Round-Competitive Algorithms for Uncertainty Problems with Parallel Queries

Thomas Erlebach
School of Informatics, University of Leicester, UK

Michael Hoffmann
School of Informatics, University of Leicester, UK

Murilo Santos de Lima[1]
School of Informatics, University of Leicester, UK

Abstract

The area of computing with uncertainty considers problems where some information about the input elements is uncertain, but can be obtained using queries. For example, instead of the weight of an element, we may be given an interval that is guaranteed to contain the weight, and a query can be performed to reveal the weight. While previous work has considered models where queries are asked either sequentially (adaptive model) or all at once (non-adaptive model), and the goal is to minimize the number of queries that are needed to solve the given problem, we propose and study a new model where k queries can be made in parallel in each round, and the goal is to minimize the number of query rounds. We use competitive analysis and present upper and lower bounds on the number of query rounds required by any algorithm in comparison with the optimal number of query rounds. Given a set of uncertain elements and a family of m subsets of that set, we present an algorithm for determining the value of the minimum of each of the subsets that requires at most $(2+\varepsilon) \cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$ rounds for every $0 < \varepsilon < 1$, where opt_k is the optimal number of rounds, as well as nearly matching lower bounds. For the problem of determining the i-th smallest value and identifying all elements with that value in a set of uncertain elements, we give a 2-round-competitive algorithm. We also show that the problem of sorting a family of sets of uncertain elements admits a 2-round-competitive algorithm and this is the best possible.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete mathematics; Theory of computation → Theory and algorithms for application domains

Keywords and phrases online algorithms, competitive analysis, explorable uncertainty, parallel algorithms, minimum problem, selection problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.27

Related Version *Full Version*: `https://arxiv.org/abs/2101.05032`

Funding This research was supported by EPSRC grant EP/S033483/1.

Acknowledgements We would like to thank Markus Jablonka for helpful discussions.

1 Introduction

Motivated by real-world applications where only rough information about the input data is initially available but precise information can be obtained at a cost, researchers have considered a range of **uncertainty problems with queries** [7, 13, 14, 15, 16, 19, 26]. This research area has also been referred to as **queryable uncertainty** [12] or **explorable uncertainty** [17]. For example, in the input to a sorting problem, we may be given for each

[1] Corresponding author

38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021).
Editors: Markus Bläser and Benjamin Monmege; Article No. 27; pp. 27:1–27:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

input element, instead of its precise value, only an interval containing that point. Querying an element reveals its precise value. The goal is to make as few queries as possible until enough information has been obtained to solve the sorting problem, i.e., to determine a linear order of the input elements that is consistent with the linear order of the precise values. Motivation for explorable uncertainty comes from many different areas (see [12] and the references given there for further examples): The uncertain input elements may, e.g., be locations of mobile nodes or approximate statistics derived from a distributed database cache [29]. Exact information can be obtained at a cost, e.g., by requesting GPS coordinates from a mobile node, by querying the master database or by a distributed consensus algorithm.

The main model that has been studied in the explorable uncertainty setting is the **adaptive query model**: The algorithm makes queries one by one, and the results of previous queries can be taken into account when determining the next query. The number of queries made by the algorithm is then compared with the best possible number of queries for the given input (i.e., the minimum number of queries sufficient to solve the problem) using competitive analysis [5]. An algorithm is **ρ-query-competitive** (or simply ρ-competitive) if it makes at most ρ times as many queries as an optimal query set. A very successful algorithm design paradigm in this area is based on the concept of **witness sets** [7, 14]. A witness set is a set of input elements for which it is guaranteed that every query set that solves the problem contains at least one query in that set. If a problem admits witness sets of size at most ρ, one obtains a ρ-query-competitive algorithm by repeatedly finding a witness set and querying all its elements.

Some work has also considered the **non-adaptive query model** (see, e.g., [15, 28, 29]), where all queries are made simultaneously and the set of queries must be chosen in such a way that they certainly reveal sufficient information to solve the problem. In the non-adaptive query model, one is interested in complexity results and approximation algorithms.

In settings where the execution of a query takes a non-negligible amount of time and there are sufficient resources to execute a bounded number of queries simultaneously, the query process can be completed faster if queries are not executed one at a time, but in **rounds** with k simultaneous queries. Such scenarios include e.g. IoT environments (such as drones measuring geographic data), or teams of interviewers doing market research. Apart from being well motivated from an application point of view, this variation of the model is also theoretically interesting because it poses new challenges in selecting a useful set of k queries to be made simultaneously. Somewhat surprisingly, however, this has not been studied yet. In this paper, we address this gap and analyze for the first time a model where the algorithm can make up to k queries per round, for a given value k. The query results from previous rounds can be taken into account when determining the queries to be made in the next round. Instead of minimizing the total number of queries, we are interested in minimizing the number of query rounds, and we say that an algorithm is **ρ-round-competitive** if, for any input, it requires at most ρ times as many rounds as the optimal query set.

A main challenge in the setting with k queries per round is that the witness set paradigm alone is no longer sufficient for obtaining a good algorithm. For example, if a problem admits witness sets with at most 2 elements, this immediately implies a 2-query-competitive algorithm for the adaptive model, but only a k-round-competitive algorithm for the model with k queries per round. (The algorithm is obtained by simply querying one witness set in each round, and not making use of the other $k-2$ available queries.) The issue is that, even if one can find a witness set of size at most ρ, the identity of subsequent witness sets may depend on the outcome of the queries for the first witness set, and hence we may not know how to compute a number of different witness sets that can fill a query round if $k \gg \rho$.

Our contribution. Apart from introducing the model of explorable uncertainty with k queries per round, we study several problems in this model: Minimum, Selection and Sorting. For Minimum (or Sorting), we assume that the input can be a family $\mathcal{S}$ of subsets of a given ground set $\mathcal{I}$ of uncertain elements, and that we want to determine the value of the minimum of (or sort) all those subsets. For Selection, we are given a set $\mathcal{I}$ of n uncertain elements and an index $i \in \{1, \ldots, n\}$, and we want to determine the i-th smallest value of the n precise values, and all the elements of $\mathcal{I}$ whose value is equal to that value.

Our main contribution lies in our results for the Minimum problem. We present an algorithm that requires at most $(2 + \varepsilon) \cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$ rounds, for every $0 < \varepsilon < 1$, where opt_k is the optimal number of rounds and $m = |\mathcal{S}|$. (The execution of the algorithm does not depend on ε, so the upper bound holds in particular for the best choice of $0 < \varepsilon < 1$ for given opt_k and m.) Interestingly, our algorithm follows a non-obvious approach that is reminiscent of primal-dual algorithms, but no linear programming formulation features in the analysis. For the case that the sets in $\mathcal{S}$ are disjoint, we obtain some improved bounds using a more straightforward algorithm. We also give lower bounds that apply even to the case of disjoint sets, and show that our upper bounds are close to best possible. Note that the Minimum problem is equivalent to the problem of determining the maximum element of each of the sets in $\mathcal{S}$, e.g., by simply negating all the numbers involved. A motivation for studying the Minimum problem thus arises from the minimum spanning tree problem with uncertain edge weights [11, 14, 17, 26]: Determining the maximum-weight edge of each cycle of a given graph allows one to determine a minimum spanning tree. Therefore, there is a connection between the problem of determining the maximum of each set in a family of possibly overlapping sets (which could be the edge sets of the cycles of a given graph) and the minimum spanning tree problem. The minimum spanning tree problem with uncertain edge weights has not been studied yet for the model with k queries per round, and seems to be difficult for that setting. In particular, it is not clear in advance for which cycles of the graph a maximum-weight edge actually needs to be determined, and this makes it very difficult to determine a set of k queries that are useful to be asked in parallel. We hope that our results for Minimum provide a first step towards solving the minimum spanning tree problem.

Another motivation for solving multiple possibly overlapping sets comes from distributed database caches [29], where one wants to answer database queries using cached local data and a minimum number of queries to the master database. Values in the local database cache may be uncertain, and exact values can be obtained by communicating with the central master database. Different database queries might ask for the record with minimum value in the field with uncertain information among a set of database records satisfying certain criteria, or for a list of such database records sorted by the field with uncertain information. Answering such database queries while making a minimum number of queries for exact values to the master database corresponds to the Minimum and Sorting problems we consider.

For the Selection problem, we obtain a 2-round-competitive algorithm. For Sorting, we show that there is a 2-round-competitive algorithm, by adapting ideas from a recent algorithm for sorting in the standard adaptive model [21], and that this is best possible.

We also discuss the relationship between our model and another model of parallel queries proposed by Meißner [27], and we give general reductions between both settings.

Literature overview. The seminal paper on minimizing the number of queries to solve a problem on uncertainty intervals is by Kahan [22]. Given n elements in uncertainty intervals, he presented optimal deterministic adaptive algorithms for finding the maximum, the median, the closest pair, and for sorting. Olston and Widom [29] proposed a distributed database system which exploits uncertainty intervals to improve performance. They gave non-adaptive algorithms for finding the maximum, the sum, the average and for counting problems. They also considered the case in which errors are allowed within a given bound, so a trade-off between performance and accuracy can be achieved. Khanna and Tan [23] extended this previous work by investigating adaptive algorithms for the situation in which bounded errors are allowed. They also considered the case in which query costs may be non-uniform, and presented results for the selection, sum and average problems, and for compositions of such functions. Feder *et al.* [16] studied the generalized median/selection problem, presenting optimal adaptive and non-adaptive algorithms. They proved that those are the best possible adaptive and non-adaptive algorithms, respectively, instead of evaluating them from a competitive analysis perspective. They also investigated the **price of obliviousness**, which is the ratio between the non-adaptive and adaptive strategies.

After this initial foundation, many classic discrete problems were studied in this framework, including geometric problems [7, 9], shortest paths [15], network verification [4], minimum spanning tree [11, 14, 17, 26], cheapest set and minimum matroid base [13, 28], linear programming [25, 30], traveling salesman [32], knapsack [19], and scheduling [2, 3, 10]. The concept of witness sets was proposed by Bruce *et al.* [7], and identified as a pattern in many algorithms by Erlebach and Hoffmann [12]. Gupta *et al.* [20] extended this framework to the setting where a query may return a refined interval, instead of the exact value of the element.

The problem of sorting uncertainty data has received some attention recently. Halldórsson and de Lima [21] presented better query-competitive algorithms, by using randomization or assumptions on the underlying graph structure. Other related work on sorting has considered sorting with noisy information [1, 6] or preprocessing the uncertain intervals so that the actual numbers can be sorted efficiently once their precise value are revealed [31].

The idea of performing multiple queries in parallel was also investigated by Meißner [27]. Her model is different, however. Each round/batch can query an unlimited number of intervals, but at most a fixed number of rounds can be performed. The goal is to minimize the total number of queries. Meißner gave results for selection, sorting and minimum spanning tree problems. We discuss this model in Section 6. A similar model was also studied by Canonne and Gur for property testing [8].

Organization of the paper. We present some definitions and preliminary results in Section 2. Sections 3, 4 and 5 are devoted to the sorting, minimum and selection problems, respectively. In Section 6, we discuss the relationship between the model we study and the model of Meißner for parallel queries [27]. We conclude in Section 7.

2 Preliminaries and Definitions

For the problems we consider, the input consists of a set of n continuous uncertainty intervals $\mathcal{I} = \{I_1, \ldots, I_n\}$ in the real line. The precise value of each data item is $v_i \in I_i$, which can be learnt by performing a query; formally, a query on I_i replaces this interval with $\{v_i\}$. We wish to solve the given problem by performing the minimum number of queries (or query rounds). We say that a closed interval $I_i = [\ell_i, u_i]$ is **trivial** if $\ell_i = u_i$; clearly $I_i = \{v_i\}$, so trivial intervals never need to be queried. Some problems require that intervals are either open or trivial; we will discuss this in further detail when addressing each problem. For a given realization $v_1, \ldots, v_n$ of the precise values, a set $Q \subseteq \mathcal{I}$ of intervals is a **feasible query set** if querying Q is enough to solve the given problem (i.e., to output a solution that can be proved correct based only on the given intervals and the answers to the queries in Q), and an **optimal query set** is a feasible query set of minimum size. Since the precise values are initially unknown to the algorithm and can be defined adversarially, we have an online exploration problem [5]. We fix an optimal query set OPT_1, and we write $\mathrm{opt}_1 := |\mathrm{OPT}_1|$. An algorithm which performs up to $\rho \cdot \mathrm{opt}_1$ queries is said to be **ρ-query-competitive**. Throughout this paper, we only consider deterministic algorithms.

In previous work on the adaptive model, it is assumed that queries are made sequentially, and the algorithm can take the results of all previous queries into account when deciding the next query. We consider a model where queries are made in **rounds** and we can perform up to k queries in parallel in each round. The algorithm can take into account the results from all queries made in previous rounds when deciding which queries to make in the next round. The adaptive model with sequential queries is the special case of our model with $k = 1$. We denote by opt_k the optimal number of rounds to solve the given instance. Note that $\mathrm{opt}_k = \lceil \mathrm{opt}_1/k \rceil$ as OPT_1 only depends on the input intervals and their precise values and can be distributed into rounds of k queries arbitrarily. For an algorithm ALG we denote by ALG_1 the number of queries it makes, and by ALG_k the number of rounds it uses. An algorithm which solves the problem in up to $\rho \cdot \mathrm{opt}_k$ rounds is said to be **ρ-round-competitive**. A query performed by an algorithm that is not in OPT_1 is called a **wasted** query, and we say that the algorithm **wastes** that query; a query performed by an algorithm that is not wasted is **useful**.

▶ **Proposition 2.1.** *If an algorithm makes all queries in* OPT_1*, wastes w queries in total over all rounds excluding the final round, always makes k queries per round except possibly in the final round, and stops as soon as the queries made so far suffice to solve the problem, then its number of rounds will be* $\lceil (\mathrm{opt}_1 + w)/k \rceil \leq \mathrm{opt}_k + \lceil w/k \rceil$.

The problems we consider are Minimum, Sorting and Selection. For Minimum and Sorting, we assume that we are given a set $\mathcal{I}$ of n intervals and a family $\mathcal{S}$ of m subsets of $\mathcal{I}$. For Sorting, the task is to output, for each set $S \in \mathcal{S}$, an ordering of the elements in S that is consistent with the order of their precise values. For Minimum, the task is to output, for each $S \in \mathcal{S}$, an element whose precise value is the minimum of the precise values of all elements in S, along with the value of that element.[2] Regarding the family $\mathcal{S}$, we can distinguish the cases where $\mathcal{S}$ contains a single set, where all sets in $\mathcal{S}$ are pairwise disjoint, and the case where the sets in $\mathcal{S}$ may overlap, i.e., may have common elements. For Selection, we are given a set $\mathcal{I}$ of n intervals and an index $i \in \{1, \ldots, n\}$. The task is to output the i-th smallest value v^* (i.e., the value in position i in a sorted list of the precise values of the n intervals), as well as the set of intervals whose precise value equals v^*. We also discuss briefly a variant of Minimum in which we seek all elements whose precise value is the minimum and a variant of Selection in which we only seek the value v^*.

For a better understanding of the problems, we give a simple example for Sorting with $k = 1$. We have a single set with two intersecting intervals. There are four different configurations of the realizations of the precise values, which are shown in Figure 1. In Figure 1a, it is enough to query I_1 to learn that $v_1 < v_2$; however, if an algorithm first queries I_2, it cannot decide the order, so it must query I_1 as well. In Figure 1b we have a symmetric situation. In Figure 1c, both intervals must be queried (i.e., the only feasible query set is $\{I_1, I_2\}$), otherwise it is not possible to decide the order. Finally, in Figure 1d it is

[2] In some of the literature, it is only required to identify the element with minimum value. Returning the precise minimum value, however, is also an important problem, as discussed in [26, Section 7] for the minimum spanning tree problem.

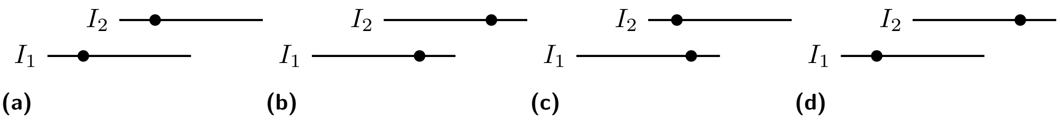

Figure 1 Example of Sorting for two intervals and the possible realizations of the precise values. We have that $\text{opt}_1 = 1$ in (a), (b) and (d), and $\text{opt}_1 = 2$ in (c).

enough to query either I_1 or I_2; hence, both $\{I_1\}$ and $\{I_2\}$ are feasible query sets. Since those realizations are initially identical to the algorithm, this example shows that no deterministic algorithm can be better than 2-query-competitive, and this example can be generalized by taking multiple copies of the given structure. For Minimum, however, an optimum solution can always be obtained by first querying I_1 (and then I_2 only if necessary): Since we need the precise value of the minimum element, in Figure 1b it is not enough to just query I_2.

3 Sorting

In this section we discuss the Sorting problem. We allow open, half-open, closed, and trivial intervals in the input, i.e., I_i can be of the form $[\ell_i, u_i]$ with $\ell_i \leq u_i$, or $(\ell_i, u_i]$, $[\ell_i, u_i)$ or (ℓ_i, u_i) with $\ell_i < u_i$.

First, we consider the case where $\mathcal{S}$ consists of a single set S, which we can assume to contain all n of the given intervals. We wish to find a permutation $\pi : [n] \to [n]$ such that $v_i \leq v_j$ if $\pi(i) < \pi(j)$, by performing the minimum number of queries possible. This problem was addressed for $k = 1$ in [21, 22, 27]; it admits 2-query-competitive deterministic algorithms and has a deterministic lower bound of 2.

For Sorting, if two intervals $I_i = [\ell_i, u_i]$ and $I_j = [\ell_j, u_j]$ are such that $I_i \cap I_j = \{u_i\} = \{\ell_j\}$, then we can put them in a valid order without any further queries, because clearly $v_i \leq v_j$. Therefore, we say that two intervals I_i and I_j **intersect** (or are **dependent**) if either their intersection contains more than one point, or if I_i is trivial and $v_i \in (\ell_j, u_j)$ (or *vice versa*). This is equivalent to saying that I_i and I_j are dependent if and only if $u_i > \ell_j$ and $u_j > \ell_i$. Two simple facts are important to notice, which are proven in [21]:

- For any pair of intersecting intervals, at least one of them must be queried in order to decide their relative order; i.e., any intersecting pair is a witness set.
- The **dependency graph** that represents this relation, with a vertex for each interval and an edge between intersecting intervals, is an interval graph [24].

We adapt the 2-query-competitive algorithm for Sorting by Halldórsson and de Lima [21] for $k = 1$ to the case of arbitrary k. Their algorithm first queries all non-trivial intervals in a minimum vertex cover in the dependency graph. By the duality between vertex covers and independent sets, the unqueried intervals form an independent set, so no query is necessary to decide the order between them. However, the algorithm still must query intervals in the independent set that intersect a trivial interval or the value of a queried interval. To adapt the algorithm to the case of arbitrary k, we first compute a minimum vertex cover and fill as many rounds as necessary with the given queries. After the answers to the queries are returned, we use as many rounds as necessary to query the intervals of the remaining independent set that contain a trivial point.

▶ **Theorem 3.1.** *The algorithm of Halldórsson and de Lima [21] yields a 2-round-competitive algorithm for Sorting that runs in polynomial time.*

Proof. Any feasible query set is a vertex cover in the dependency graph, due to the fact that at least one interval in each intersecting pair must be queried. Therefore a minimum vertex cover is at most the size of an optimal query set, so the first phase of the algorithm spends at most opt_k rounds. Since all intervals queried in the second phase are in any solution, again we spend at most another opt_k rounds. As the minimum vertex cover problem for interval graphs can be solved in polynomial time [18], the overall algorithm is polynomial as well. ◀

The problem has a lower bound of 2 on the round-competitive factor. This can be shown by having kc copies of a structure consisting of two dependent intervals, for some $c \geq 1$. OPT_1 may query only one interval in each pair, while we can force any deterministic algorithm to query both of them (cf. the configurations shown in Figures 1a and 1b). We have that $\mathrm{opt}_k = c$ while any deterministic algorithm will spend at least $2c$ rounds.

We remark that the 2-query-competitive algorithm for Sorting with $k = 1$ due to Meißner [27], when adapted to the setting with arbitrary k in the obvious way, only gives a bound of $2 \cdot \mathrm{opt}_k + 1$ rounds. Her algorithm first greedily computes a maximal matching in the dependency graph and queries all non-trivial matched vertices, and then all remaining intervals that contain a trivial point.

Now we study the case of solving a number of problems on different subsets of the same ground set of uncertain elements. In such a setting, it may be better to perform queries that can be reused by different problems, even if the optimum solution for one problem may not query that interval. We can reuse ideas from the algorithms for single problems that rely on the dependency graph. We define a new dependency relation (and dependency graph) in such a way that two intervals are dependent if and only if they intersect *and* belong to a common set. Note that the resulting graph may not be an interval graph, so some algorithms for single problems may not run in polynomial time for this generalization.

If we perform one query at a time ($k = 1$), then there are 2-competitive algorithms. One such is the algorithm by Meißner [27] described above; since a maximal matching can be computed greedily in polynomial time for arbitrary graphs, this algorithm runs in polynomial time for non-disjoint problems. If we can make $k \geq 2$ queries in parallel, then this algorithm performs at most $2 \cdot \mathrm{opt}_k + 1$ rounds, and the analysis is tight since we may have an incomplete round in between the two phases of the algorithm. If we relax the requirement that the algorithm runs in polynomial time, then we can obtain an algorithm that needs at most $2 \cdot \mathrm{opt}_k$ rounds, by first querying non-trivial intervals in a minimum vertex cover of the dependency graph (in as many rounds as necessary) and then the intervals that contain a trivial interval or the value of a queried interval (again, in as many rounds as necessary).

4 The Minimum Problem

For the Minimum problem, we assume without loss of generality that the intervals are sorted by non-decreasing left endpoints; intervals with the same left endpoint can be ordered arbitrarily. The **leftmost** interval among a subset of $\mathcal{I}$ is the one that comes earliest in this ordering. We also assume that all intervals are open or trivial; otherwise the problem has a trivial lower bound of n on the query-competitive ratio [20].

First, consider the case $\mathcal{S} = \{\mathcal{I}\}$, i.e., we have a single set. It is easy to see that the optimal query set consists of all intervals whose left endpoint is strictly smaller than the precise value of the minimum: If I_i with precise value v_i is a minimum element, then all other intervals with left endpoint strictly smaller than v_i must be queried to rule out that their value is smaller than v_i, and I_i must be queried (unless it is a trivial interval) to determine the value of the minimum. The optimal set of queries is hence a *prefix* of the sorted list of uncertain intervals (sorted by non-decreasing left endpoint). This shows that there is a 1-query-competitive algorithm when $k = 1$, and a 1-round-competitive algorithm for arbitrary k: In each round we simply query the next k uncertain intervals in the order of non-decreasing left endpoint, until the problem is solved. For $k = 1$, the same method yields a 1-query-competitive algorithm for the case with several sets: The algorithm can always query an interval with smallest left endpoint for any of the sets that have not yet been solved.[3]

In the remainder of this section, we consider the case of multiple sets and $k > 1$. We first present a more general result for potentially overlapping sets, then we give better upper bounds for disjoint sets. At the end of the section, we also present lower bounds.

Let $W(x) = x \lg x$; the inverse W^{-1} of W will show up in our analysis. Note that $W^{-1}(x) = \Theta(x/\lg x)$; this can be proved via implicit differentiation.

Throughout this section, we assume w.l.o.g. that the optimum must make at least one query in each set (or we consider only sets that require some query). We also assume that any algorithm always discards from each set all elements that are certainly not the minimum of that set, i.e., all elements for which it is already clear based on the available information that their value must be larger than the minimum value of the set (this is where the right endpoints of intervals also need to be considered). We adopt the following terminology. A set in $\mathcal{S}$ is **solved** if we can determine the value of its minimum element. A set is **active** at the start of a round if the queries made in previous rounds have not solved the set yet. An active set **survives** a round if it is still active at the start of the next round. An active set that does not survive the current round is said to be **solved in** the current round.

To illustrate these concepts, let us discuss a first simple strategy to build a query set Q for a round. Let $\mathcal{P}$ be the set of intervals queried in previous rounds. The **prefix length** of an active set S is the length of the maximum prefix of elements from Q in the list of non-trivial intervals in $S \setminus \mathcal{P}$ ordered by non-decreasing left endpoints. The algorithm proceeds by repeatedly adding to Q the leftmost non-trivial element not in $Q \cup \mathcal{P}$ from an arbitrary active set with minimum prefix length. We call this the **balanced** algorithm, and denote it by BAL. We give an example of its execution in Figure 2, with $m = 3$ disjoint sets and $k = 5$. The optimum solution queries the first three elements in S_1 and S_2, and all elements in S_3. Since the algorithm picks an arbitrary active set with minimum prefix length, it may give preference to S_1 and S_2 over S_3, thus wasting one query in S_1 and one in S_2 in round 2. All sets are active at the beginning of round 2; S_1 and S_2 are solved in round 2, while S_3 survives round 2. Since S_1 and S_2 are solved in round 2, they are no longer active in round 3, so the algorithm no longer queries any of their elements.

4.1 The Minimum Problem with Arbitrary Sets

We are given a set $\mathcal{I}$ of n intervals and a family $\mathcal{S}$ of m possibly overlapping subsets of $\mathcal{I}$, and a number $k \geq 2$ of queries that can be performed in each round.

Unfortunately, it is possible to construct an instance in which BAL uses as many as $k \cdot \mathrm{opt}_k$ rounds. In particular, it does not take into consideration that some elements are shared between different sets. The challenge is how to balance queries between sets in a better way.

[3] If we want to determine all elements whose value equals the minimum, it is not hard to see that the optimal set of queries for each set is again a prefix. As all our algorithms require only this property, we obtain corresponding results for that problem variant, even for inputs with arbitrary closed, open and half-open intervals.

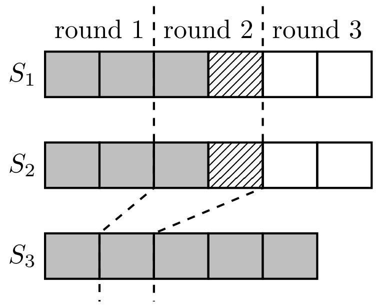

Figure 2 Possible execution of BAL for $m = 3$ disjoint sets and $k = 5$. Each interval is represented by a box, and the optimum solution is a prefix of each set. The solid boxes are useful queries, the two hatched boxes are wasted queries, and the white boxes are not queried by the algorithm.

Algorithm 1 Computing a query round for possibly non-disjoint sets.

Data: family $\mathcal{S} = \{S_1, \ldots, S_m\}$ of active subsets of the ground set $\mathcal{I}$
Result: set $Q \subseteq \mathcal{I}$ of at most k queries to make

begin
  $Q \leftarrow$ set of leftmost unqueried elements of all sets in $\mathcal{S}$;
  if $|Q| \geq k$ **then**
    $Q \leftarrow$ arbitrary subset of Q with size k;
  else
    $b_i \leftarrow 0$ for all $S_i \in \mathcal{S}$;
    while $|Q| < k$ *and there are unqueried elements in* $\mathcal{I} \setminus Q$ **do**
      foreach $e \in \mathcal{I} \setminus Q$ **do**
        $F_e \leftarrow \{i \mid e$ is the leftmost unqueried element from $\mathcal{I} \setminus Q$ in $S_i\}$;
      increase all b_i simultaneously at the same rate until there is an unqueried element $e \in \mathcal{I} \setminus Q$ that satisfies $\sum_{i \in F_e} b_i = 1$;
      $Q \leftarrow Q \cup \{e\}$;
      $b_i \leftarrow 0$ for all $i \in F_e$;
  return Q;

We give an algorithm that requires at most $(2 + \varepsilon) \cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$ rounds, for every $0 < \varepsilon < 1$. (The execution of the algorithm does not depend on ε, so the upper bound holds in particular for the best choice of $0 < \varepsilon < 1$ for given opt_k and m.) It is inspired by how some primal-dual algorithms work. The pseudocode for determining the queries to be made in a round is shown in Algorithm 1. First, we try to include the leftmost element of each set in the set of queries Q. If those are not enough to fill a round, then we maintain a variable b_i for each set S_i, which can be interpreted as a budget for each set. The variables are increased simultaneously at the same rate, until the sets that share a current leftmost unqueried element not in Q have enough budget to buy it. More precisely, at a given point of the execution, for each element $e \in \mathcal{I} \setminus Q$, let F_e contain the indices of the sets that have e as their leftmost unqueried element not in Q. We include e in Q when $\sum_{i \in F_e} b_i = 1$, and then we set b_i to zero for all $i \in F_e$. We repeat this process until $|Q| = k$ or there are no unqueried elements in $\mathcal{I} \setminus Q$.

When a query e is added to Q, we say that it is **charged** to the sets S_i with $i \in F_e$. The amount of charge for set S_i is equal to the value of b_i just before b_i is reset to 0 after adding e to Q. We also say that the set S_i **pays** this amount for e.

▶ **Definition 4.1.** *Let $\varepsilon > 0$. A round is* **ε-good** *if at least $k/2$ of the queries made by Algorithm 1 are also in OPT_1 (i.e., are useful queries), or if at least a/r active sets are solved in that round, where a is the number of active sets at the start of the round and $r = (2(1+\varepsilon) + \sqrt{2\varepsilon^2 + 4\varepsilon + 4})/\varepsilon$. A round that is not ε-good is called* **ε-bad**.

Note that $r > 2$ for any $\varepsilon > 0$.

▶ **Lemma 4.2.** *If a round is ε-bad, then Algorithm 1 will make at least $2k/(2+\varepsilon)$ useful queries in the following round.*

Proof. Let a denote the number of active sets at the start of an ε-bad round. Let s be the number of sets that are solved in the current round; note that $s < a/r$ because the current round is ε-bad. Let T be the total amount by which each value b_i has increased during the execution of Algorithm 1. If the simultaneous increase of all b_i is interpreted as time passing, then T corresponds to the point in time when the computation of the set Q has been completed. For example, if some set S_i did not pay for any element during the whole execution, then T is equal to the value of b_i at the end of the execution of Algorithm 1.

Let Q be the set of queries that Algorithm 1 makes in the current round. We claim that every wasted query in Q is charged only to sets that are solved in this round. Consider a wasted query e that is in some set S_j not solved in this round. At the time e was selected, j cannot have been in F_e because otherwise e would be a useful query. Therefore, we do not charge e to S_j.

The total number of wasted queries is therefore bounded by Ts, as these queries are paid for by the s sets solved in this round. As the number of wasted queries in a bad round is larger than $k/2$, we therefore have $Ts > k/2$. As $s < a/r$, we get $k/2 < Ta/r$, so $T > (r/2) \cdot (k/a)$.

Call a surviving set S_i **rich** if $b_i > k/a$ when the computation of Q is completed. A set that is not rich is called **poor**. Note that a poor set must have spent at least an amount of $(r/2 - 1) \cdot (k/a) > 0$, as its total budget would be at least $T > (r/2) \cdot (k/a)$ if it had not paid for any queries. As the poor sets have paid for fewer than $k/2$ elements in total (as there are fewer than $k/2$ useful queries in the current round), the number of poor sets is bounded by $\frac{k/2}{(r/2-1)\cdot(k/a)} = a/(r-2) > 0$. As there are more than $(1 - 1/r) \cdot a$ surviving sets and at most $a/(r-2)$ of them are poor, there are at least $(1 - 1/r) \cdot a - a/(r-2) = ((r-2)(r-1) - r)/(r(r-2)) \cdot a = 2a/(2+\varepsilon) > 0$ surviving sets that are rich.

Let e be any element that is the leftmost unqueried element (at the end of the current round) of a rich surviving set. If e was the leftmost unqueried element of more than a/k rich surviving sets, those sets would have been able to pay for e (because their total remaining budget would be greater than $k/a \cdot a/k = 1$) before the end of the execution of Algorithm 1, a contradiction to e not being included in Q. Hence, the number of distinct leftmost unqueried elements of the at least $2a/(2+\varepsilon)$ rich surviving sets is at least $(2a/(2+\varepsilon))/(a/k) = 2k/(2+\varepsilon)$. So the following round will query at least $2k/(2+\varepsilon)$ elements that are the leftmost unqueried element of an active set, and all those are useful queries that are made in the next round. ◀

▶ **Theorem 4.3.** *Let opt_k denote the optimal number of rounds and A_k the number of rounds made if the queries are determined using Algorithm 1. Then, for every $0 < \varepsilon < 1$, $A_k \leq (2+\varepsilon) \cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$.*

Proof. In every round, one of the following must hold:
- The algorithm makes at least $k/2$ useful queries.
- The algorithm solves at least a fraction of $1/r$ of the active sets.
- If none of the above hold, the algorithm makes at least $2k/(2+\varepsilon)$ useful queries in the following round (by Lemma 4.2).

The number of rounds in which the algorithm solves at least a fraction of $1/r$ of the active sets is bounded by $\lceil \log_{r/(r-1)} m \rceil = \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$, since $1/\left(\lg \frac{r}{r-1}\right) < 5/\varepsilon$ for $0 < \varepsilon < 1$. In every round where the algorithm does not solve at least a fraction of $1/r$ of the active sets, the algorithm makes at least $k/(2+\varepsilon)$ useful queries on average (if in any such round it makes fewer than $k/2$ useful queries, it makes $2k/(2+\varepsilon)$ useful queries in the following round). The number of such rounds is therefore bounded by $(2+\varepsilon) \cdot \mathrm{opt}_k$. ◀

We do not know if this analysis is tight, so it would be worth investigating this question.

4.2 The Minimum Problem with Disjoint Sets

We now consider the case where $k \geq 2$ and the m sets in the given family $\mathcal{S}$ are pairwise disjoint. For this case, it turns out that the balanced algorithm achieves good upper bounds.

▶ **Theorem 4.4.** $\mathrm{BAL}_k \leq \mathrm{opt}_k + \mathrm{O}(\lg \min\{k, m\})$.

Proof. First we prove the bound for $m \leq k$. Index the sets in such a way that S_i is the i-th set that is solved by BAL, for $1 \leq i \leq m$. Sets that are solved in the same round are ordered by non-decreasing number of queries made in them in that round by BAL. In the round when S_i is solved, there are at least $m - (i-1)$ active sets, so the number of wasted queries in S_i is at most $\frac{k}{m-(i-1)}$. (BAL makes at most $\left\lceil \frac{k}{m-(i-1)} \right\rceil$ queries in S_i, and at least one of these is not wasted.) The total number of wasted queries is then at most $\sum_{i=1}^{m} \frac{k}{m-(i-1)} = \sum_{i=1}^{m} k/i = k \cdot H(m)$, where $H(m)$ denotes the m-th Harmonic number. By Proposition 2.1, $\mathrm{BAL}_k \leq \mathrm{opt}_k + \mathrm{O}(\lg m)$.

If $m > k$, observe that the algorithm does not waste any queries until the number of active sets is at most k. From that point on, it wastes at most $k \cdot H(k)$ queries following the arguments in the previous paragraph, so the number of rounds is bounded by $\mathrm{opt}_k + \mathrm{O}(\log k)$. ◀

We now give a more refined analysis that provides a better bound for $\mathrm{opt}_k = 1$, as well as a better multiplicative bound than what would follow from Theorem 4.4.

▶ **Lemma 4.5.** *If* $\mathrm{opt}_k = 1$, *then* $\mathrm{BAL}_k \leq \mathrm{O}(\lg m / \lg \lg m)$.

Proof. Consider an arbitrary instance of the problem with $\mathrm{opt}_k = 1$. Let $R+1$ be the number of rounds needed by the algorithm. For each of the first R rounds, we consider the fraction b_i of active sets that are not solved in that round. More formally, for the i-th round, for $1 \leq i \leq R$, if a_i denotes the number of active sets at the start of round i and a_{i+1} the number of active sets at the end of round i, then we define $b_i = a_{i+1}/a_i$.

Consider round i, $1 \leq i \leq R$. A set that is active at the start of round i and is still active at the start of the round $i+1$ is called a **surviving set**. A set that is active at the start of round i and gets solved by the queries made in round i is called a **solved set**. For each surviving set, all queries made in that set in round i are useful. For each solved set, at least one query made in that set is useful. We claim that this implies the algorithm makes at least kb_i useful queries in round i. To see this, observe that if the algorithm makes $\lfloor k/a_i \rfloor$ queries in a surviving set and $\lceil k/a_i \rceil$ queries in a solved set, we can conceptually move one useful query from the solved set to the surviving set. After this, the a_{i+1} surviving sets contain at least k/a_i useful queries on average, and hence $a_{i+1} \cdot k/a_i = b_i k$ useful queries in total.

As OPT_1 must make all useful queries and makes at most k queries in total, we have that $\sum_{i=1}^{R} kb_i \leq \mathrm{opt}_1 \leq k$, so $\sum_{i=1}^{R} b_i \leq 1$. Furthermore, as there are m active sets initially and there is still at least one active set after round R, we have that $\prod_{i=1}^{R} b_i = a_{R+1}/a_1 \geq 1/m$. To get an upper bound on R, we need to determine the largest possible value of R for which there exist values $b_i > 0$ for $1 \leq i \leq R$ satisfying $\sum_{i=1}^{R} b_i \leq 1$ and $\prod_{i=1}^{R} b_i \geq 1/m$. We gain nothing from choosing b_i with $\sum_{i=1}^{R} b_i < 1$, so we can assume $\sum_{i=1}^{R} b_i = 1$. In that case, the value of $\prod_{i=1}^{R} b_i$ is maximized if we set all b_i equal, namely $b_i = 1/R$. So we need to determine the largest value of R that satisfies $\prod_{i=1}^{R} 1/R \geq 1/m$, or equivalently $R^R \leq m$, or $R \lg R \leq \lg m$. This shows that $R \leq W^{-1}(\lg m) = \mathrm{O}(\lg m / \lg\lg m)$. ◀

▶ **Corollary 4.6.** *If* $\mathrm{opt}_k = 1$*, then* $\mathrm{BAL}_k \leq \mathrm{O}(\lg k / \lg\lg k)$*.*

Proof. If $k \geq m$, then the corollary follows from Lemma 4.5. If $k < m$, there can be at most k active sets, because the optimum performs at most k queries since $\mathrm{opt}_k = 1$. Hence, we only need to consider these k sets and can apply Lemma 4.5 with $m = k$. ◀

Now we wish to extend these bounds to arbitrary opt_k. It turns out that we can reduce the analysis for an instance with arbitrary opt_k to the analysis for an instance with $\mathrm{opt}_k = 1$, assuming that BAL is implemented in a round-robin fashion. A formal description of such an implementation is as follows: fix an arbitrary order of the m sets of the original problem instance as $S_1, S_2, \ldots, S_m$, and consider it as a cyclic order where the set after S_m is S_1. In each round, BAL distributes the k queries to the active sets as follows. Let i be the index of the set to which the last query was distributed in the previous round (or let $i = m$ if we are in the first round). Then initialize $Q = \emptyset$ and repeat the following step k times. Let j be the first index after i such that S_j is active and has unqueried non-trivial elements that are not in Q; pick the leftmost unqueried non-trivial element in $S_j \setminus Q$, insert it into Q, and set $i = j$. The resulting set Q is then queried. The proof of the following theorem is omitted.

▶ **Theorem 4.7.** BAL *is* $\mathrm{O}(\lg \min\{k, m\} / \lg\lg \min\{k, m\})$*-round-competitive.*

4.3 Lower Bounds

In this section we present lower bounds for Minimum that hold even for the more restricted case where the family $\mathcal{S}$ consists of disjoint sets.

▶ **Theorem 4.8.** *For arbitrarily large m and any deterministic algorithm* ALG*, there exists an instance with m sets and $k > m$ queries per round, such that* $\mathrm{opt}_k = 1$*,* $\mathrm{ALG}_k \geq W^{-1}(\lg m)$ *and* $\mathrm{ALG}_k = \Omega(W^{-1}(\lg k))$*. Hence, there is no* $\mathrm{o}(\lg \min\{k, m\} / \lg\lg \min\{k, m\})$*-round-competitive deterministic algorithm.*

Proof. Fix an arbitrarily large positive integer M. Consider an instance with $m = M^M$ sets, and let $k = M^{M+1}$. Each set contains Mk elements, with the i-th element having uncertainty interval $(1 + i\varepsilon, 100 + i\varepsilon)$ for $\varepsilon = 1/m$. The adversary will pick for each set an index j and set the j-th element to be the minimum, by letting it have value $1 + (j + 0.5)\varepsilon$, while the i-th element for $i \neq j$ is given value $100 + (i - 0.5)\varepsilon$. The optimal query set for the set is thus its first j elements. We assume that an algorithm queries the elements of each set in order of increasing lower interval endpoints. (Otherwise, the lower bound only becomes larger.)

Consider the start of a round when $a \leq m$ sets are still active; initially $a = m$. The adversary observes how the algorithm distributes its k queries among the active sets and repeatedly adds the active set with largest number of queries (from the current round) to a set $\mathcal{L}$, until the total number of queries from the current round in sets of $\mathcal{L}$ is at least $(M-1)k/M$. Let $\mathcal{S}'$ denote the remaining active sets. Note that $|\mathcal{S}'| \geq a/M$. For the sets in $\mathcal{L}$, the adversary chooses the minimum in such a way that a single query in the current round would have been sufficient to find it, while the sets in $\mathcal{S}'$ remain active (and so the optimum must make the same queries in them that the algorithm has made in the current round, and these are at most k/M queries). We continue for M rounds. In the M-th round, the adversary picks the minimum in all remaining sets in such way that a single query in that round would have been sufficient to solve the set. The optimal number of queries is then at most $(M-1)k/M + M^M = (M-1)k/M + k/M = k$, and hence $\mathrm{opt}_k = 1$. On the other hand, we have $\mathrm{ALG}_k = M$.

We can now express this lower bound in terms of k or m as follows: As $m = M^M$, we have $\lg m = M \lg M$ and hence $M = W^{-1}(\lg m)$. As $k = M^{M+1}$, we have $\lg k = (M+1)\lg M$ and hence $M = \Omega(W^{-1}(\lg k))$. Thus, the theorem follows. ◀

▶ **Theorem 4.9.** *No deterministic algorithm* ALG *attains* $\mathrm{ALG}_k \leq \mathrm{opt}_k + \mathrm{o}(\lg \min\{k, m\})$.

Proof. Let $k = m$ be an arbitrarily large integer. The intervals of the m sets are chosen as in the proof of Theorem 4.8, for a sufficiently large value of M. Let a be the number of active sets at the start of a round; initially $a = m$. After each round, the adversary considers the set S_j in which the algorithm has made the largest number of queries, which must be at least k/a. The adversary picks the minimum element in S_j in such a way that a single query in the current round would have been enough to solve it, and keeps all other sets active. This continues for m rounds. The number of wasted queries is at least $k/m + k/(m-1) + \cdots + k/2 + k - m = k \cdot (H(m) - 1) = k \cdot \Omega(\lg k)$. As the algorithm must also make all queries in OPT_1, the theorem follows from Proposition 2.1. ◀

We conclude thus that the balanced algorithm attains matching upper bounds for disjoint sets. For non-disjoint sets, a small gap remains between our lower and upper bounds.

5 Selection

An instance of the Selection problem is given by a set $\mathcal{I}$ of n intervals and an integer i, $1 \leq i \leq n$. Throughout this section we denote the i-th smallest value in the set of n precise values by v^*.

If we only want to find the value v^*, then we can adapt the analysis in [20] to obtain an algorithm that performs at most $\mathrm{opt}_1 + i - 1$ queries provided all input intervals are open or trivial, simply by querying the intervals in the order of their left endpoints. This is the best possible and can easily be parallelized in $\mathrm{opt}_k + \lceil \frac{i-1}{k} \rceil$ rounds (we omit a proof). Thus we focus here on the task of finding v^* as well as identifying all intervals in $\mathcal{I}$ whose precise value equals v^*.

For ease of presentation, we assume that all the intervals in $\mathcal{I}$ are closed. The result can be generalized to arbitrary intervals without any significant new ideas, but the proofs become longer and require more cases, so we defer them to an extended version of the paper.

Let us begin by observing that the optimal query set is easy to characterize (proof omitted).

▶ **Lemma 5.1.** *Every feasible query set contains all non-trivial intervals that contain v^*. The optimal query set OPT_1 contains all non-trivial intervals that contain v^* and no other intervals.*

Let I_{j_1} be the interval with the i-th smallest left endpoint, and let I_{j_2} be the interval with the i-th smallest right endpoint. Then it is clear that v^* must lie in the interval $[\ell_{j_1}, u_{j_2}]$, which we call the **target area**. The following lemma was essentially shown by Kahan [22].

▶ **Lemma 5.2** (Kahan, 1991). *Assume that the current instance of* SELECTION *is not yet solved. Then there is at least one non-trivial interval I_j in $\mathcal{I}$ that contains the target area, i.e., satisfies $\ell_j \leq \ell_{j_1}$ and $u_j \geq u_{j_2}$.*

For $k = 1$, there is therefore an online algorithm that makes opt_1 queries: In each round, it determines the target area of the current instance and queries a non-trivial interval that contains the target area. (This algorithm was essentially proposed by Kahan [22] for determining all elements with value equal to v^*, without necessarily determining v^*.) For larger k, the difficulty is how to select additional intervals to query if there are fewer than k intervals that contain the target area.

The intervals that intersect the target area can be classified into four categories:

(1) a non-trivial intervals $[\ell_j, u_j]$ with $\ell_j \leq \ell_{j_1}$ and $u_j \geq u_{j_2}$; they **contain** the target area;

(2) b intervals $[\ell_j, u_j]$ with $\ell_j > \ell_{j_1}$ and $u_j < u_{j_2}$; they **are strictly contained** in the target area and contain neither endpoint of the target area;

(3) c intervals $[\ell_j, u_j]$ with $\ell_j \leq \ell_{j_1}$ and $u_j < u_{j_2}$; they intersect the target area on the **left**;

(4) d intervals $[\ell_j, u_j]$ with $\ell_j > \ell_{j_1}$ and $u_j \geq u_{j_2}$; they intersect the target area on the **right**.

We propose the following algorithm for rounds with k queries: Each round is filled with as many non-trivial intervals as possible, using the following order: first all intervals of category (1); then intervals of category (2); then picking intervals alternatingly from categories (3) and (4), starting with category (3). If one of the two categories (3) and (4) is exhausted, the rest of the k queries is chosen from the other category. Intervals of categories (3) and (4) are picked in order of non-increasing length of overlap with the target area, i.e., intervals of category (3) are chosen in non-increasing order of right endpoint, and intervals of category (4) in non-decreasing order of left endpoint. When a round is filled, it is queried, and the algorithm restarts, with a new target area and the intervals redistributed into the categories.

▶ **Proposition 5.3.** *At the start of any round, $a \geq 1$ and $b \leq a - 1$.*

Proof. Lemma 5.2 shows $a \geq 1$. If the target area is trivial, we have $b = 0$ and hence $b \leq a - 1$. From now on assume that the target area is non-trivial.

Let L be the set of intervals in $\mathcal{I}$ that lie to the left of the target area, i.e., intervals I_j with $u_j < \ell_{j_1}$. Similarly, let R be the set of intervals that lie to the right of the target area. Observe that $a + b + c + d + |L| + |R| = n$.

The intervals in L and the intervals of type (1) and (3) include all intervals with left endpoint at most ℓ_{j_1}. As ℓ_{j_1} is the i-th smallest left endpoint, we have $|L| + a + c \geq i$.

Similarly, the intervals in R and the intervals of type (1) and (4) include all intervals with right endpoint at least u_{j_2}. As u_{j_2} is the i-th smallest right endpoint, or equivalently the $(n - i + 1)$-th largest right endpoint, we have $|R| + a + d \geq n - i + 1$.

Adding the two inequalities derived in the previous two paragraphs, we get $2a + c + d + |L| + |R| \geq n + 1$. Combined with $a + b + c + d + |L| + |R| = n$, this yields $b \leq a - 1$. ◀

We omit the proof of the following lemma.

▶ **Lemma 5.4.** *If the current round of the algorithm is not the last one, then the following holds: If the algorithm queries at least one interval of categories (3) or (4), then the algorithm does not query all intervals of category (3) that contain v^*, or it does not query all intervals of category (4) that contain v^*.*

▶ **Theorem 5.5.** *There is a* 2*-round-competitive algorithm for* Selection*.*

Proof. Consider any round of the algorithm that is not the last one. Let A, B, C and D be the sets of intervals of categories (1), (2), (3) and (4) that are queried in this round, respectively. Let A^*, B^*, C^* and D^* be the subsets of A, B, C and D that are in OPT_1, respectively. By Lemmas 5.1 and 5.2, $|A| = |A^*| \geq 1$. Since the algorithm prioritizes category (1), by Proposition 5.3 we have $|B| \leq |A| - 1$, and thus $|A \cup B| \leq 2 \cdot |A| - 1 = 2 \cdot |A^*| - 1 \leq 2(|A^*| + |B^*|) - 1$.

For bounding the size of $C \cup D$, first note that the order in which the algorithm selects the elements of categories (3) and (4) ensures that, within each category, the intervals that contain v^* are selected first. By Lemma 5.4, there exists a category in which the algorithm does not query all intervals that contain v^* in the current round. If that category is (3), we have $|C| = |C^*|$ and, by the alternating choice of intervals from (3) and (4) starting with (3), $|D| \leq |C|$ and hence $|C \cup D| \leq 2 \cdot |C^*| \leq 2(|C^*| + |D^*|)$. If that category is (4), we have $|D| = |D^*|$ and $|C| \leq |D| + 1$, giving $|C \cup D| \leq 2 \cdot |D^*| + 1 \leq 2(|C^*| + |D^*|) + 1$. In both cases, we thus have $|C \cup D| \leq 2(|C^*| + |D^*|) + 1$.

Combining the bounds obtained in the two previous paragraphs, we get $|A \cup B \cup C \cup D| \leq 2(|A^*| + |B^*| + |C^*| + |D^*|)$. This shows that, among the queries made in the round, at most half are wasted. The total number of wasted queries in all rounds except the last one is hence bounded by opt_1. Since the algorithm fills each round except possibly the last one and also queries all intervals in OPT_1, the theorem follows by Proposition 2.1. ◀

We also have the following lower bound, which proves that our algorithm has the best possible multiplicative factor. We remark that it uses instances with $\mathrm{opt}_k = 1$, and we do not know how to scale it to larger values of opt_k. In its present form, it does not exclude the possibility of an algorithm using at most $\mathrm{opt}_k + 1$ rounds.

▶ **Lemma 5.6.** *There is a family of instances of* Selection *with* $k = i \geq 2$ *with* $\mathrm{opt}_1 \leq i$ *(and hence* $\mathrm{opt}_k = 1$*) such that any algorithm that makes* k *queries in the first round needs at least two rounds and performs at least* $\mathrm{opt}_1 + \lceil (i-1)/2 \rceil$ *queries.*

6 Relationship with the Parallel Model by Meißner

In [27, Section 4.5], Meißner describes a slightly different model for parallelization of queries. There, one is given a maximum number r of **batches** that can be performed, and there is no constraint on the number of queries that can be performed in a given batch. The goal is to minimize the total number of queries performed, and the algorithm is compared to an optimal query set. The number of uncertain elements in the input is denoted by n. In this section, we discuss the relationship between this model and the one we described in the previous sections.

Meißner argues that the sorting problem admits a 2-query-competitive algorithm for $r \geq 2$ batches. For the minimum problem with one set, she gives an algorithm which is $\lceil n^{1/r} \rceil$-query-competitive, with a matching lower bound. She also gives results for the selection and the minimum spanning tree problems.

▶ **Theorem 6.1.** *If there is an* α*-query-competitive algorithm that performs at most* r *batches, then there is an algorithm that performs at most* $\alpha \cdot \mathrm{opt}_k + r - 1$ *rounds of* k *queries. Conversely, if a problem has a lower bound of* $\beta \cdot \mathrm{opt}_k + t$ *on the number of rounds of* k *queries, then any algorithm running at most* $t + 1$ *batches has query-competitive ratio at least* β*.*

Proof. Given an α-query-competitive algorithm A on r batches, we construct an algorithm B for rounds of k queries in the following way. For each batch in A, algorithm B simply performs all queries in as many rounds as necessary. In between batches, we may have an incomplete round, but there are only $r-1$ such rounds. ◀

In view of Meißner's lower bound for the minimum problem with one set mentioned above, the following result is close to being asymptotically optimal for that problem (using $\alpha = 1$). The proof of the following theorem is omitted due to space constraints.

▶ **Theorem 6.2.** *If there is an α-round-competitive algorithm for rounds of k queries, with α independent of k, then there is an algorithm that performs at most r batches with query-competitive ratio $\mathrm{O}(\alpha \cdot n^{\lfloor \alpha \rfloor/(r-1)})$, with $r \geq \lfloor \alpha \rfloor \cdot x + 1$ for an arbitrary natural number x. In particular, for $r \geq \lfloor \alpha \rfloor \cdot \lg n + 1$, the query-competitive factor is $\mathrm{O}(\alpha)$.*

Therefore, an algorithm that uses a constant number of batches implies an algorithm with the same asymptotic round-competitive ratio for rounds of k queries. On the other hand, some problems have worse query-competitive ratio if we require few batches, even if we have round-competitive algorithms for rounds of k queries, but the ratio is preserved by a constant if the number of batches is sufficiently large.

7 Final Remarks

We propose a model with parallel queries and the goal of minimizing the number of query rounds when solving uncertainty problems. Our results show that, even though the techniques developed for the sequential setting can be utilized in the new framework, they are not enough, and some problems are harder (have a higher lower bound on the competitive ratio).

One interesting open question is how to extend our algorithms for Minimum to the variant where it is not necessary to return the precise minimum value, but just to identify the minimum element. Another problem one could attack is the following generalization of Selection: Given multiple sets $S_1, \ldots, S_m \subseteq \mathcal{I}$ and indices $i_1, \ldots, i_m$, identify the i_j-smallest precise value and all elements with that value in S_j, for $j = 1, \ldots, m$. It would be interesting to see if the techniques we developed for Minimum with multiple sets can be adapted to Selection with multiple sets.

It would be nice to close the gaps in the round-competitive ratio, to understand if the analysis of Algorithm 1 is tight, and to study whether randomization can help to obtain better upper bounds. One could also study other problems in the parallel model, such as the minimum spanning tree problem.

References

1 M. Ajtai, V. Feldman, A. Hassidim, and J. Nelson. Sorting and selection with imprecise comparisons. *ACM Transactions on Algorithms*, 12(2):19:1–19:19, 2016. doi:10.1145/2701427.

2 S. Albers and A. Eckl. Explorable uncertainty in scheduling with non-uniform testing times. In *WAOA 2020: 18th International Workshop on Approximation and Online Algorithms*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2021. To appear. Also: arXiv preprint, arXiv:2009.13316, 2020.

3 L. Arantes, E. Bampis, A. V. Kononov, M. Letsios, G. Lucarelli, and P. Sens. Scheduling under uncertainty: A query-based approach. In *IJCAI 2018: 27th International Joint Conference on Artificial Intelligence*, pages 4646–4652, 2018. doi:10.24963/ijcai.2018/646.

4 Z. Beerliova, F. Eberhard, T. Erlebach, A. Hall, M. Hoffmann, M. Mihal'ák, and L. S. Ram. Network discovery and verification. *IEEE Journal on Selected Areas in Communications*, 24(12):2168–2181, 2006. doi:10.1109/JSAC.2006.884015.

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

6 M. Braverman and E. Mossel. Sorting from noisy information. *arXiv preprint*, 2009. arXiv:0910.1191.

7 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. doi:10.1007/s00224-004-1180-4.

8 C. L. Canonne and T. Gur. An adaptivity hierarchy theorem for property testing. *computational complexity*, 27:671–716, 2018. doi:10.1007/s00037-018-0168-4.

9 G. Charalambous and M. Hoffmann. Verification problem of maximal points under uncertainty. In T. Lecroq and L. Mouchard, editors, *IWOCA 2013: 24th International Workshop on Combinatorial Algorithms*, volume 8288 of *Lecture Notes in Computer Science*, pages 94–105. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-45278-9_9.

10 C. Dürr, T. Erlebach, N. Megow, and J. Meißner. An adversarial model for scheduling with testing. *Algorithmica*, 2020. doi:10.1007/s00453-020-00742-2.

11 T. Erlebach and M. Hoffmann. Minimum spanning tree verification under uncertainty. In D. Kratsch and I. Todinca, editors, *WG 2014: International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 8747 of *Lecture Notes in Computer Science*, pages 164–175. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-319-12340-0_14.

12 T. Erlebach and M. Hoffmann. Query-competitive algorithms for computing with uncertainty. *Bulletin of the EATCS*, 116:22–39, 2015. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/335.

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

14 T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihal'ák, and R. Raman. Computing minimum spanning trees with uncertainty. In S. Albers and P. Weil, editors, *STACS'08: 25th International Symposium on Theoretical Aspects of Computer Science*, volume 1 of *Leibniz International Proceedings in Informatics*, pages 277–288. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2008. doi:10.4230/LIPIcs.STACS.2008.1358.

15 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. doi:10.1016/j.jalgor.2004.07.005.

16 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. doi:10.1137/S0097539701395668.

17 J. Focke, N. Megow, and J. Meißner. Minimum spanning tree under explorable uncertainty in theory and experiments. In C. S. Iliopoulos, S. P. Pissis, S. J. Puglisi, and R. Raman, editors, *SEA 2017: 16th International Symposium on Experimental Algorithms*, volume 75 of *Leibniz International Proceedings in Informatics*, pages 22:1–22:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.SEA.2017.22.

18 F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972. doi:10.1137/0201013.

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

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

21 M. M. Halldórsson and M. S. de Lima. Query-competitive sorting with uncertainty. In P. Rossmanith, P. Heggernes, and J.-P. Katoen, editors, *MFCS 2019: 44th International Symposium on Mathematical Foundations of Computer Science*, volume 138 of *Leibniz International Proceedings in Informatics*, pages 7:1–7:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.7.

22 S. Kahan. A model for data in motion. In *STOC'91: 23rd Annual ACM Symposium on Theory of Computing*, pages 265–277, 1991. doi:10.1145/103418.103449.

23 S. Khanna and W.-C. Tan. On computing functions with uncertainty. In *PODS'01: 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 171–182, 2001. doi:10.1145/375551.375577.

24 C. Lekkerkerker and J. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962. URL: https://eudml.org/doc/213681.

25 T. Maehara and Y. Yamaguchi. Stochastic packing integer programs with few queries. *Mathematical Programming*, 182:141–174, 2020. doi:10.1007/s10107-019-01388-x.

26 N. Megow, J. Meißner, and M. Skutella. Randomization helps computing a minimum spanning tree under uncertainty. *SIAM Journal on Computing*, 46(4):1217–1240, 2017. doi:10.1137/16M1088375.

27 J. Meißner. *Uncertainty Exploration: Algorithms, Competitive Analysis, and Computational Experiments*. PhD thesis, Technische Universität Berlin, 2018. doi:10.14279/depositonce-7327.

28 A. I. Merino and J. A. Soto. The minimum cost query problem on matroids with uncertainty areas. In C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, editors, *ICALP 2019: 46th International Colloquium on Automata, Languages, and Programming*, volume 132 of *Leibniz International Proceedings in Informatics*, pages 83:1–83:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.83.

29 C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB 2000: 26th International Conference on Very Large Data Bases*, pages 144–155, 2000. URL: http://ilpubs.stanford.edu:8090/437/.

30 I. O. Ryzhov and W. B. Powell. Information collection for linear programs with uncertain objective coefficients. *SIAM Journal on Optimization*, 22(4):1344–1368, 2012. doi:10.1137/12086279X.

31 I. van der Hoog, I. Kostitsyna, M. Löffler, and B. Speckmann. Preprocessing ambiguous imprecise points. In G. Barequet and Y. Wang, editors, *SoCG 2019: 35th International Symposium on Computational Geometry*, volume 129 of *Leibniz International Proceedings in Informatics*, pages 42:1–42:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.SoCG.2019.42.

32 W. A. Welz. *Robot Tour Planning with High Determination Costs*. PhD thesis, Technische Universität Berlin, 2014. URL: https://www.depositonce.tu-berlin.de/handle/11303/4597.