latter two parameters, however, we could not find a relation to the

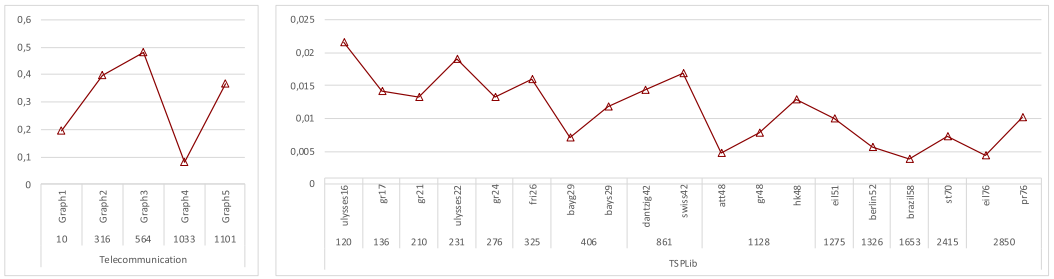


Fig. 3. Size of the optimal solution Opt divided by the number of edges m on the y-axis and the uncertainty graphs with uniformly distributed weights sorted by data set and increasing number of edges on the x-axis.

algorithm's performance. We summarize our experimental results in the following subsections. We make our program code, the complete input and output data, and further analysis available.[1]

6.1 The Optimal Solution

To compute the number of queries in an optimal solution Opt, we implement the algorithm proposed in [5]. 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 3 for results on instances with weights following a uniform distribution. There are instances among the telecommunication data for which the ratio Opt/m is as large as 0.48, that is, even in an optimal solution roughly half of the edges in the graph must be queried. Other instances, however, have a very small ratio Opt/m. Among this small number of instances the parameter behavior seems arbitrary. For the TSPLib data the ratio Opt/m is significantly smaller, between 0.004 and 0.022, and it decreases when m increases. Our theoretical analysis in Section 4.4 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.

6.2 The Preprocessing

In this section, we first investigate how many of the queries in a solution are made during the preprocessing step. Afterwards, we make a comparison between the performance of the deterministic algorithms with and without the preprocessing.

The preprocessing is very powerful on certain real-world data. Preprocessing $(\prec_L, \prec_U)$ solves all telecommunication instances following a two-point distribution to optimality. We prove this theoretically in Proposition 4.1 and observe that this is due to the interval structure and not the distribution.

Figure 4 shows the percentage of the optimal number of queries that the preprocessing identifies. Also among the TSPLib data, a fair percentage (15% on average) of the optimal number of queries is found. Clearly, the graph structure plays an important role: While the preprocessing finds up to 40% of the queries for some graphs, it does not reduce the uncertainty at all for other graphs.

Generally, the preprocessing seems much stronger on instances with weights following a two-point distribution as opposed to those with uniform distributions. This can be clearly seen when comparing Figure 4 with Figure 5. While telecommunication instances with two-point distributed weights are solved completely, under uniform distributions, the preprocessing queries only 33% of

[1] http://www.coga.tu-berlin.de/fileadmin/i26/coga/MSTData.zip.

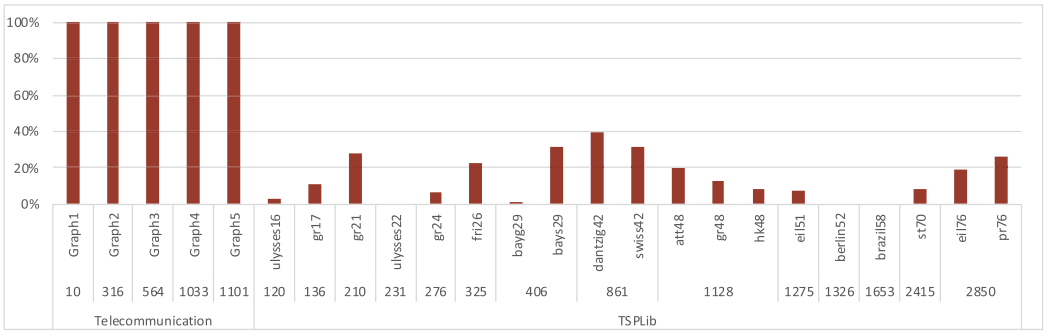


Fig. 4. Percentage of the optimal number of queries found by the preprocessing for the two-point distribution.

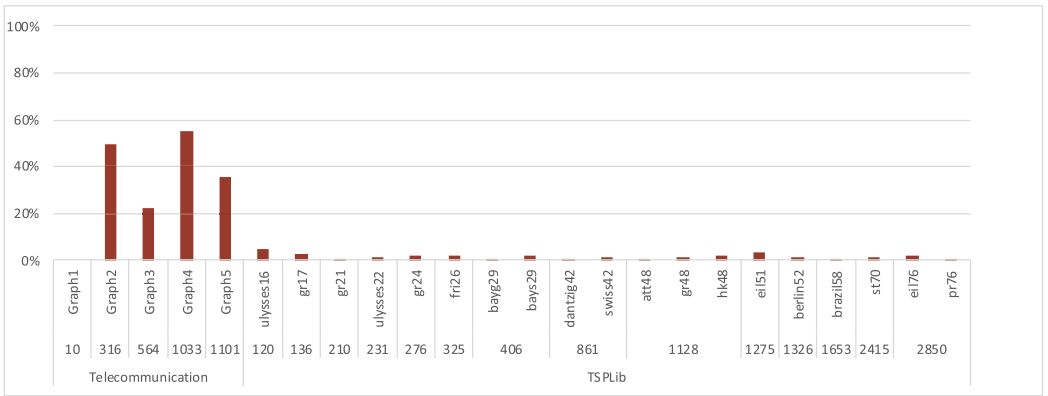


Fig. 5. Percentage of the optimal number of queries found by the preprocessing for the uniform distribution.

an optimal solution, on average, with a maximum at 55% (*Graph4* with 1033 edges) and a minimum at 0% (*Graph1* with 10 edges).

For the TSPLib data the behavior varies and there are graphs for which the preprocessing actually queries a larger percentage of the optimal number of queries in the case of the uniform distribution compared to the same graph in the two-point distribution setting. An example for this behavior is the graph *ulysses16* for which the preprocessing queries 4.5% of an optimal solution in the uniform case but only 3% in the case of two-point distributed edge weights. However, on average the preprocessing for uniformly distributed instances appears again much weaker than for two-point distributed instances. For uniform distributions the preprocessing finds on average 1.6% and at best 4.5% of the optimal number of queries, whereas for two-point distributions the average is 15% and the best 40%.

As mentioned before, the preprocessing is a necessary part of the algorithm Random. Since the preprocessing is essentially a modification of the input instance it is sensible to investigate the performance of all algorithms on the same preprocessed instance. In Section 4.1, we have pointed out that in theory the preprocessing can impair the performance of the deterministic algorithms. In Figure 6, we show that in our experiments we do not observe this behavior. (We only display the analysis for Cycle, since it is basically identical to that of Cut.) To the contrary, the experiments match our intuition that a precursory reduction of the uncertainty within the input instance leads to an overall improvement in the performance of the algorithms Cycle and Cut. In detail, for each

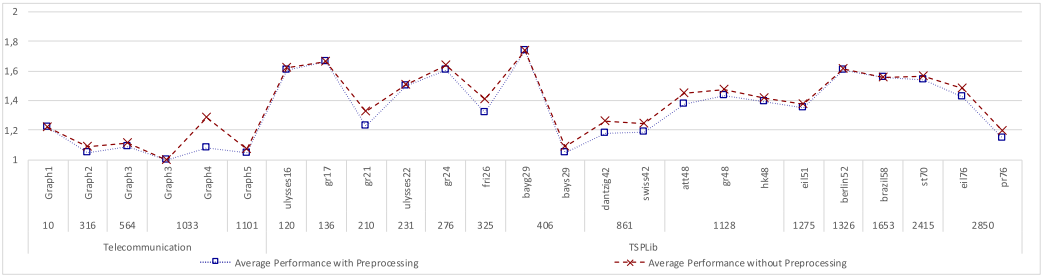


Fig. 6. The competitive ratio of the algorithm Cycle as defined in Algorithm 2 in comparison to its competitive ratio when omitting the preprocessing step. For each graph, we consider the average over all instances including both instances with uniform and instances with two-point edge weight distribution.

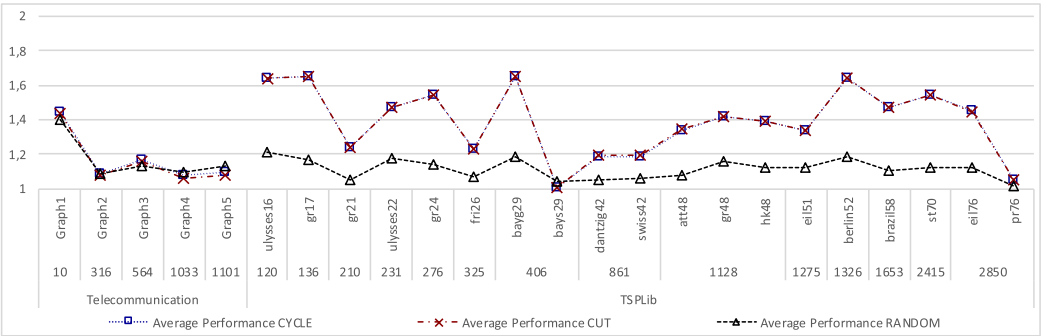


Fig. 7. Average performance, i.e., the ratio of the algorithm query set size over the optimal query set size, for the three algorithms on instances with uniformly distributed weights.

graph in both data sets the competitive ratio of Cycle including the preprocessing never exceeds the competitive ratio of Cycle without the preprocessing. In most instances the preprocessing leads to a slight improvement in performance. The maximum improvement occurs for *Graph4* from the telecommunication data. For this graph Cycle has an average competitive ratio of 1.08 when the preprocessing is included, whereas without the preprocessing the competitive ratio is 1.28. The overall average competitive ratio over all graphs is 1.35 with and 1.39 without the preprocessing.

6.3 Comparing Average Competitive Ratios

In this section, we compare the average competitive ratios achieved by the deterministic algorithms Cycle and Cut with the performance of the randomized algorithm Random. A more fine-grained analysis of the variance follows in Section 6.4.

For the telecommunication data, we consider only instances with uniformly distributed weights. Otherwise the competitive ratio is 1 for all three algorithms due to the power of the preprocessing; see Section 2 above. Interestingly, the competitive ratios of all three algorithms have roughly the same average when weights are uniformly distributed; see Figure 7. Averaging over all telecommunication instances, all algorithms have competitive ratio 1.17. For the TSPLib data under uniform distribution, the two deterministic algorithms have equal average competitive ratio 1.39, 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.

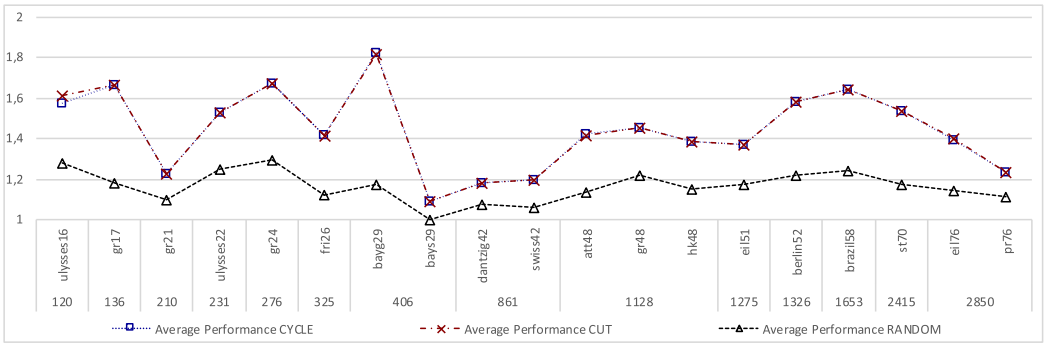


Fig. 8. Average competitive ratio of Cycle, Cut, and Random for instances with two-point distributed weights.

For the sake of completeness, we also evaluate the performance of the algorithms on TSPLib data with a two-point distribution; see Figure 8. Averaging over all instances, Cycle and Cut both have the same competitive ratio 1.44 and Random has the substantially better competitive ratio of 1.16.

All average competitive ratios are far from their theoretical worst-case guarantee, which is 2 for both deterministic algorithms and approximately 1.707 for Random. In Section 4.3, we showed that there are instances on which Random shows a worse performance than the deterministic algorithms. Surprisingly, we see this behavior confirmed for some graphs for uniformly distributed edge weights. In particular, for *Graph4* and *Graph5* from the telecommunication data set (both of which have more than 1,000 edges) the average competitive ratio of Random is 1.10 and 1.13, respectively. However, Cycle has competitive ratios of 1.08 and 1.09, respectively, and Cut is even better with ratios of 1.06 and 1.08, respectively. On the TSPLib data set under uniform distribution, we observe that Random performs substantially better than the deterministic algorithms for almost every graph, with *bays29* as the single exception. Finally, when looking at the TSPLib data graphs under two-point distributed edge weights, Random outperforms the deterministic algorithms considerably on every single graph.

In Section 4.2, 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 7, their average performance is identical for all graphs and both data sets. On an instance by instance comparison, the two ratios are equal for 92% of all instances we evaluate; see Figure 9. The largest difference between performance ratios we observe in our experiments are instances with a difference of 0.33 and one with 0.4.

6.4 Variance in Performance

As a measure of variance/standard deviation, 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 performances. Again, we consider only instances with uniformly distributed weights, as otherwise the competitive ratio is 1 for all three algorithms for the telecommunication data. The variance is very similar for Cycle and Cut, and therefore we only display the graph for Cycle. Figures 10 and 11 show that the variance is substantially larger for the deterministic algorithms than for Random.

Averaging over all telecommunication instances, the worst performance ratio of Cycle for the best-solved 25% of instances is 1.07 (25%-quantile), whereas for the best-solved 75% it is

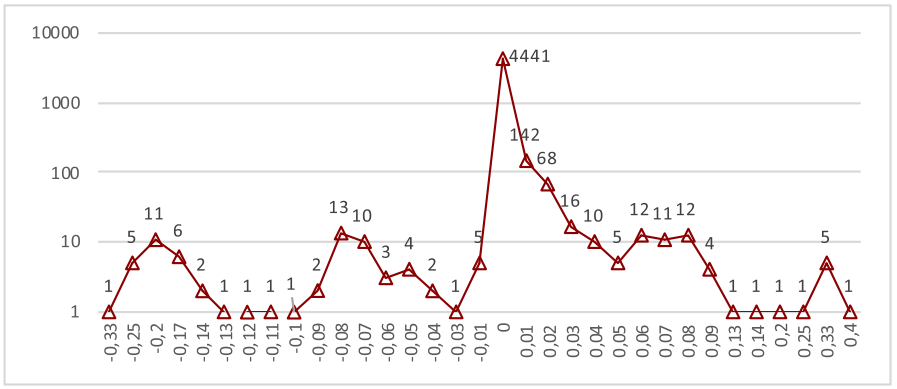


Fig. 9. Number of instances with performance difference between Cycle and Cut, i.e., competitive ratio of Cut minus that of Cycle, rounded to 1/100, displayed on a logarithmic scale (over all considered instances).

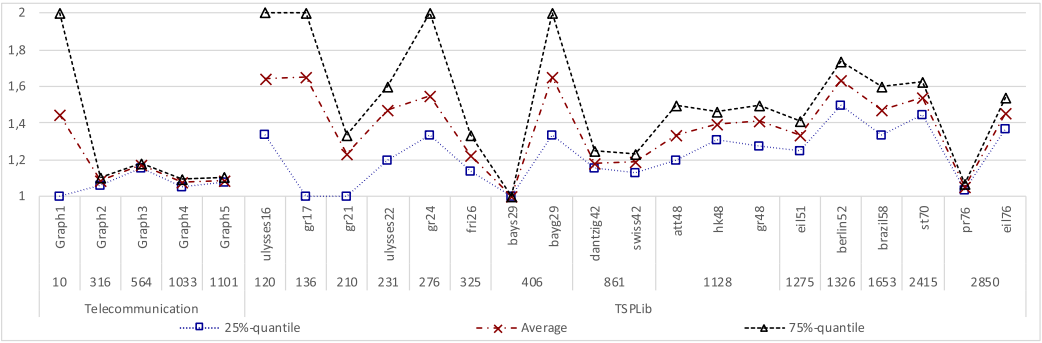


Fig. 10. We show the variance of the average performance of Cycle for each uncertainty graph with uniformly distributed weights, showing the 25%-quantile, the average, and the 75%-quantile of the algorithm performance.

1.29 (75%-quantile). However, the smallest graph in this graph class, *Graph1* with 10 edges, is an extreme case with the 25%-quantile at ratio 1 and the 75%-quantile at ratio 2. The variance of Cycle for the other graphs is negligible. Random has a similar average variance, with a 25%-quantile of 1.09 and a 75%-quantile of 1.2.

The general behavior is similar for the TSPLib data. Cycle's performance ratio at the 25% and 75%-quantiles are 1.23 and 1.54, respectively, where the variance observed for an individual graph is between ratio 1 and 2. The performance of Random is much better with a variance between 1.03 and 1.17 at the 25% and 75%-quantiles. The largest 75%-quantile observed for a graph is 1.33.

6.5 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 very small (at most 13). We display the data for the algorithm Cycle in Figure 12. For the telecommunication data the worst case is attained only on the pathological example of *Graph1* consisting of a single cycle. For the TSPLib data less than half of the graphs showcase any instances that lead to a competitive ratio of 2.

In our experiments there are more graphs for which Cycle achieves competitive ratio 1 on at least some instances than there are graphs for which it showcases its worst-case performance.

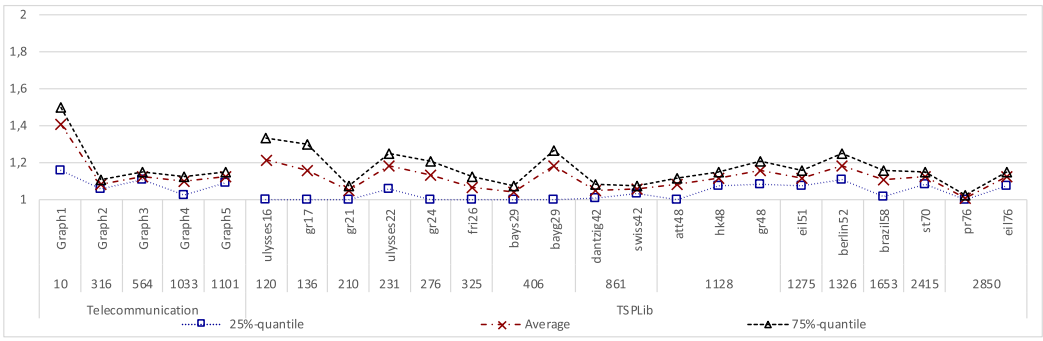
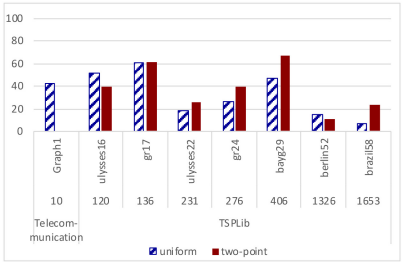
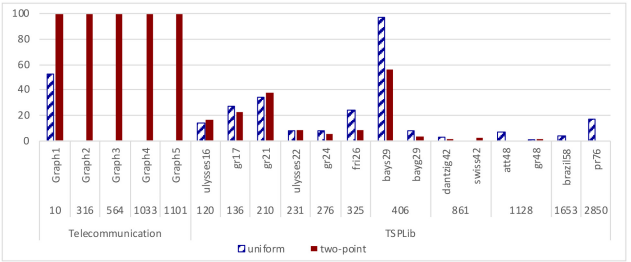


Fig. 11. Variance of the average performance of RANDOM for each uncertainty graph with uniformly distributed weights, showing the 25%-quantile, the average, and the 75%-quantile of the algorithm performance.



(a) Worst-possible competitive ratio 2.



(b) Best-possible competitive ratio 1.

Fig. 12. Number of instances for which CYCLE obtains best-possible and worst-possible competitive ratio.

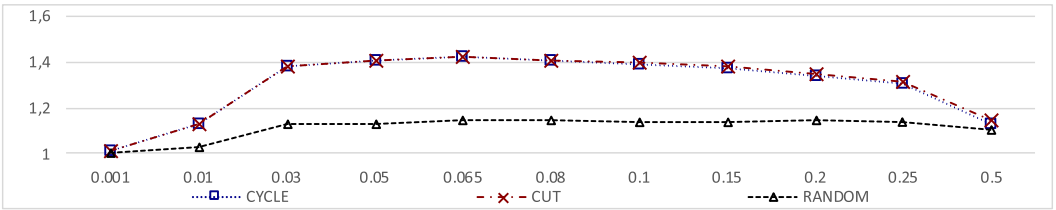


Fig. 13. Average performance of the three algorithms for the TSPLib data for different values of the parameter d = *interval size over exact edge weight.*

Overall, there are also more instances that are solved optimally by CYCLE than there are instances for which CYCLE has competitive ratio 2. Interestingly, in the case of uniform edge weight distributions, there are a lot of graphs from the TSPLib data for which CYCLE achieves the best possible ratio of 1—however, apart from the special Graph1 (which only contains a single cycle), it never obtains an optimal solution on the telecommunication data in this case.

6.6 Interval Size

To create the TSPLib data, we experimented with different interval sizes. Figure 13 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

Fig. 14. Visual comparison of the runtime of the algorithms in milliseconds.

any algorithm, and thus, it explains why Cycle, Cut, and Random have an average competitive ratio close to 1 for very small and very large d.

6.7 Runtime

We run our experiments on a Linux system with an AMD Phenom II X6 1090T (3.2 GHz) processor and 8 GB RAM. The total execution time of all three algorithms together is about 1, 200 ms, on average over all considered instances. The preprocessing dominates the runtime with a duration of 840 ms on average. On a one-by-one comparison Cycle and Random have similar average runtimes of around 20 ms, but Cut's average runtime is substantially larger with around 380 ms. The data is displayed in Figure 14. It is worth pointing out that the average runtime of the deterministic algorithms is almost identical, whether we execute them on the preprocessed or the original instances. In that sense the preprocessing is very runtime-inefficient. However, the preprocessing may detect already a large number of queries or even solve the entire instance, as discussed in Section 6.2. As expected, in general the runtime increases with the graph size. In total, our data set of 400 instances with graphs of size up to 70 nodes and 3,000 edges can be generated and solved in roughly four hours. As we did not optimize the implementation in terms of runtime, we expect that larger instances can also be solved in reasonable time.

7 FINAL REMARKS

In this article, we report on the first practical experiments on the MST problem in the uncertainty exploration model. We compare known algorithms and their theoretical and practical behavior on real-world data from a telecommunication provider as well as on data from the TSPLib.

As it is rather common in such experiments, the practical performance of the algorithms experienced on real-world data is rather far from the theoretical worst-case performance. As expected, Random generally shows a better performance than the deterministic algorithms. However, the amount of improvement varies greatly and depends on the input data: Random is significantly superior for the TSPLib data but not for the telecommunication data. Somewhat surprisingly, Random even showcases a slightly worse performance for some instances; cf. Sections 4.3 and 6.3. This means the usefulness of randomization depends on the considered data set.

Furthermore, we investigate a preprocessing from [15] to determine and query edges that every algorithm has to query. We develop an implementation thereof that maximally reduces the data uncertainty. This is theoretically interesting but also practically relevant, as it reduces the data uncertainty to a minimum before even starting an (arbitrary) algorithm. Interestingly, the preprocessing solves all our telecommunication data with two-point distributed edge weights to optimality.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.
[2] A. Ben-Tal, L. El Ghaoui, and A. S. Nemirovski. 2009. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press.

[3] J. R. Birge and F. Louveaux. 1997. *Introduction to Stochastic Programming*. Springer Series in Operations Research. Springer.
[4] A. Borodin and R. El-Yaniv. 1998. *Online Computation and Competitive Analysis*. Cambridge University Press.
[5] T. Erlebach and M. Hoffmann. 2014. Minimum spanning tree verification under uncertainty. In *Proceedings of the International Workshop on Graph-Theoretic Concepts in Computer Science (WG'14)*. 164–175. DOI: 10.1007/978-3-319-12340-0_14
[6] T. Erlebach and M. Hoffmann. 2015. Query-competitive algorithms for computing with uncertainty. *Bull. EATCS* 116 (2015).
[7] T. Erlebach, M. Hoffmann, and F. Kammer. 2016. Query-competitive algorithms for cheapest set problems under uncertainty. *Theor. Comput. Sci.* 613 (2016), 51–64. DOI: 10.1016/j.tcs.2015.11.025
[8] T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihalák, and R. Raman. 2008. Computing minimum spanning trees with uncertainty. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science (STACS'08)*. 277–288. DOI: 10.4230/LIPIcs.STACS.2008.1358
[9] T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. 2007. Computing shortest paths with uncertainty. *J. Algor.* 62 (2007), 1–18. DOI: 10.1016/j.jalgor.2004.07.005
[10] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. 2003. Computing the median with uncertainty. *SIAM J. Comput.* 32 (2003), 538–547. DOI: 10.1137/S0097539701395668
[11] J. Focke, N. Megow, J. Meißner. 2017. Minimum spanning tree under explorable uncertainty in theory and experiments. In *Proceedings of the International Symposium on Experimental Algorithms (SEA'17)*. 1–22. DOI: 10.4230/LIPIcs.SEA.2017.22
[12] M. Goerigk, M. Gupta, J. Ide, A. Schöbel, and S. Sen. 2015. The robust knapsack problem with queries. *Comput. OR* 55 (2015), 12–22. DOI: 10.1016/j.cor.2014.09.010
[13] M. Gupta, Y. Sabharwal, and S. Sen. 2016. The update complexity of selection and related problems. *Theory Comput. Syst.* 59, 1 (2016), 112–132. DOI: 10.1007/s00224-015-9664-y
[14] S. Kahan. 1991. A model for data in motion. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC'91)*. 267–277. DOI: 10.1145/103418.103449
[15] N. Megow, J. Meißner, and M. Skutella. 2017. Randomization helps computing a minimum spanning tree under uncertainty. *SIAM J. Comput.* 46, 4 (2017), 1217–1240. DOI: 10.1137/16M1088375
[16] G. Reinelt. 1991. TSPLIB—A traveling salesman problem library. *ORSA J. Comput.* 3, 4 (1991), 376–384. DOI: 10.1287/ijoc.3.4.376

Received January 2018; revised March 2019; accepted August 2020