The Algorithmics Column

by

Gerhard J Woeginger

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
gwoegi@win.tue.nl

Query-Competitive Algorithms for Computing with Uncertainty

Thomas Erlebach   Michael Hoffmann

Department of Computer Science
University of Leicester, UK
{te17,mh55}@leicester.ac.uk

Abstract

Motivated by real-life situations in which exact input data is not available initially but can be obtained at a cost, one can consider the model of computing with uncertainty where the input to a problem is initially not known precisely. For each input element, only a set that contains the exact value of that input element is given. The algorithm can repeatedly perform queries to reveal the exact value of an input element. The goal is to minimize the number of queries needed until enough information has been obtained to produce the desired output. The performance of an algorithm is measured using competitive analysis, comparing the number of queries with the best possible number of queries for the given instance. We give a survey of known results and techniques for this model of queryable uncertainty, mention results for some related models, and point to possible directions for future work.

1 Introduction

In the study of algorithms, we often assume that exact input data is available to the algorithm, and our goal is to design algorithms that compute good solutions efficiently. However, there are also scenarios where precise input data is not immediately available. Instead, only rough, uncertain information about the input may be given, for example intervals or other sets that contain the unknown exact input values. In some such scenarios, it is possible to obtain exact information about each part of the input at an additional cost. We refer to the operation of obtaining exact information for one input element as a *query*, and we are interested in minimizing the number of queries to be made until sufficient information has been obtained for calculating a solution. The performance of an algorithm can then be measured by comparing the number of queries made by the algorithm with the best possible number of queries for the given input, following the approach of competitive analysis for online algorithms.

In this article, we give a survey of known results and techniques for the design of query-competitive algorithms in the model of computing with uncertainty sketched above. We also mention results for related questions and point to possible directions for future work.

1.1 Motivation

There are numerous examples of application scenarios that provide motivation for studying computing with uncertainty. For example, as in Kahan's pioneering paper [12], one can consider applications where the input involves objects (e.g., airplanes) that are in motion and whose current positions are not known precisely. If the exact location of an object was known some time in the past and if the object moves with limited speed, then the current location of the object must lie within a known region, and it may be possible to obtain the exact location of the object by communicating with it (e.g., a radio communication with the pilot of an airplane). The goal may be to compute a function of the locations of the objects (e.g., the closest pair of airplanes). A similar application arises in mobile ad-hoc networks, where estimates of the current positions of the nodes may be easily available but obtaining the exact position of a node requires extra communication or measurements. Kahan mentions further applications such as graphic animation (selecting from moving objects those within the field of vision), integrating the maximum of a set of functions of bounded variation, and dynamic scheduling of jobs with time-varying priorities [12].

Another set of applications originates from distributed computing. For example, in systems with distributed database caches, an estimate of an aggregate data value may be available from a local cache, but obtaining the exact value requires more expensive communication (e.g., a query to the master server). Moreover, a system could maintain intervals of values in local database caches, so that updates to data on the master server need to be replicated to a local cache only when the new value lies outside the interval stored there. When a database query cannot be answered based on the intervals stored in the local cache, exact values can be retrieved from the master server. Olston and Widom propose a concrete replication mechanism employing this principle, the TRAPP system [16].

1.2 Other Approaches to Dealing with Uncertainty

There are various other approaches to modeling or addressing problems whose input involves uncertain information. In *stochastic optimization*, uncertain input values are assumed to be drawn from (known) probability distributions, and the goal is to find a solution that optimizes the expected value of the objective function, or that is good or feasible with high probability. In *robust optimization*, a single solution has to be computed that is good in each possible scenario of how the uncertain values can be realized. In two-stage optimization problems, a partial solution has to be computed based on the given uncertain values, and the solution has to be completed in a second stage after the uncertain values have been realized as exact values. This notion can be generalized to multi-stage optimization problems. Neither of these models or approaches involve querying specific input elements in order to obtain their exact values. Therefore, we consider such work outside the scope of this article.

2 Preliminaries

In computing with uncertainty, an instance of a problem typically consists of some structural information S, a set U of elements with uncertain values, and a function A that maps each element $u \in U$ to an *uncertainty set*[1] A_u. The exact values of the uncertain input elements are represented by a function w that maps each $u \in U$ to its exact value w_u, and we require $w_u \in A_u$. Initially the algorithm might not know any of the values w_u. Querying[2] an element $u \in U$ reveals its exact value w_u. We can view a query of $u \in U$ as the operation of replacing A_u by the singleton set $\{w_u\}$. Note that depending on the concrete problem under consideration, w_u could be a real number, a vector of real numbers, or any other type of input data. In cases where the exact values are vectors representing point coordinates, we will sometimes write p_u for w_u.

For a given instance $I = (S, U, A, w)$ of a problem in the model of computing with uncertainty (we refer to such a problem also as an *uncertainty problem*), we let $\phi(S, U, w)$ denote the set of solutions. Note that the set of solutions depends only on the exact values w_u for $u \in U$ but not the uncertainty sets A_u. An algorithm only receives (S, U, A) as input. The goal of the algorithm is to compute a solution in $\phi(S, U, w)$ (or all solutions in $\phi(S, U, w)$) after making a minimum number of queries. Queries are made one by one, and the results of previous queries can be taken into account when determining the next query to make. Therefore, this model is also referred to as the *adaptive* query model.

For a given instance $I = (S, U, A, w)$, we denote by OPT_I (or simply OPT) the minimum number of queries that provide sufficient information for computing a solution in $\phi(S, U, w)$. An algorithm that makes ALG_I queries to solve an instance I is called *ρ-query-competitive* or simply *ρ-competitive* if $ALG_I \leq \rho OPT_I + c$ for

[1]In the literature, the term *uncertainty area* has also been used.

[2]In the literature, queries have also been called *updates* in this context.

all instances I of the problem, where c is a constant independent of the instance. If $c = 0$, we say that the algorithm is *strongly ρ-competitive*. For randomized algorithms, ALG_I is the expected number of queries that the algorithm makes on instance I. Note that the exact value w_u of an uncertain element $u \in U$ can be any value in A_u. We do not assume that w_u is drawn from a probability distribution over the set A_u.

We should briefly clarify what it means formally to say that a set of queries is sufficient to solve a problem. Let $Q \subseteq U$ be a set of queries. Let A^Q be the uncertainty sets arising from A by setting $A_u^Q = \{w_u\}$ if $u \in Q$ and $A_u^Q = A_u$ otherwise. The set of queries Q is sufficient for computing a solution x if and only if $x \in \phi(S, U, w')$ for all w' such that $w'_u \in A_u^Q$ for all $u \in U$ (i.e., if x is a solution for all choices of exact values w' that are consistent with A^Q). If Q is sufficient for computing a solution, we also say that the instance (S, U, A^Q, w) is *solved* (and that Q is a *query solution*), and otherwise *unsolved*.

Another aspect to discuss is which types of sets are allowed as uncertainty sets A_u. In cases where the uncertain elements represent numbers, we will usually assume that each set A_u is either an interval or a singleton set $\{w_u\}$. In the latter case, the exact weight of u is already known, and we say that the element u and its uncertainty set are *trivial*. Elements that are not trivial (and their uncertainty sets) are called *non-trivial*. We write open intervals as (a, b) and closed intervals as $[a, b]$.

Problems in the model of computing with uncertainty (e.g., the minimum spanning tree problem) often display the following behavior: If the task is to compute a single solution, there is a constant-competitive algorithm for the case of open intervals as uncertainty sets but not for the case of closed intervals. The difficulty with closed intervals is that a single query of the right interval can prove that the value of the corresponding element is maximum or minimum among a number of candidates, while a deterministic algorithm has no chance of identifying that interval without querying a large number of elements. On the other hand, if the task is to output all solutions, then a constant competitive ratio can often be achieved also for closed uncertainty sets [12]. Similarly, if the task is to compute the lexicographically first solution, a constant competitive ratio is sometimes possible for closed uncertainty sets [11].

2.1 Example: MST with Uncertainty

Let us consider the minimum spanning tree problem with edge uncertainty. The structural part of an instance of the problem is a connected, undirected graph $G = (V, E)$. The uncertain elements are the edges, and for each edge $e \in E$ an uncertainty set A_e containing its exact weight w_e is given. The task is to compute the edge set of a minimum spanning tree (MST) of the graph G with edge weights

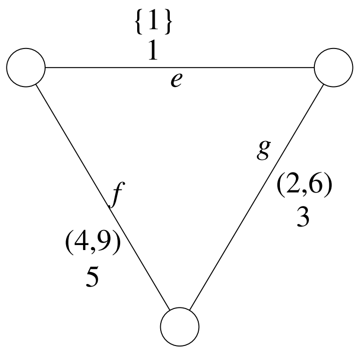

Figure 1: Instance of MST with Uncertainty

given by w, i.e., $\phi(G, E, w)$ is the set consisting of the edge sets of all minimum spanning trees of (G, w).

Consider the example instance $I = (G, E, A, w)$ depicted in Figure 1. The three edges e, f, g have uncertainty sets $A_e = \{1\}$, $A_f = (4, 9)$ and $A_g = (2, 6)$. The exact weights of the edges are $w_e = 1$, $w_f = 5$ and $w_g = 3$. Initially, the algorithm is given only G and A. This information is not sufficient for determining the edge set of an MST: The edge e is clearly contained in any MST, but without querying f or g we do not know which of these two edges is the more expensive one and therefore not contained in the MST. If we decide to query f first, the query reveals that $w_f = 5$. At this point, we still do not know whether f is cheaper or more expensive than g as $A_g = (2, 6)$, so we also need to query g. When we receive the answer $w_g = 3$, we know that g is cheaper than f, and we can determine and output the edge set $\{e, g\}$ as a correct minimum spanning tree. Note that querying only the edge g would also have been sufficient for determining that $\{e, g\}$ is a minimum spanning tree. Therefore, $OPT_I = 1$ but we have made two queries, twice the optimal number. Algorithmic results for the minimum spanning tree problem with uncertainty will be discussed in Section 6.

3 The Witness Algorithm

The main approach that has been used to obtain query-competitive algorithms for computing with uncertainty, as introduced by Bruce et al. [1] and stated in general form by Erlebach et al. [8], is based on considering witness sets: For a given instance $I = (S, U, A, w)$ of an uncertainty problem, a set $W \subseteq U$ is called a *witness set* if it is impossible to determine a solution without querying at least one element of W. In other words, W is a witness set if the instance $(S, U, A^{U \setminus W}, w)$ is unsolved. A natural idea for solving an uncertainty problem is now to repeatedly determine a witness set and query all elements of that witness set (elements with

```
while instance is unsolved do
    $W \leftarrow$ a witness set of the current instance;
    query all $u$ in $W$;
end
```

Algorithm 1: Witness algorithm

a trivial uncertainty set can be skipped, of course), until the instance is solved. An algorithm based on this template, shown in Algorithm 1, is called a *witness algorithm*. Note that the instance changes in each iteration of the while-loop in the algorithm, as the uncertainty sets of the elements in W are replaced by singleton sets.

Most of the known results regarding query-competitive algorithms for uncertainty problems are instantiations or adaptations of the witness algorithm. There is a direct link between the size of witness sets and the competitive ratio, as shown by the following lemma.

Lemma 1 ([1, 8])**.** *If the size of each witness set W that a witness algorithm queries is bounded by ρ, then the algorithm is strongly ρ-competitive*

Lemma 1 can be proved by showing that any query solution must contain at least one distinct element from each witness set queried by the algorithm.

We remark that if an algorithm during its execution queries k sets of elements that are witness sets (and possibly some additional elements), we can infer that any query solution must make at least k queries.

4 Selection and Related Problems

4.1 Identifying All Solutions

Kahan [12] considers the setting where the input consists of a set U of n elements with uncertain values. The uncertainty set of each element $u \in U$ is assumed to be either a closed interval $[u_l, u_h]$ or trivial (in which case the lower bound u_l equals the upper bound u_h).

Maximum. For the problem of identifying all elements of U whose value is equal to the maximum value $\max_{u \in U} w_u$ in U, Kahan presents an algorithm that makes at most $OPT+1$ queries [12]. The algorithm repeatedly queries a non-trivial element u with largest upper bound u_h until all elements with maximum value V have been queried and no non-trivial element u with $u_h \geq V$ remains. Kahan shows that in any query solution at most one non-trivial element u with value $u_h \geq V$ is not queried. As the algorithm queries only elements u with $u_h \geq V$, it makes at most $OPT + 1$ queries.

Considering the input $U = \{u, v\}$ with $A_u = [1, 5]$ and $A_v = [3, 10]$ and either $w_u = 2, w_v = 4$ or $w_u = 4, w_v = 7$, it is clear that making $OPT + 1$ queries in the worst case is optimal among deterministic algorithms.

Median. Assume that $n = |U|$ is odd. For the problem of identifying all elements of U whose value is equal to the value of the median (i.e., the $\lceil n/2 \rceil$-smallest value) of $\{w_u \mid u \in U\}$, Kahan also presents an algorithm that makes at most $OPT + 1$ queries [12]. Let $k = \lceil n/2 \rceil$. It is clear that the value of the median must lie in the range $[m_l, m_h]$, where m_l is the k-smallest lower bound and m_h is the k-largest upper bound. Kahan shows that there must be at least one element $u \in U$ with $[m_l, m_h] \subseteq [u_l, u_h]$. The algorithm for the median problem now works as follows: Compute $[m_l, m_h]$. If there is only one element u whose interval intersects $[m_l, m_h]$, return u as the unique median and quit. If every element whose range contains $[m_l, m_h]$ has already been queried, we must have $m_l = m_h$ and we output all those elements as the set of medians and quit. Otherwise, we query any element whose interval contains $[m_l, m_h]$ and repeat. Again, Kahan shows that in any query solution at most one non-trivial element u whose interval contains the median value is not queried. As the algorithm queries only elements u whose intervals contain the median value, it makes at most $OPT + 1$ queries, and this is again optimal among deterministic algorithms.

Minimum Gap. Now consider the Minimum Gap problem, where the goal is to identify all pairs of elements $u, v \in U$ such that their distance $|w_u - w_v|$ is minimum. Note that for any two elements of U, their uncertainty sets imply lower and upper bounds on the distance between these two elements (and we thus also have an upper bound on the minimum gap). For the Minimum Gap problem, Kahan presents a (weakly) 2-competitive greedy algorithm. The algorithm first queries all non-trivial elements whose interval contains the interval of at least one other element. These elements (except possibly one of them) must be queried by any query solution. The algorithm then repeatedly queries a non-trivial element u for which there is another element v such that the lower bound on the distance between u and v is smaller than the current upper bound on the minimum gap, and is the smallest current lower bound among all pairs of elements (with ties broken in favor of an element whose interval minimizes the maximum distance to non-trivial intervals on the left and right). Kahan analyzes the algorithm by considering connected components of elements in an auxiliary graph that contains an edge between any two elements whose distance lower bound is less than or equal to the minimum gap. He shows that any query solution must contain at least half the elements in a component (with at most one exception), while the greedy algorithm may contain all its elements. Hence, the algorithm is 2-competitive. Furthermore, one can show that no deterministic algorithm can achieve weak competitive ratio smaller than 2 [12].

We remark that an alternative 2-competitive algorithm can be obtained using a witness algorithm: Let L be the current lower bound on the minimum gap. While there are two elements u, v whose distance lower bound is at most L (and at least one of which is non-trivial), query all non-trivial elements in $\{u, v\}$. It is not difficult to see that the set $\{u, v\}$ is a witness set, except possibly in the case where u, v is the unique pair of elements whose distance equals the minimum gap. If the algorithm terminates after k iterations, the algorithm makes at most $2k$ queries and, by the remark after Lemma 1, any query solution makes at least $k-1$ queries. Hence, the algorithm makes at most $2OPT + 2$ queries, matching the performance of Kahan's algorithm.

4.2 Identifying One Solution

Khanna and Tan [13] also consider the setting where the input consists of a set U of n elements with uncertain values and the uncertainty set of each element $u \in U$ is either a closed interval $[u_l, u_h]$ or trivial. Defining ω to be the maximum clique size of the interval graph induced by the uncertainty sets of the elements of U, they present an ω-competitive algorithm for the problem of identifying a k-smallest $u \in U$, for given $k \in \{1, \dots, n\}$. They generalize their result also to the case where queries have different costs, i.e., querying $u \in U$ has a cost of c_u instead of unit cost. Furthermore, they consider the variations of the problem where the goal is to identify an element whose rank in U differs from the given rank k at most by a factor of α (*relative precision*) or by an absolute difference of α (*absolute precision*). They present $O(\omega)$-competitive algorithms for these variations. To show that no algorithm can be better than ω-competitive for the problem of identifying an element with smallest value, Khanna and Tan consider an instance with n elements where all uncertainty sets are $[0, 2]$, one element has value 0, and $n-1$ elements have value 1. This instance has clique size $\omega = n$. A deterministic algorithm can be forced to query all n elements (all queries except the last one return the value 1), while the optimal query set has size 1 and consists of only the minimum element.

4.3 Computing or Approximating the Solution Value

Khanna and Tan [13] additionally consider the problem of approximating the sum (or average) of the values of the n elements in U up to a certain relative precision $\alpha \geq 1$. They present an $O(\min\{\omega, \alpha\})$-competitive algorithm that can also handle different query costs. For the problem of approximating the sum of the values of the elements in U with absolute precision in the case of unit query costs, they remark that a simple greedy algorithm yields the optimal query set. Furthermore, they consider the computation of functions that are composed of aggregation functions and selection functions.

Charikar et al. [3] consider the evaluation of AND/OR trees and generalizations including MIN/MAX trees and present query algorithms with best possible competitive ratio (or within a factor of 2 of the best possible competitive ratio in the generalized case).

5 Geometric Problems

Bruce et al. [1] consider geometric problems in the Euclidean plane in the model of computing with uncertainty. Each input element $u \in U$ corresponds to a point $p_u = (x_u, y_u)$ in the Euclidean plane. The uncertainty set A_u of each $u \in U$ is assumed to be either trivial or the closure of an open, connected region. We refer to an uncertainty set also as an *uncertainty region* in the following.

Bruce et al. introduce the concepts of witness sets and witness algorithms and apply them to the maximal points problem and the convex hull problem. For a property ϕ (such as being a maximal point or a point lying on the convex hull), they classify each uncertainty region A_u as *always* ϕ, *partly* ϕ, *dependent* ϕ or *never* ϕ, depending on whether all points or some point in A_u satisfy the property ϕ and on whether this depends on the point locations in the other uncertainty regions or not.

Maximal points. The task of the maximal points problem is to output all elements $u \in U$ that are maximal, i.e., all u such that there is no element $v \in U$ with $x_v \geq x_u$ and $y_v \geq y_u$ where at least one of the two inequalities is strict. Bruce et al. show that if there is a partly maximal region, then there is a witness set of size 2, and if there is a dependent maximal region but no partly maximal region, then there is a witness set of size 3. Hence, one can always find a witness set of size at most 3 until the instance is solved (i.e., until all uncertainty regions are either always maximal or never maximal), and by Lemma 1 the algorithm is strongly 3-competitive. They also give a lower bound showing that no deterministic algorithm can be better than 3-competitive and prove that there is a linear lower bound on the competitive ratio if arbitrary connected sets are allowed as uncertainty regions.

Convex hull. The task of the convex hull problem is to identify all elements $u \in U$ that correspond to points that lie on the boundary of the convex hull of the point set $\{p_u \mid u \in U\}$. Assuming that the intersection between any two non-trivial uncertainty regions that is non-empty contains at least an ε-ball for some $\varepsilon > 0$, Bruce et al. show that there is always a witness set of size 3, implying a strongly 3-competitive algorithm by Lemma 1. Furthermore, they show that no deterministic algorithm can achieve competitive ratio less than 3, and that there is a linear lower bound on the competitive ratio if arbitrary connected uncertainty sets are allowed.

6 Minimum Spanning Tree

Erlebach et al. [8] consider two variants of the minimum spanning tree (MST) problem in the model of computing with uncertainty. In both variants, the task of the algorithm is to output the edge set of a minimum spanning tree of a given graph $G = (V, E)$. The graph G here represents the structural information S that is provided to the algorithm in addition to the uncertain information.

In the MST problem with edge uncertainty, the edge weights of the given graph $G = (V, E)$ are given in the form of uncertainty sets, i.e., we have $U = E$ and the weight of each edge $e \in E$ is a value w_e contained in the uncertainty set A_e. The uncertainty set of each edge is either trivial or an open set. Each edge has a lower bound and an upper bound (the infimum and the supremum of its uncertainty set). A strongly 2-competitive algorithm is obtained by adapting Kruskal's algorithm to the setting of uncertainty. Edges are inserted into a growing forest in non-decreasing order of lower bounds. When a cycle is formed and if it is not possible to identify an edge of maximum weight in the cycle, a witness set consisting of two edges of the cycle can be identified and queried, and the algorithm is restarted. Otherwise, an edge of maximum weight in the cycle is removed and the algorithm continues. Erlebach et al. also show that no deterministic algorithm can achieve competitive ratio smaller than 2. We illustrate the proof of this lower bound using the graph shown in Figure 1 in Section 2.1: If the algorithm queries f first, the weights of f and g are as shown in the figure. If the algorithm queries g first, the weights are set to $w_f = 8$ and $w_g = 5$. In either case, the algorithm must make two queries while there is a query solution with just one query, and the example can be scaled up arbitrarily by repeating the triangle.

In the MST problem with vertex uncertainty, each vertex $v \in V$ corresponds to a point p_v in the Euclidean plane and the weight of an edge $\{u, v\}$ is the Euclidean distance between p_u and p_v. Instead of the exact point locations, only an uncertainty set A_v is given for each vertex v. It is assumed that each uncertainty set is either trivial or an open region. Erlebach et al. address the problem by observing that it can be viewed as an edge uncertainty problem since the uncertainty sets of the two endpoints of an edge $e = \{u, v\}$ yield an uncertainty set for the weight of e that is trivial or an open set. Hence, the 2-competitive algorithm for the edge uncertainty case can be applied to the vertex uncertainty case, where the query of an edge is replaced by queries of both endpoints of the edge. This yields a strongly 4-competitive algorithm for MST with vertex uncertainty, and it is also shown that no deterministic algorithm can achieve a better competitive ratio.

In [6], it is shown that the strongly 2-competitive algorithm for MST with edge uncertainty can be generalized to the minimum weight matroid base problem.

7 Cheapest Set Problems

Many combinatorial problems can be viewed as set selection problems: The input contains a set U of elements, some subsets of U are *feasible* solutions, and the goal is to output a feasible solution of minimum cost (where the cost of a solution could be the sum of the costs of its elements). Examples of problems that can be seen as set selection problems include the minimum spanning tree problem (the feasible solutions are the edge sets of all spanning trees), the minimum-weight perfect matching problem (the perfect matchings are the feasible subsets of the edge set), or minimum cut problems (the set of edges crossing a cut is a feasible solution). Erlebach et al. [6] study the general formulation of the set selection problem where the input specifies a set U and the family $\mathcal{F}$ of all feasible subsets of U. Instead of the exact weight w_u of each element $u \in U$, an uncertainty set A_u that can be an open interval or trivial is given. The task of the cheapest set problem is to identify a set C in $\mathcal{F}$ that has minimum weight, i.e., $\sum_{u \in C} w_u$ is minimum among all sets in $\mathcal{F}$.

For the case that all sets in $\mathcal{F}$ have cardinality at most d, it is shown that there is an algorithm for the cheapest set problem that makes at most $dOPT + d$ queries, and that this is best possible among deterministic algorithms. The algorithm repeatedly queries all elements of a *robust cheapest set* until a cheapest set can be identified. Here, a robust cheapest set is a set C in $\mathcal{F}$ with the following property: For any choice of exact weights $w_u \in A_u$ for the elements of $u \in U \setminus C$, there are weights $w_u \in A_u$ for each element $u \in C$ such that C is a cheapest set. The analysis shows that a robust cheapest set always exists and that all sets queried by the algorithm, except at most one, are witness sets. By the remark after Lemma 1, this shows that the algorithm makes at most $dOPT + d$ queries.

For cheapest set problems where the sets have a special structure, better algorithms are possible. As we have seen, the minimum spanning tree problem, although corresponding to a cheapest set problem with sets of size $n - 1$ for graphs with n vertices, admits a 2-competitive algorithm. Another family of cheapest set problems with special structure considered in [6] is the family of problems that satisfy the *1-gap property*: If the instance is not solved, there exist two sets $B, C \in \mathcal{F}$ such that $|B \cup C| \leq d + 1$, C is a robust cheapest set, and B is a set that is potentially cheaper than C. An algorithm is presented that makes at most $dOPT + 1$ queries to solve a cheapest set problem that satisfies the 1-gap property. Furthermore, it is shown that the minimum-multicut problem in trees with d terminal pairs satisfies the 1-gap property and can hence be solved with $dOPT + 1$ queries. A matching lower bound showing that this is best possible among deterministic algorithms is also presented.

8 Computing *OPT*: The Verification Problem

In the competitive analysis of algorithms for computing with uncertainty, the number of queries is always compared with the best possible number OPT of queries that suffice to produce the desired output. (Kahan [12] refers to this number OPT as the *lucky number*.) The question then arises how difficult it is to calculate the value of OPT (and possibly also a query solution consisting of OPT queries) if the whole instance (S, U, A, w) of a problem is provided as input (i.e., if also the exact weights of all elements in U are provided to the algorithm). There are at least two scenarios that motivate the computation of an optimal query solution: On the one hand, if the performance of algorithms for computing with uncertainty is evaluated in computational experiments, it is useful to be able to calculate OPT in order to compare the number of queries made by different algorithms with OPT. On the other hand, there may be application scenarios where the values of uncertain elements can change over time, but changes are rare events. Assume that the exact values w_u of all uncertain elements $u \in U$ were known at a point in the past. Let w'_u denote the (unknown) current exact value of each uncertain element $u \in U$. For each $u \in U$, an uncertainty set A_u that guarantees $w'_u \in A_u$ is given. The values w'_u are unknown to the algorithm. We would now like to have an algorithm, called a *verification algorithm*, that queries elements of U and produces one of the following outputs:

(1) A valid solution x with respect to the current weights w'_u.

(2) At least one element $u \in U$ has changed its value (i.e., $w_u \neq w'_u$).

In scenarios where the typical case is that $w'_u = w_u$ for all $u \in U$, querying the elements of an optimal query set for (S, U, A, w) is the best possible strategy for a verification algorithm since it minimizes the number of queries that need to be made until a valid solution x can be identified. Following the terminology of the latter application setting, we refer to the (off-line) problem of computing an optimal query set for a given instance (S, U, A, w) (which is wholly presented to the algorithm) as a *verification problem* (or the *verification version* of an uncertainty problem).

It is easy to see that the verification versions of the maximum, median and minimum gap problems discussed in Section 4 can be solved in polynomial time. In [5], we show that the verification version of MST with edge uncertainty can be solved in polynomial time while the verification version of MST with vertex uncertainty is *NP*-hard.

For general cheapest set problems, Erlebach et al. [7] show that the verification problem is *NP*-hard for sets of size d for any $d \geq 2$. For the minimum multicut problem in trees with d terminal pairs under uncertainty, they prove that the verification version can be solved in polynomial time for fixed d and is *NP*-hard if d is part of the input.

For the problem of outputting all maximal points for a given uncertain point set, Charalambous and Hoffmann [2] show that the verification problem is *NP*-hard. In their reduction, every uncertainty set is either trivial or consists of just two points. The complexity of the verification problem for the case of connected uncertainty sets is left open.

9 Model Variations

9.1 Refinement Queries

We have assumed so far that the query of an uncertain element $u \in U$ returns its exact value w_u. One may also consider settings where the answer to a query is not the exact value, but a reduction of uncertainty. For example, the query of an uncertainty interval $[5, 10]$ could produce a smaller uncertainty interval $[7, 9]$. The same uncertain element may now have to be queried several times, each query reducing the uncertainty interval further. We can then again aim to minimize the number of queries needed until a solution can be computed.

Tseng and Kirkpatrick [17] consider this problem variation in a setting where each input number is given as a finite stream of bits (given in decreasing order of significance) that are initially unknown to the algorithm. Reading an additional bit from one of the bit streams corresponds to a query. The goal is to minimize the number of bits that the algorithm needs to read before it can compute a solution. Algorithms that perform well with respect to this measure are called *input-thrifty* algorithms. Tseng and Kirkpatrick analyze the performance of input-thrifty algorithms relative to a suitably defined *intrinsic cost* of the given instance. They present an algorithm for the extrema testing problem that is within a logarithmic factor of the intrinsic cost of the given instance. They mention that their results also hold in a more general model where a query does not yield one extra bit of a number but the query results are nested uncertainty intervals.

Gupta et al. [11] study the selection problem and the minimum spanning tree problem in a general framework where queries yield refined estimates in the form of sub-intervals. They distinguish all possible cases of combinations of closed intervals, open intervals and trivial sets for the input uncertainty and for the results of queries. Although there are 49 possible models in principle, they are able to classify them into five different main categories and present results on the competitive ratio for selection problems and minimum spanning tree problems for them. For example, they show that the approach of witness algorithms can be generalized to the model with refinement queries for several of the five categories. They also show that for models with closed intervals, query-competitive algorithms become possible if the output is required to be a lexicographically smallest solution rather than an arbitrary solution.

9.2 Non-adaptive Queries

Instead of allowing the algorithm to make queries one by one and take into account the results of previous queries when selecting the next query, one can also require that the algorithm specifies a set $Q \subseteq U$ of queries only once. The queries are then executed in parallel and the algorithm receives the exact values w_u for all queries $u \in Q$. This information must be enough for the algorithm to solve the problem, i.e., no further queries are allowed. This means that the query results, no matter which element of A_u turns out to be the exact value of $u \in Q$, must be sufficient to be able to identify a solution. A query set Q with this property is called *feasible*. The problem of computing a smallest feasible query set Q in this non-adaptive model is actually an off-line problem, as the feasibility of a query set Q does not depend on the exact values of $u \in U$. Note that the problem of determining a smallest feasible query set for the non-adaptive model is different from the verification problem discussed in Section 8.

Much of the existing work on the non-adaptive query model has considered the generalization where different queries have different costs, i.e., each $u \in U$ has a cost $c_u \geq 0$ and the goal is to compute a feasible query set Q of minimum total cost. For given elements with closed intervals as uncertainty sets, Olston and Widom consider selection and aggregation problems in the non-adaptive model, assuming that the goal is to output a range $[L, H]$ that contains the exact solution, where $H - L \leq \delta$ for a given precision requirement δ [16]. For example, it turns out that for the problem of computing the sum of uncertain values up to a given precision, determining an optimal feasible query set boils down to solving a knapsack problem. Feder et al. [10] further study the problem of determining the value of the k-th smallest element, for any $1 \leq k \leq n$, up to a precision of δ in the same model and show that a feasible query set of minimum cost can be computed in polynomial time via linear programming.

In another paper, Feder et al. [9] consider the shortest s-t path problem in the non-adaptive query model. The input consists of a directed acyclic graph $G = (V, E)$ with distinguished vertices $s, t \in V$. The edge weights are uncertain. Each uncertainty set A_e that contains the exact weight w_e of an edge $e \in E$ is assumed to be a closed interval. A set P of candidate s-t-paths is given explicitly as part of the input or implicitly via a description in a certain recursive form (that captures, e.g., the set of all s-t paths in a series-parallel graph). For a given precision parameter $\delta > 0$, the task is to determine a range $[L, H]$ with $H - L \leq \delta$ that contains the length of a shortest s-t-path in P with respect to the exact weights $w_e \geq 0$ for all $e \in E$. They show that the problem of computing a feasible query set of minimum total cost can be solved in polynomial time if $\delta = 0$, i.e., if the exact length of the shortest path in P is sought. For $\delta > 0$, they show that different restrictions of the problem are *NP*-hard or co-*NP*-hard and so the general problem cannot be in *NP* nor co-*NP* if *NP* $\neq$ co-*NP*. They also show that the general problem is in Σ_2.

9.3 Solutions of Minimum or Maximum Objective Value

Another question that has been studied for optimization problems with uncertain input is determining the minimum or maximum possible objective value of an optimal solution, where the minimum/maximum is taken over all possible exact values that the uncertain input elements could take. This is an off-line problem, and queries are not considered in this setting.

For example, Löffler and van Kreveld study the problem of minimizing or maximizing a number of basic geometric measures for input point sets whose locations are described by uncertainty sets. They consider measures such as diameter, width, closest pair, smallest enclosing circle, smallest enclosing bounding box, length of the convex hull, area of the convex hull, or length of a tour visiting the points in a given order. For all these measures and different types of uncertainty sets (e.g., disks, squares, line segments), they present hardness results or efficient algorithms [14, 15]. Dorrigiv et al. [4] consider the setting where the input consists of points in the plane whose uncertainty regions are disjoint disks. They prove that even for this restricted setting it is *NP*-hard to determine the minimum or maximum possible cost of a Euclidean spanning tree of the exact points, and that there is no FPTAS for these problems unless $P = NP$. They also give approximation algorithms with ratios depending on a certain separability parameter of the given instance.

10 Directions for Future Work

We have aimed to give an overview of the existing body of work for computing with uncertainty in models where the algorithm can query the exact values of uncertain elements. Interesting questions that could be addressed in future work include:

- For which uncertainty problems can *randomized algorithms* achieve better competitive ratios than the best possible deterministic algorithms? In particular, it would be interesting to see whether a randomized algorithm can improve over the 2-competitive algorithm for the minimum spanning tree problem under edge uncertainty [8]. The adversarial construction that yields the lower bound of 2 for deterministic algorithms can be adapted to a lower bound of 1.5 for randomized algorithms, but it is open whether there exists a randomized algorithm with competitive ratio better than 2.

- Existing work has considered the adaptive query model where queries are asked one by one, and the non-adaptive query model where all queries are asked in parallel. It may be interesting to consider a model where *queries are asked in rounds*. Each round makes a number of queries in parallel, and the results of queries made in previous rounds can be taken into account when determining the queries for the next round. For example, one could study the trade-off between the number of rounds and the total query cost or consider settings where the number of rounds is constrained to be at most a given value k. Another direction would be to restrict the number of queries that can be made in one round to a given number and aim to minimize the number of query rounds.

- As far as we are aware, in the existing published work on computing with uncertainty it has been assumed that the uncertainty of different input elements is independent, in the sense that receiving the answer of one query does not provide any information about the exact values of the remaining uncertain elements. One could imagine settings where querying one element also reduces the uncertainty of other elements. For example, if the input consists of three uncertain numbers together with the exact value of their sum, learning the exact value of one number may potentially reduce the size of the uncertainty sets of the other two numbers. It would be interesting to study uncertainty problems with such *dependencies between the uncertainty sets of different input elements*.

- As discussed in Section 9.3, for every uncertain problem input one can ask for the minimum and maximum possible objective value of the solution, over all possible exact input values that lie in the given uncertainty sets. One can view the difference between the minimum and the maximum objective value as uncertainty in the solution value, and an interesting direction could be to study query strategies that *reduce the uncertainty in the solution value* to a given target value.

- It would be interesting to develop additional techniques or even a *general theory* that allows us to classify problems with respect to the best possible query-competitive ratio that can be achieved. For example, we know that the existence of small witness sets (that can be found by the algorithm) for a problem leads to algorithms with small competitive ratio by Lemma 1, but it is unclear whether there are further general criteria with this property. As an example of a more general type of result, we mention that Kahan [12] considers the problem of computing a function f of n uncertain real-valued elements. The n exact input values can be represented as a point in n-dimensional space. The function f maps $\mathbb{R}^n$ to $\mathbb{R}$ and partitions its domain into regions where the function value is constant. Kahan relates the achievable competitive ratio to the existence of a point on some partition boundary that is tangent to an $(n-1)$-dimensional hyperplane intersecting k co-ordinate axes.

- As a query algorithm that solves an uncertainty problem can be viewed as trying to learn the solution by asking queries about the input, it appears that there may be *connections to work in machine learning* that could be explored further.

References

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

[2] G. Charalambous and M. Hoffmann. Verification problem of maximal points under uncertainty. In *International Workshop on Combinatorial Algorithms (IWOCA 2013)*, LNCS 8288, pages 94–105. Springer, 2013.

[3] M. Charikar, R. Fagin, V. Guruswami, J. M. Kleinberg, P. Raghavan, and A. Sahai. Query strategies for priced information. *Journal of Computer and System Sciences*, 64(4):785–819, 2002.

[4] R. Dorrigiv, R. Fraser, M. He, S. Kamali, A. Kawamura, A. López-Ortiz, and D. Seco. On minimum- and maximum-weight minimum spanning trees with neighborhoods. *Theory of Computing Systems*, November 2014. Online First publication.

[5] T. Erlebach and M. Hoffmann. Minimum spanning tree verification under uncertainty. In *41st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2014)*, LNCS 8747, pages 164–175. Springer, 2014.

[6] T. Erlebach, M. Hoffmann, and F. Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. In *39th International Symposium on Mathematical Foundations of Computer Science (MFCS 2014), Part II*, pages 263–274. Springer, 2014.

[7] T. Erlebach, M. Hoffmann, and F. Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. Full version of [6]. Submitted, 2014.

[8] T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihalák, and R. Raman. Computing minimum spanning trees with uncertainty. In *25th International Symposium on Theoretical Aspects of Computer Science (STACS'08)*, pages 277–288, 2008.

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

[10] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32(2):538–547, 2003.

[11] M. Gupta, Y. Sabharwal, and S. Sen. The update complexity of selection and related problems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *LIPIcs*, pages 325–338. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

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

[13] S. Khanna and W.-C. Tan. On computing functions with uncertainty. In *20th Symposium on Principles of Database Systems (PODS'01)*, pages 171–182, 2001.

[14] M. Löffler and M. J. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.

[15] M. Löffler and M. J. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry*, 43(4):419–433, 2010.

[16] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *26th International Conference on Very Large Data Bases (VLDB'00)*, pages 144–155, 2000.

[17] K.-C. R. Tseng and D. G. Kirkpatrick. Input-thrifty extrema testing. In *22nd International Symposium on Algorithms and Computation (ISAAC 2011)*, LNCS 7074, pages 554–563. Springer, 2011.