Algorithmica (2023) 85:406–443
https://doi.org/10.1007/s00453-022-01035-6

Round-Competitive Algorithms for Uncertainty Problems with Parallel Queries

Thomas Erlebach[1] · Michael Hoffmann[2] · Murilo Santos de Lima[3]

Received: 18 May 2021 / Accepted: 28 August 2022 / Published online: 15 September 2022

Abstract

In computing with explorable uncertainty, one considers problems where the values of some input elements are uncertain, typically represented as intervals, but can be obtained using queries. Previous work has considered query minimization in the settings where queries are asked sequentially (adaptive model) or all at once (non-adaptive model). We introduce 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. Using competitive analysis, we present upper and lower bounds on the number of query rounds required by any algorithm in comparison with the optimal number of query rounds for the given instance. Given a set of uncertain elements and a family of m subsets of that set, we study the problems of sorting all m subsets and of determining the minimum value (or the minimum element(s)) of each subset. We also study the selection problem, i.e., the problem of determining the i-th smallest value and identifying all elements with that value in a given set of uncertain elements. Our results include 2-round-competitive algorithms for sorting and selection and an algorithm for the minimum value problem

The work by Murilo S. de Lima was done while employed at University of Leicester and supported by EPSRC grant EP/S033483/1.

A preliminary version of this paper appeared in the proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021), volume 187 of LIPICs, article 27, 2021. https://doi.org/10.4230/LIPIcs.STACS.2021.27.

✉ Thomas Erlebach
thomas.erlebach@durham.ac.uk

Michael Hoffmann
mh55@le.ac.uk

Murilo Santos de Lima
mslima@ic.unicamp.br

[1] Department of Computer Science, Durham University, Durham, UK

[2] School of Informatics, University of Leicester, Leicester, UK

[3] Munich, Germany

that uses at most $(2+\varepsilon)\cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon}\cdot \lg m\right)$ query rounds for every $0 < \varepsilon < 1$, where opt_k is the optimal number of query rounds.

Keywords Online algorithms · Competitive analysis · Explorable uncertainty · Parallel algorithms · Minimum problem · Selection problem · Sorting problem

Mathematics Subject Classification 68W40 · 68W10

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, 16, 17, 20, 28]. This research area has also been referred to as *queryable uncertainty* [12] or *explorable uncertainty* [18]. For example, in the input to a sorting problem, we may be given for each input element, instead of its precise value, only an interval containing that value. 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 [31]. 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 such a witness set and querying all its elements.

Some work has also considered the *non-adaptive query model* (see, e.g., [16, 30, 31]), 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. (In some sense, this model can be interpreted as being midway between the adaptive and non-adaptive query models.) 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 $i \in \{1, \dots, n\}$, 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. We also study the variants of Minimum and Selection in which we do not need to determine the minimum or i-th smallest value, but only an element (or all elements) with that value; we call the corresponding variants MinimumElement and ElementSelection.

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. If $\mathrm{opt}_k > \log m$, we can set $\varepsilon = \sqrt{(\log m)/\mathrm{opt}_k}$ to get a bound of $2\mathrm{opt}_k + \mathrm{O}(\sqrt{\mathrm{opt}_k \log m})$ rounds.) Interestingly, our algorithm follows a non-obvious approach that is reminiscent of primal-dual algorithms, but no linear programming formulation features in the analysis. We then modify this algorithm to solve MinimumElement; the algorithm requires at most $(12+\varepsilon)\cdot\mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon^2}\cdot \lg m\right)$

rounds, for every $0 < \varepsilon < 1$. (If $\mathrm{opt}_k > \log m$, we can set $\varepsilon = ((\log m)/\mathrm{opt}_k)^{1/3}$ to get a bound of $12\mathrm{opt}_k + \mathrm{O}(\mathrm{opt}_k^{2/3}(\log m)^{1/3})$ rounds.) 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, 18, 28]: 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 addressing the minimum spanning tree problem in the model of explorable uncertainty with k queries per round.

Another motivation for solving these problems for multiple possibly overlapping sets comes from distributed database caches [31], where one wants to answer database queries using cached local data and a minimum number of queries (or rounds 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. For example, the database might contain employee records, with only the salaries being uncertain in the local database cache. Answering the queries “who is the female employee with the highest salary”, “who is the black employee with the highest salary”, and “who is the employee with the highest salary among all employees under the age of 30” then translates into identifying the employee with maximum salary in each of the three potentially overlapping sets (the set of female employees, the set of black employees, and the set of employees under the age of 30). This corresponds to the problem of identifying the minimum elements in potentially overlapping sets (via the above-mentioned equivalence of the problem of identifying the maximum element and the problem of identifying the minimum element). Another example would be a database containing air pollution levels for cities, with the current air pollution level being an uncertain field in the local database cache. Answering the queries “output a list of cities in the North of England ordered by current air pollution level” and “output a list of cities in the UK with population above 500,000, ordered by current air pollution level” then corresponds to the problem of sorting potentially overlapping sets (the set of cities in the North of England, and the set of cities in the UK with population above 500,000). Answering such database queries with a minimum number of rounds of queries for exact values to the master database corresponds to the MinimumElement and Sorting problems we consider in this paper.

For the Selection problem, we obtain a 2-round-competitive algorithm, and we note that the same algorithm uses at most $2 \cdot \mathrm{opt}_k + 1$ rounds if we want to solve ElementSelection. 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 [22], and that this is best possible.

We also discuss the relationship between our model and another model of parallel queries proposed by Meißner [29], 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 [24]. 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 [31] 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 [25] 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.* [17] 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 [16], network verification [4], minimum spanning tree [11, 14, 18, 28], cheapest set and minimum matroid base [13, 30], linear programming [27, 32], traveling salesman [34], knapsack [20], 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.* [21] 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 uncertain data has received some attention recently. Halldórsson and de Lima [22] 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 values are revealed [33].

The idea of performing multiple queries in parallel was also investigated by Meißner [29]. 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 Sect. 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 Sect. 2. Sections 3, 4 and 5 are devoted to the sorting, minimum and selection problems, respectively. In Sect. 6, we discuss the relationship between the model we study and the model of Meißner for parallel queries [29]. We conclude in Sect. 7.

2 Preliminaries and Definitions

Throughout the paper, we write lg for $\log_2$. For the problems we consider, the input consists of a set of n continuous uncertainty intervals $\mathcal{I} = \{I_1, \ldots, I_n\}$ on 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 $\ell_i = u_i$, in which case $I_i = \{v_i\}$, so trivial intervals never need to be queried. For some problems we 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* $\left\lceil (\mathrm{opt}_1 + w)/k \right\rceil \leq \mathrm{opt}_k + \lceil w/k \rceil$.

The problems we consider are Minimum, MinimumElement, Sorting, Selection and ElementSelection. For Minimum, MinimumElement 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 and MinimumElement, 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, and for Minimum we are also required to output the value of that element.[1] 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 and ElementSelection, we are given a set $\mathcal{I}$ of n intervals and an index $i \in \{1, \dots, n\}$. For Selection, 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^*; for ElementSelection, we only need to output an element whose precise value is the i-th smallest. 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 Fig. 1. In Fig. 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 Fig. 1b we have a symmetric situation. In Fig. 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 Fig. 1d it is 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. The same argumentation applies to MinimumElement. 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 Fig. 1b, d 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.

[1] Most of the literature in this area is devoted to problems in the form of MinimumElement. Returning the precise minimum value, however, is also an important problem, as discussed in [28, Section 7] for the minimum spanning tree problem.

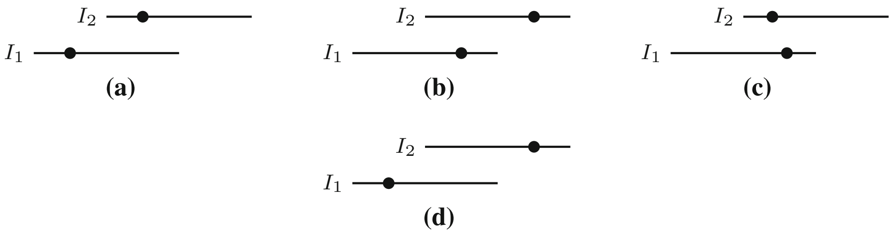

Fig. 1 Example of SORTING for two intervals and the possible realizations of the precise values. We have that $\text{opt}_1 = 1$ in **a**, **b**, **d**, and $\text{opt}_1 = 2$ in **c**.

This problem was addressed for $k = 1$ in [22, 24, 29]; 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 [22]:

- 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 [26].

We adapt the 2-query-competitive algorithm for SORTING by Halldórsson and de Lima [22] 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* [22] *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 [19], 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 needs to query only one interval in each pair, while we can force any deterministic algorithm to query both of them (cf. the configurations shown in Fig. 1a, b). 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 [29], 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 sorting 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 [29] 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 [21].

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).[2] 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.

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 (first for the Minimum problem and then for the MinimumElement problem), 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 $W^{-1}(x) = \Theta(x/\lg x)$ (see, e.g., [23, Theorem 2.7]).

Throughout this section, we assume w.l.o.g. that the optimum must make at least one query in each set (otherwise, 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. For an active set S, consider the non-trivial intervals in $S \backslash \mathcal{P}$ ordered by non-decreasing left endpoints. If the first i of those intervals have already been added to Q in the present round but the $(i + 1)$-th interval has not yet been added to Q, we say that the *Q-prefix length* of S is i. Note that, if the Q-prefix length of S is i, this says nothing about whether the $(i + j)$-th interval for $j \geq 2$ is in Q or not. 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 Q-prefix length. We call this the *balanced* algorithm, and denote it by BAL. We give an example of its execution in Fig. 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 four elements in S_3. It can query these 10 elements in two rounds. Since the algorithm picks an arbitrary active set with minimum Q-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.

[2] All our algorithmic results for the Minimum problem extend to inputs with arbitrary closed, open and half-open intervals if we require that the algorithm must also determine *all* elements whose value equals the minimum. This is because for this problem variant the optimal set of queries for each set is a prefix, and our algorithms only require this property.

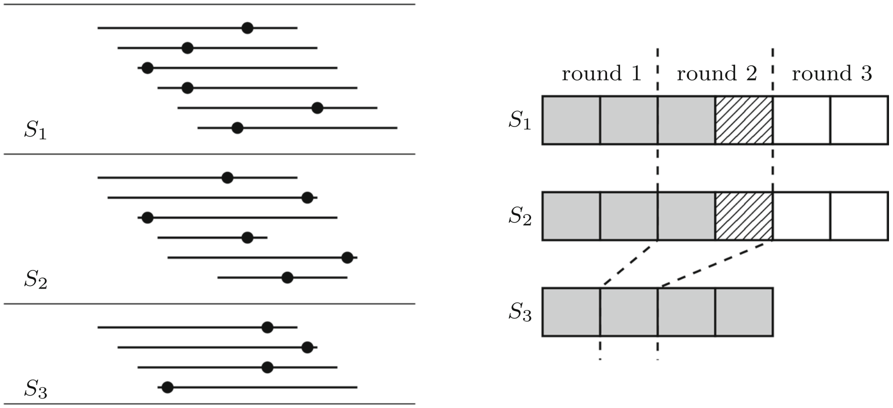

Fig. 2 Possible execution of BAL for $m = 3$ disjoint sets and $k = 5$. The three disjoint sets of intervals are shown on the left. On the right, each interval is represented by a box, with the i-th box of a set corresponding to the interval with i-th smallest left endpoint in that set. The optimum solution 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

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. Let c be a multiple of k. We have $m = c \cdot (k - 1)$ sets, which are divided in c groups with $k - 1$ sets. For $i = 1, \ldots, c$, the sets in groups $i, \ldots, c$ share the i leftmost elements. Furthermore, each set has one extra element which is unique to that set. The precise values are such that each set in the i-th group is solved after querying the first i elements. We give an example in Fig. 3 with $k = 3$ and $c = 3$. If we let BAL query the intervals in the order given by the indices, it is easy to see that it queries $c \cdot k$ intervals, while the c intervals that are shared by more than one set are enough to solve all sets. In particular, note that BAL 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.

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. These variables b_i are set to 0 at the beginning of the computation of the query set of a round. 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 $e \in \mathcal{I} \backslash Q$, let F_e contain the indices of the sets that have e as their leftmost unqueried

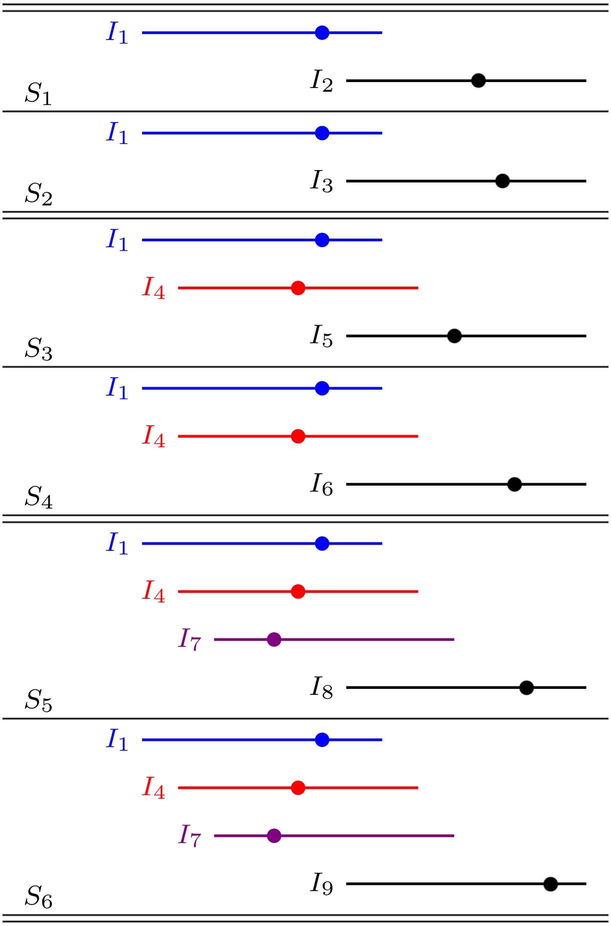

Fig. 3 Bad instance for BAL with overlapping sets, with $k = 3$ and $c = 3$. BAL will query the following rounds: $\{I_1, I_2, I_3\}$, $\{I_4, I_5, I_6\}$, $\{I_7, I_8, I_9\}$. It is enough to query $\{I_1, I_4, I_7\}$

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 $i \in F_e$. (If several elements e satisfy $\sum_{i \in F_e} b_i = 1$ at the same time, they are processed in this way one by one in arbitrary order.) 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/γ active sets are solved in that round, where a is the number of active sets at the start of the round and $\gamma = (2(1+\varepsilon) + \sqrt{2\varepsilon^2 + 4\varepsilon + 4})/\varepsilon$. A round that is not ε-good is called ε-*bad*.

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 $\mathcal{I} \backslash Q$ do
            foreach $e \in \mathcal{I} \backslash Q$ do
                $F_e \leftarrow \{i \mid e$ is the leftmost unqueried element from $\mathcal{I} \backslash Q$ in $S_i\}$;
            increase all $b_i$ simultaneously at the same rate until there is an unqueried element
            $e \in \mathcal{I} \backslash Q$ 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 $Q$;
```

Note that $\gamma > 2$ for any $\varepsilon > 0$. The choice of γ in Definition 4.1 is motivated as follows: We will show in the proof of the following lemma that the round following an ε-bad round must make at least $\frac{(\gamma-2)(\gamma-1)-\gamma}{\gamma(\gamma-2)} \cdot k$ useful queries. The value of γ in Definition 4.1 has been chosen in such a way that this expression evaluates to $2k/(2+\varepsilon)$. As a consequence, the ε-bad round and the round following it make at least $k/(2+\varepsilon)$ useful queries on average. This will lead to the term $(2+\varepsilon)\mathrm{opt}_k$ in the overall bound on the number of query rounds that we prove further down (Theorem 4.3).

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 $s < a/\gamma$ 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, which can be seen as follows: As e is a wasted query and S_j is not solved, there must be an interval in S_j that comes before e (in the ordering by non-decreasing left endpoints) and was not queried yet; hence, j cannot have been in F_e. Therefore, we do not charge e to S_j.

The total number of wasted queries is therefore bounded by $T \cdot s$, as these queries are paid for by the s sets solved in this round. As the number of wasted queries in an

ε-bad round is larger than $k/2$, we therefore have $T \cdot s > k/2$. As $s < a/\gamma$, we get $k/2 < Ta/\gamma$, so $T > (\gamma/2) \cdot (k/a)$.

Call a surviving set S_i *rich* if $b_i > k/a$ when the computation of Q is completed. A surviving set that is not rich is called *poor*. Note that a poor set must have spent at least an amount of $(\gamma/2 - 1) \cdot (k/a) > 0$, as its total budget would be at least $T > (\gamma/2) \cdot (k/a)$ 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 $\frac{k/2}{(\gamma/2-1)\cdot(k/a)} = a/(\gamma - 2)$. As there are more than $(1 - 1/\gamma) \cdot a$ surviving sets and at most $a/(\gamma - 2)$ of them are poor, there are at least $(1-1/\gamma) \cdot a - a/(\gamma-2) = ((\gamma-2)(\gamma-1)-\gamma)/(\gamma(\gamma-2)) \cdot a = 2a/(2+\varepsilon) > 0$ 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 used 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 $1/\gamma$ 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 $1/\gamma$ of the active sets is bounded by $\lceil \log_{\gamma/(\gamma-1)} m \rceil = \mathrm{O}\left(\frac{1}{\varepsilon} \cdot \lg m\right)$, since $1/\left(\lg \frac{\gamma}{\gamma-1}\right) < 5/\varepsilon$ for $0 < \varepsilon < 1$. In every round where the algorithm does not solve at least a fraction of $1/\gamma$ 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 Element Problem with Arbitrary Sets

We now consider the MINIMUMELEMENT problem, in which we want to find the minimum element of each set, but we do not need to output the corresponding value.

We assume that, for every set $S_i \in \mathcal{S}$, any two elements $f, g \in S_i$ satisfy $I_f \cap I_g \neq \emptyset$; otherwise, the element with higher left endpoint is clearly not the minimum in S_i, so we can remove it from S_i.

First let us consider the case where $k = 1$ and we have a single set. Here, querying the intervals in the order of left endpoints until the problem is solved may use up to $\mathrm{opt}_1 + 1$ queries [21]. The reason why we may not obtain an optimum solution is that in some instances the optimal solution can identify the minimum element without querying it. For example, if the set consists of the two intervals $I_1 = (1, 5)$ and $I_2 = (3, 8)$ with values $v_1 = 4$ and $v_2 = 7$, then querying I_2 is sufficient to determine that I_1 is the minimum element. Note that any interval that contains the minimum value but is not the minimum element must be queried in any solution.

For multiple sets, the problem is also easy when $k \leq 2$ because of the following proposition, which is implicit in previous work (see, e.g., [14, 21, 24]).

Proposition 4.4 *The leftmost unqueried interval of a set $S_i \in \mathcal{S}$ and any other unqueried interval in S_i constitute a witness set* [24].

Proof Let $I_1 = (\ell_1, u_1)$ be the leftmost unqueried element of S_i, and let I_2 be any other unqueried interval in S_i. Note that I_2 overlaps I_1 because we assume that all intervals that do not overlap I_1 have been removed from S_i. Assume there is a solution that queries neither I_1 nor I_2. Then it is impossible to determine whether I_1 is the minimum element (this happens if v_1 is closer to ℓ_1 than the value of any other element of S_i) or I_1 is not the minimum element (this happens if v_1 is larger than v_2, which is possible because I_1 and I_2 overlap). Thus, every solution must query at least one interval of $\{I_1, I_2\}$. □

Thus there is a 2-query-competitive algorithm, which can be turned into a 2-round-competitive algorithm for $k \leq 2$, and it is easy to see that this is the best possible.

If $k > 2$ and the sets in $\mathcal{S}$ are disjoint, then MinimumElement can be reduced to the Minimum problem while losing a factor of 2 in the round-competitive ratio: We waste at most one query in each set and, since every set is initially active, we make at least one useful query in each set, and those are all distinct.

However, if the sets in $\mathcal{S}$ overlap, this reduction does not work. So let us study this case in more detail. Before we proceed, let us observe the following: Call the prefix P of S_i the *minimum solving prefix* of S_i if querying P solves S_i while no shorter prefix of S_i, when queried, solves S_i. We will show later that, in order to solve a set S_i, we must either (a) query all intervals except the minimum (this option exists only if all intervals except the minimum have their precise values to the right of the interval of the minimum element) or (b) query at least all intervals of the minimum solving prefix of S_i.

We give an algorithm for MinimumElement that requires at most $(12 + \varepsilon) \cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon^2} \cdot \lg m\right)$ rounds for every $0 < \varepsilon < 1$, which we describe next. The pseudocode for the whole execution is shown in Algorithm 2. We assume that each set in $\mathcal{S}$ is ordered by non-decreasing left endpoints.

We begin by selecting a set Q of elements that are *known mandatory*, i.e., elements that are clearly part of any feasible query set. An element e is known mandatory if

Algorithm 2: Algorithm for the MinimumElement problem

```
Data: Family $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of the ground set $\mathcal{I}$
Result: List $\mathcal{Q}$ of sets of elements queried in each round, ordered by round, each set of size at most $k$
begin
    $\mathcal{Q} \leftarrow ()$;
    let $b_{ie} \leftarrow 0$, for each $e \in \mathcal{I}$ and $S_i \in \mathcal{S}$;
    while there is some active set in $\mathcal{S}$ do
        $\mathcal{B} \leftarrow \emptyset$;
        $Q \leftarrow$ known mandatory elements;
        if $|Q| \geq k$ then
            $Q \leftarrow$ arbitrary subset of $Q$ with size $k$;
        else
            $M \leftarrow$ maximal matching in the star union graph;
            if $|Q| + 2 \cdot |M| \geq k$ then
                $M \leftarrow$ arbitrary subset of $M$ with size $\left\lfloor \frac{k-|Q|}{2} \right\rfloor$;
            $Q \leftarrow Q \cup \bigcup_{uv \in M} \{u, v\}$;
        if $|Q| < k$ then
            while $|Q| < k$ and there are unqueried elements in $\mathcal{I} \backslash Q$ do
                foreach active set $S_i \in \mathcal{S}$ do
                    $W_i \leftarrow$ the first and second unqueried element (if any) from $\mathcal{I} \backslash Q$ in $S_i$;
                increase simultaneously at the same rate all $b_{ie}$ where $S_i$ is active and $e \in W_i$, until
                there is some unqueried element $f \in \mathcal{I} \backslash Q$ with $\sum_{j=1}^{m} b_{jf} = 1$;
                let $\alpha$ be the amount by which those variables increased (same value for all);
                foreach active set $S_i \in \mathcal{S}$ do
                    if $W_i \neq \emptyset$ then
                        $\mathcal{B} \leftarrow \mathcal{B} \cup \{(W_i, i, \alpha)\}$/* Remember the set $W_i$ containing
                            the elements $e$ of $S_i$ for which $b_{ie}$ was increased
                            by $\alpha$ in this iteration                                  */
                        ;
                $Q \leftarrow Q \cup \{f\}$;
        query $Q$, append $Q$ to $\mathcal{Q}$;
        foreach $(W, i, \alpha) \in \mathcal{B}$ do
            if $S_i$ is solved, $W$ has an unqueried element and $W$ is not a witness set for $S_i$ then
                foreach unqueried element $e \in W$ do
                    $b_{ie} \leftarrow b_{ie} - \alpha$;
    return $\mathcal{Q}$;
```

there is a set S_i that is not solved, has e as its leftmost unqueried element, and e contains another interval in S_i or the value of a previously queried interval in S_i [15]. If known mandatory elements are not enough to fill the round, then we build a *star union graph* G, which has a vertex for each element in $\mathcal{I}$ and, for each set S_i, an edge between the leftmost unqueried interval of S_i and every other unqueried interval in S_i. (Every such edge corresponds to a witness set by Proposition 4.4.) We then compute a maximal matching in $G[\mathcal{I} \backslash Q]$, i.e., we do not consider edges where an endpoint is a known mandatory element.

If this is still not enough to fill the round, then we proceed in a similar fashion to the algorithm in the previous section, increasing the budget of each set until there is enough shared budget to include an element in the query set. To simplify the argument, here we have a budget variable b_{ie} for each element $e \in \mathcal{I}$ and each set $S_i \in \mathcal{S}$. The difference is that now we increase a separate budget for the leftmost and the *second leftmost* unqueried interval in each set: The idea is that we want to increase the budget simultaneously for such pairs of elements that constitute a witness set. However, we do not know for sure if all such pairs are witness sets, so we maintain a history $\mathcal{B}$ of the budget increases; i.e., we add an entry (W, i, α) to $\mathcal{B}$ if, in the given iteration of the loop in Lines 16–24, S_i increased the budget of the elements in W by α. Another difference is that we *carry* the unused budget in a given round to the following rounds; we do not reset the budgets to zero once the round is complete.

After the elements selected in the given round are queried, we look at each entry $(W, i, \alpha) \in \mathcal{B}$ and subtract the budget α from unqueried elements in W if we realize that W is not a witness set for S_i. We will prove in Lemma 4.5 below that such a test is possible; note that each set considered has size 1 or 2, and that sets of size 2 satisfy the condition in the lemma due to the behavior of the algorithm.

Now let us proceed to the analysis of the algorithm. Fix an optimal solution OPT_1. We say that a query performed by the algorithm is *useful* if it is in OPT_1; otherwise it is *wasted*.

We show that, for each set S_i, any query set that solves S_i must either contain the prefix of S_i that consists of all intervals that contain the precise minimum value, or it must contain all elements of S_i except for a single one (which is not the last in S_i). In the latter case, the element e_i that need not be queried is the minimum of S_i, and all other intervals that contain the precise value of e_i have to be queried by any solution (i.e., these intervals have to be queried in any solution that queries e_i and in any solution that does not query e_i). Either way, as alluded to earlier, each set has a minimum solving prefix, such that the set is solved after querying this prefix. As OPT_1 solves S_i, either $\mathrm{OPT}_1 \cap S_i$ is a superset of the minimum solving prefix, or every interval in the minimum solving prefix except for the minimum is also in OPT_1. Therefore, as stated earlier, in order to solve S_i, we must either (a) query all intervals except the minimum (this option exists only if all intervals except the minimum have their precise values to the right of the interval of the minimum element) or (b) query at least all intervals of the minimum solving prefix of S_i. This shows that any query set that solves S_i must contain (at least) all elements that are not the minimum element and that belong to the minimum solving prefix. Therefore, the only queries in S_i that can potentially be wasted queries are the query of the minimum element and the queries of elements that come after the minimum solving prefix.

Lemma 4.5 *Consider a set $S_i \in \mathcal{S}$ that is solved in a round r, and a subset $W \subseteq S_i$ with $|W| \in \{1, 2\}$ whose budget was increased by S_i in round r, and suppose W contains an unqueried element at the end of round r. Furthermore, if $|W| = 2$, then suppose that all elements in S_i with smaller left endpoint than the leftmost element of W were queried in rounds $1, \ldots, r$. In both cases ($|W| = 1$ or $|W| = 2$), one can determine in Line 27 of Algorithm 2 whether W is a witness set for S_i.*

Proof If $|W| = 1$, then clearly W is not a witness set, since we assume W contains an unqueried element and S_i is solved.

Now assume $|W| = 2$ and write $W = \{f, g\}$, with f preceding g in the ordering by left endpoints. We divide the proof into two cases. Let e_i be the minimum element of S_i.

1. *The set S_i was solved by querying all elements except e_i.* Then clearly $e_i \in W$. Since W consists of the minimum element in S_i and some other element of S_i, then it is a witness set.
2. *The set S_i was solved by querying its minimum solving prefix, plus possibly additional elements after that prefix.* Note that it is possible to determine the minimum solving prefix, even if not all elements of S_i were queried. If both f and g come after the minimum solving prefix, then W is not a witness set, since we can solve S_i without querying W. Otherwise, since W contains an unqueried element, f must be in the minimum solving prefix, but g is not. If $f \neq e_i$, then f is in every feasible query set, because every feasible query set either queries all of $S_i \backslash \{e_i\}$ or the minimum solving prefix, so W is a witness set; otherwise, W consists of e_i plus another element of S_i, so it is a witness set in this case, too.

Summing up, the answer for the witness set test in Line 27 is NO if $|W| = 1$ or both elements come after the minimum solving prefix of S_i, and YES otherwise. □

Lemma 4.6 *Consider a set S_i and a round at the end of which S_i remains unsolved. At the beginning of every iteration of the loop in Lines 16–24, the first and second unqueried elements from $\mathcal{I} \backslash Q$ in S_i (or the first element if there is only one such element) constitute a witness set.*

Proof If there is only one unqueried element from $\mathcal{I} \backslash Q$ in S_i, then this element must be in any feasible query set, since S_i survives the round but all other elements of S_i have been queried by the end of the round. So let us assume that there are at least two unqueried elements from $\mathcal{I} \backslash Q$ in S at the beginning of the iteration.

Let $e_1, \ldots, e_{|S_i|}$ be the elements of S_i in the order considered by the algorithm (with non-decreasing left endpoints), and let e_t be the first element of S_i that remains unqueried by the end of the round.

Let f and g be the first and second element, respectively, from $\mathcal{I} \backslash Q$ in S_i at the beginning of the iteration. As S_i is not solved in the round and all elements before f are queried by the end of the round, f must be a member of the minimum solving prefix of S_i. If f is not the minimum element of S_i, then f is in every feasible query set for S_i, and hence $\{f, g\}$ is a witness set. If f is the minimum element of S_i, then $\{f, g\}$ contains the minimum element of S_i and another interval of S_i that overlaps the minimum element, so $\{f, g\}$ is again a witness set. □

When a query e is added to Q, we say that it is *charged* to the sets S_i with $b_{ie} > 0$. We also say that the set S_i *pays a charge* of b_{ie} for e in the round in which e is queried. Note that, for an element selected in Line 6, 8 or 13, it may be that not all of its cost is charged to the sets; we then just say that the remaining cost $(1 - \sum_{i=1}^{m} b_{ie})$ is charged to the whole algorithm.

Suppose that the algorithm uses R rounds to solve the problem, and let $b_{ie}^{(0)} = 0$ and $b_{ie}^{(r)}$ be the value of b_{ie} at the end of round r, for $r = 1, \dots, R$; to simplify notation, we write $b_{ie} = b_{ie}^{(R)}$. For each element e and each set S_i with $e \in S_i$, we divide b_{ie} into two parts, a *bounded charge* b_{ie}^- and an *unbounded charge* b_{ie}^+; the idea is that bounded charges are associated with witness sets, while unbounded charges will be used to prove a similar result to that of Lemma 4.2. If e is selected in Line 12 or is a useful query (note that all elements selected in Lines 6 and 8 are useful), then we set $b_{ie}^- = b_{ie}$ and $b_{ie}^+ = 0$ for every $S_i \in \mathcal{S}$ with $e \in S_i$; in that case, if $\sum_{i=1}^m b_{ie} < 1$, then we also say that the amount $(1 - \sum_{i=1}^m b_{ie})$ is bounded, even though we do not charge it to any specific set. If e is a wasted query selected in Line 24, then let r_e be the round in which e is queried and let r_i be the round in which S_i is solved. If $r_e \neq r_i$, then we also set $b_{ie}^- = b_{ie}$ and $b_{ie}^+ = 0$. If $r_e = r_i$, then we set $b_{ie}^- = b_{ie}^{(r_i-1)}$ and $b_{ie}^+ = b_{ie} - b_{ie}^-$. Note that, if e is not the first or second unqueried element of S_i at the beginning of round r_i, we have $b_{ie}^{(r_i-1)} = 0$, so $b_{ie}^- = 0$. This definition ensures that, if a portion of the budget is carried between rounds, then it will only consist of bounded charges.

Lemma 4.7 *The total bounded charge is at most* $3 \cdot |\mathrm{OPT}_1|$.

Proof Let $U \subseteq \mathcal{I}$ be the set of useful queries performed by the algorithm; note that all elements selected in Lines 6 and 8 are in U. The total bounded charge of useful queries is clearly at most $|\mathrm{OPT}_1|$, and so $\sum_{i=1}^m \sum_{e \in U \cap S_i} b_{ie} \leq |\mathrm{OPT}_1|$.

Moreover, by definition of the algorithm the total budget accumulated by any element $e \in \mathcal{I}$ is at most 1, no matter whether the element is queried or not. Hence, we also have $\sum_{i=1}^m \sum_{e \in \mathrm{OPT}_1 \cap S_i} b_{ie} \leq |\mathrm{OPT}_1|$.

Each edge selected in Line 12 is a witness set (and thus contains at least one element of OPT_1), so the total bounded charge of wasted elements selected in Line 12 is at most $|\mathrm{OPT}_1|$.

It remains to show that the total bounded charge spent on wasted queries in Line 24 is at most $|\mathrm{OPT}_1|$. Let $V \subseteq \mathcal{I}$ be the set of queries wasted in Line 24. We show that $\sum_{e \in V \cap S_i} b_{ie}^- \leq \sum_{e \in \mathrm{OPT}_1 \cap S_i} b_{ie}$ for every set $S_i \in \mathcal{I}$, from which we immediately obtain that $\sum_{i=1}^m \sum_{e \in V \cap S_i} b_{ie}^- \leq |\mathrm{OPT}_1|$.

Let $e \in V$ be a wasted query, r_e be the round in which e is queried and r_i be the round in which S_i is solved. Due to Lemma 4.6, it holds that $\sum_{e \in V \cap S_i} b_{ie}^{(r_i-1)} \leq \sum_{e \in \mathrm{OPT}_1 \cap S_i} b_{ie}^{(r_i-1)}$ since, in a round that S_i survives, every time the bounded charge is increased for a wasted query, the same amount of budget is increased for a useful query.

We then claim that $\sum_{e \in V \cap S_i} \left(b_{ie}^- - b_{ie}^{(r_i-1)} \right) \leq \sum_{e \in \mathrm{OPT}_1 \cap S_i} \left(b_{ie} - b_{ie}^{(r_i-1)} \right)$. If $r_e \leq r_i$, then by definition $b_{ie}^- - b_{ie}^{(r_i-1)} = 0$. If $r_e > r_i$, then note that in Line 29 we subtract the budget that was increased for pairs that are not witness sets; in conclusion, every time the bounded charge of e is increased and is not subtracted at the end of round r_i, the same amount of budget is increased for a useful query. So the claim holds, and thus $\sum_{e \in V \cap S_i} b_{ie}^- \leq \sum_{e \in \mathrm{OPT}_1 \cap S_i} b_{ie}$, as required. □

Now let us limit the unbounded charge paid by the algorithm.

Definition 4.8 Let $\varepsilon > 0$. A round of Algorithm 2 is ε-*good* if at least a total bounded charge of $k/4$ is paid in that round, or if at least a/γ active sets are solved in that round, where a is the number of active sets at the start of the round and $\gamma = 13/3 + 20/\varepsilon$. A round that is not ε-good is called ε-*bad*.

For the remainder of this section, let $\beta = 2 + \varepsilon/6$ and $\delta = 1 - \frac{1}{\gamma} - \frac{2}{3\gamma-8} - \frac{2}{\beta} = \frac{24\varepsilon^2}{5(\varepsilon+12)(13\varepsilon+60)} < \frac{24}{65}$.

Lemma 4.9 *If a round r is ε-bad, then Algorithm 2 will pay a bounded charge of at least k/β in round $r+1$, or will make k useful queries in round $r+2$, or will solve at least $a \cdot \delta$ sets in rounds $r+1$ and $r+2$, where a is the number of active sets at the start of round r.*

Proof The following round, i.e., round $r+1$, starts with a set of known mandatory elements (let us call it N) and a maximal matching M. Each edge of M is between the first element of a surviving set and another element of that surviving set. If $|N| + 2 \cdot |M| \geq k/\beta$, then the algorithm pays a bounded charge of at least k/β in round $r+1$ and the first option of the lemma holds.

So let us assume that $|N| + 2 \cdot |M| < k/\beta$. Let a denote the number of active sets at the start of round r. Let s be the number of sets that are solved in round r; note that $s < a/\gamma$ because round r is ε-bad.

Let Z be the number of iterations of the loop in Lines 16–24 in round r, and let α_z be the value of α in Line 19 at iteration z, for $z = 1, \dots, Z$. Let $T = \sum_{z=1}^{Z} \alpha_z$; if the simultaneous increase of the budget variables is interpreted as time passing, then T corresponds to the point in time when the computation of the set Q in round r was completed following the execution of Line 14. Note that any set increases its total budget in round r by at most $2T$ (since we may increase the budget of two variables simultaneously). For any set that is active at the beginning of round r and does not have all its elements included in Q at the end of the round, the total increase in budget is at least T.

Let Q be the set of queries that Algorithm 2 makes in round r. By definition, if $b_{ie}^{+} > 0$ with $e \in Q$, then S_i is solved in this round. Moreover, remember that, by definition, the budget that was carried from the previous round is always used on bounded charges. The total unbounded charge is therefore bounded by $2T \cdot s$, as this amount is paid for by the s sets solved in this round. As the total unbounded charge in an ε-bad round is larger than $3k/4$, we therefore have $T \cdot s > 3k/8$. As $s < a/\gamma$, we get $3k/8 < Ta/\gamma$, so $T > (3\gamma/8) \cdot (k/a)$.

Let J be the set of unqueried elements at the beginning of round r. Call a surviving set S_i *rich* if $\sum_{e \in J \setminus Q} b_{ie} > k/a$ when the computation of Q is completed. A surviving set that is not rich is called *poor*. Note that a poor set must have spent at least an amount of $(3\gamma/8 - 1) \cdot (k/a) > 0$, as its total unused budget at the end of the round would be at least $T > (3\gamma/8) \cdot (k/a)$ if it had not bought any queries. As the poor sets have paid for a total bounded charge of less than $k/4$ (as a charge of less than $k/4$ is bounded in round r), the number of poor sets is bounded by $\frac{k/4}{(3\gamma/8-1)\cdot(k/a)} = \frac{2}{3\gamma-8} \cdot a$. As there are more than $(1 - \frac{1}{\gamma}) \cdot a$ surviving sets and at most $\frac{2}{3\gamma-8} \cdot a$ of them are poor, there are at least $\left(1 - \frac{1}{\gamma} - \frac{2}{3\gamma-8}\right) \cdot a > 0$ surviving sets that are rich.

Let S_i be a surviving rich set, and let f and g be the first and second elements from $J \setminus Q$ in S_i at the end of round r. Note that f and g are the only elements in $S_i \cap (J \setminus Q)$ with $b_{ie} > 0$ at the end of the round. Also, since f comes before g in the order of left endpoints, we have that $b_{if} \geq b_{ig}$. Therefore, $b_{if} + b_{ig} = \sum_{e \in J \setminus Q} b_{ie} > k/a$ implies $b_{if} > k/(2a)$.

Let f be an element that is the first unqueried element of a rich surviving set at the end of round r. If f was the first unqueried element of more than $2a/k$ rich surviving sets, those sets would have been able to pay for f (because their total remaining first-element budget would be greater than $(k/(2a)) \cdot (2a/k) = 1$) before the end of round r, a contradiction to f not being included in Q.

Since $|N| + 2 \cdot |M| < k/\beta$, this means that at most $k/\beta \cdot 2a/k = 2a/\beta$ of the surviving rich sets have their first element contained in N or in a matching edge in M. Thus, there are at least $\left(1 - \frac{1}{\gamma} - \frac{2}{3\gamma - 8} - \frac{2}{\beta}\right) \cdot a = a \cdot \delta > 0$ surviving rich sets whose first element is not in a matching edge. The reason why no edge with that first element can be added to M must be that all remaining elements of that set are already contained in N or in an edge of M. This means that for each of these $a \cdot \delta$ sets, in round $r + 1$ we query at least all elements except the first element. Hence, each of these sets is either solved in round $r + 1$, or its first element becomes a known mandatory query at the end of round $r + 1$; let k' be the number of such elements. If round $r + 2$ makes k useful queries, then the lemma holds; otherwise, all those k' elements are queried in round $r + 2$. This means that each of these $a \cdot \delta$ surviving rich sets is solved in round $r + 1$ or $r + 2$.

In summary, this shows that if round r is not ε-good, at least one of the following three statements holds:

- Round $r + 1$ pays a bounded charge of at least k/β.
- Round $r + 2$ queries k useful elements.
- Rounds $r + 1$ and $r + 2$ together solve at least $a \cdot \delta$ sets. □

Theorem 4.10 *Algorithm 2 uses at most* $(12 + \varepsilon) \cdot \mathrm{opt}_k + \mathrm{O}\left(\frac{1}{\varepsilon^2} \cdot \lg m\right)$ *rounds for every* $0 < \varepsilon < 1$*, where* opt_k *denotes the optimal number of rounds.*

Proof For every round r, one of the following must hold:

(1) The algorithm pays at least a bounded charge of $k/4$ in round r.
(2) The algorithm solves at least a fraction of $1/\gamma$ of the active sets in round r.
(3) The algorithm pays at least a bounded charge of k/β in round $r + 1$.
(4) The algorithm makes k useful queries in round $r + 2$.
(5) In rounds $r + 1$ and $r + 2$, the algorithm solves a fraction of δ of the sets that are active at the beginning of round r.

If condition (1) holds for a round r, then the bounded charge paid in that round is at least $k/4$. If condition (3) holds for a round r, then the average bounded charge paid in rounds r and $r + 1$ is at least $k/(2\beta)$; note that $k/(2\beta) \leq k/4$. If condition (4) holds for a round r, then the average bounded charge paid in rounds r, $r + 1$ and $r + 2$ is at least $k/3 \geq k/(2\beta)$, because the k useful queries in round $r + 2$ are all paid for by bounded charge. Therefore, the number of rounds in which condition (1), (3) or (4) holds (where we consider two consecutive rounds for condition (3) and three consecutive rounds for condition (4)) is at most $(2\beta/k)\cdot 3\cdot|\mathrm{OPT}_1| \le (12+\varepsilon)\cdot \mathrm{opt}_k$ by Lemma 4.7.

Furthermore, the number of rounds in which condition (2) holds is bounded by $\lceil \log_{\gamma/(\gamma-1)} m\rceil = \mathrm{O}\left(\frac{1}{\varepsilon}\cdot\lg m\right)$, since $1/\left(\lg\frac{\gamma}{\gamma-1}\right) < 20/\varepsilon$ for $0<\varepsilon<1$. The number of rounds in which condition (5) holds (where we always count three consecutive rounds starting with the round in which condition (5) holds) is bounded by $3\cdot\lceil\log_{1/(1-\delta)} m\rceil = \mathrm{O}\left(\frac{1}{\varepsilon^2}\cdot\lg m\right)$, since $1/\left(\lg\frac{1}{1-\delta}\right) < 200/\varepsilon^2$ for $0<\varepsilon<1$. □

4.3 The Minimum Problem with Disjoint Sets

We now consider the Minimum problem in the case where $k\ge 2$ 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 4.11 $\mathrm{BAL}_k \le \mathrm{opt}_k + \mathrm{O}(\lg\min\{k,m\})$.

Proof First we prove the bound for $m\le k$. Index the sets in such a way that S_i is the i-th set that is solved by BAL, for $1\le i\le m$. 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, $\mathrm{BAL}_k \le \mathrm{opt}_k + \mathrm{O}(\lg m)$.

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 4.11.

Lemma 4.12 *If* $\mathrm{opt}_k = 1$, *then* $\mathrm{BAL}_k \le \mathrm{O}(\lg m/\lg\lg m)$.

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 $1\le i\le R$, 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, $1\le i\le R$. 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 4.13 *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 4.12. 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 4.12 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 $S_j \backslash Q$, insert it into Q, and set $i = j$. The resulting set Q is then queried.

Lemma 4.14 *Assume that* BAL *distributes queries to active sets in a round-robin fashion. If* $\mathrm{BAL}_k \leq \rho$ *for instances with* $\mathrm{opt}_k = 1$, *with* ρ *independent of* k, *then* BAL *is* ρ*-round-competitive for arbitrary instances.*

Proof Let $L = (\mathcal{I}, \mathcal{S})$ be an instance with $\mathrm{opt}_k(L) = t$. Note that $\mathrm{opt}_1(L) \leq tk$. Consider the instance L' which is identical to L except that the number of queries per round is $k' = tk$. Use BAL' to refer to the solution computed by BAL for the instance L' (and also to the algorithm BAL when it is executed on instance L'). Note that $\mathrm{opt}_{k'}(L') = 1$ as $\mathrm{opt}_1(L') = \mathrm{opt}_1(L) \leq tk$ and, therefore, a single round with $k' = tk$ queries is sufficient for making all queries in the optimal query set.

By our assumption, $\mathrm{BAL}'_{k'} \leq \rho$. We claim that this implies $\mathrm{BAL}_k \leq \rho t$. To establish the claim, we compare the situation when BAL' has executed x rounds on L' with the situation when BAL has executed xt rounds on L. We claim that the following two invariants hold for every x:

(1) The number of remaining active sets of BAL is at most that of BAL$'$.
(2) BAL has made at least as many queries in each active set as BAL$'$.

For a proof of these invariants, note that BAL$'$ and BAL distribute queries to sets in the same round-robin order, the only difference being that BAL performs a round of queries whenever k queries have been distributed, while BAL$'$ only performs a round of queries whenever kt queries have accumulated. Imagine the process by which both algorithms pick queries as if it was executed in parallel, with both of the algorithms choosing one query in each step. The only case where BAL and BAL$'$ can distribute the next query to a different set is when BAL$'$ distributes the next query to a set S_i that is no longer active for BAL (or all of whose non-trivial unqueried elements have already been added by BAL to the set of queries to be performed in the next round). This can happen because BAL may have already made some of the queries that BAL$'$ has distributed to sets but not yet performed. If this happens, BAL will select for the next query an element of a set that comes after S_i in the cyclical order, so it will move ahead of BAL$'$ (i.e., it chooses a query now that BAL$'$ will only choose in a later step). Hence, at any step during this process, BAL either picks the same next query as BAL$'$ or is ahead of BAL$'$. This shows that if the invariants hold when BAL and BAL$'$ have executed xt and x rounds, respectively, then they also hold after they have executed $(x+1)t$ and $x+1$ rounds, respectively. As the invariants clearly hold for $x = 0$, if follows that they always hold, and hence $\mathrm{BAL}_k \leq \rho t$. □

Lemmas 4.12 and 4.14 imply the following.

Corollary 4.15 BAL *is* $\mathrm{O}(\lg m/\lg\lg m)$*-round-competitive.*

Unfortunately, Corollary 4.13 cannot be combined with Lemma 4.14 directly to show that BAL is $\mathrm{O}(\lg k/\lg\lg k)$-round-competitive, because the proof of Lemma 4.14 assumes that ρ is not a function of k. However, we can show the claim using different arguments.

Lemma 4.16 BAL *is* $\mathrm{O}(\lg k/\lg\lg k)$*-round-competitive.*

Proof If $k \geq m$, the lemma follows from Corollary 4.15.

If $k < m$, let L be the given instance and let R_0 be the number of rounds the algorithm needs until the number of active sets falls below $k+1$ for the first time. As the algorithm makes at most one query in each active set in the first R_0 rounds, all queries made in the first R_0 rounds are useful. Let L' be the instance at the end of round R_0. As L' has at most k active sets, BAL is $\mathrm{O}(\lg k/\lg\lg k)$-round-competitive on L' by Corollary 4.15, and it needs at most $\mathrm{O}(\lg k/\lg\lg k)\cdot \mathrm{opt}_k(L') = \mathrm{O}(\lg k/\lg\lg k)\cdot \lceil \mathrm{opt}_1(L')/k\rceil$ rounds to solve L'.

We have that $\mathrm{opt}_1(L) = k\cdot R_0 + \mathrm{opt}_1(L')$, and hence $\mathrm{opt}_k(L) = R_0 + \lceil \mathrm{opt}_1(L')/k\rceil$. Thus,

$$\mathrm{BAL}_k(L) \leq R_0 + \mathrm{O}(\lg k/\lg\lg k)\cdot \lceil \mathrm{opt}_1(L')/k\rceil$$

$$\begin{aligned} &\leq \mathrm{O}(\lg k/\lg\lg k)\cdot(R_0+\lceil \mathrm{opt}_1(L')/k\rceil)\\ &=\mathrm{O}(\lg k/\lg\lg k)\cdot \mathrm{opt}_k(L), \end{aligned}$$

and the claim follows. □

The following theorem then follows from Corollary 4.15 and Lemma 4.16.

Theorem 4.17 BAL *is* $\mathrm{O}(\lg\min\{k,m\}/\lg\lg\min\{k,m\})$*-round-competitive.*

We note that, if we use BAL to solve the MinimumElement problem on disjoint sets, then we only lose an extra factor of 2 in the round-competitive factor. This is because we waste at most one extra query per set (the minimum element of that set) and, since every set is initially active, each set contains at least a useful query. This is still asymptotically optimal given the lower bounds in the next section.

4.4 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 4.18 *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 $\varepsilon=1/M^{M+2}$. 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 for the algorithm. The optimum must make the same queries to these active sets that the algorithm made in the current round, and there at most k/M such 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 set of queries then consists of the at most k/M queries that the algorithm makes in surviving sets in each of the first $M-1$ rounds, plus a single query in each of the $m=M^M$ sets to solve it. 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 4.19 *No deterministic algorithm* ALG *attains* $\mathrm{ALG}_k \le \mathrm{opt}_k + \mathrm{o}(\lg \min\{k, m\})$ *for all* k.

Proof Let $k = m$ be an arbitrarily large integer. The intervals of the m sets are chosen as in the proof of Theorem 4.18, 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 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, $1 \le i \le n$. Throughout this section we denote the i-th smallest value in the set of n precise values by v^*.

5.1 Finding the Value v^*

If we only want to find the value v^*, then we can adapt the analysis in [21] to obtain an algorithm that performs at most $\mathrm{opt}_1 + i - 1$ queries, simply by querying the intervals in the order of their left endpoints. This is the best possible and can easily be parallelized in $\mathrm{opt}_k + \left\lceil \frac{i-1}{k} \right\rceil$ rounds. Note that we can assume $i \le \lceil n/2 \rceil$, since otherwise we can consider the i-th largest value problem, noting that the i-th smallest value is the $(n-i+1)$-th largest value. We also assume that every input interval is either trivial or open, since otherwise (if arbitrary closed intervals are allowed) the problem has a lower bound of n on the competitive ratio, using the same instance as presented in [21] for the MinimumElement problem on a single set (included for the sake of completeness in Appendix A).

Let ℓ be the i-th smallest left endpoint, and let u be the i-th smallest right endpoint. Note that any interval I_j with $u_j < \ell$ or $\ell_j > u$ can be discarded (and the value of i adjusted accordingly).

We analyze the algorithm that simply queries the k leftmost non-trivial intervals until the problem is solved.

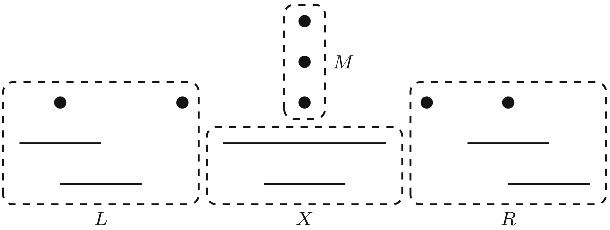

Fig. 4 An illustration of sets M, X, L and R (after the queries in OPT_1 have been executed) in the proof of Theorem 5.1

Theorem 5.1 *For instances of the i-th smallest value problem where all input intervals are open or trivial, there is an algorithm that returns the i-th smallest value v^* and uses at most*

$$\left\lceil \frac{\mathrm{opt}_1 + i - 1}{k} \right\rceil \leq \mathrm{opt}_k + \left\lceil \frac{i-1}{k} \right\rceil$$

rounds.

Proof Let $\mathcal{I}'$ be the set of non-trivial intervals in the input, ordered by non-decreasing left endpoint. We show that the prefix Q of $\mathcal{I}'$ of size $\mathrm{opt}_1 + i - 1$ has the property that, after querying Q, the instance is solved. Given the existence of such a set Q, it is clear that the theorem follows.

Fix an optimum query set OPT_1, and let v^* be the i-th smallest value. After querying OPT_1, assume that there are m trivial intervals with value v^*. Note that $m \geq 1$, since it is necessary to determine the value v^*. Those m intervals are either queried in OPT_1 or already were trivial intervals in the input. We classify the intervals in $\mathcal{I}$ into the following categories:

1. The set M (of size m) consisting of trivial intervals whose value is v^* and of non-trivial intervals that are in OPT_1 and have value v^*;
2. The set X consisting of non-trivial intervals that contain v^* and are not in OPT_1;
3. The set L of intervals that are to the left of v^* or that are in OPT_1 and have a value to the left of v^*;
4. The set R of intervals that are to the right of v^* or that are in OPT_1 and have a value to the right of v^*.

We illustrate this classification in Fig. 4. Note that intervals in L and R may intersect intervals in X, but cannot contain v^*. Let $M^* = M \cap \mathrm{OPT}_1$, $L^* = L \cap \mathrm{OPT}_1$ and $R^* = R \cap \mathrm{OPT}_1$. Note that $X \cap \mathrm{OPT}_1 = \emptyset$, and that every interval in $M \backslash M^*$ is trivial in the input.

We claim that the set $Q = (L \cap \mathcal{I}') \cup X \cup M^* \cup R^*$ is a prefix of $\mathcal{I}'$ in the given ordering (note that $(X \cup M^* \cup R^*) \backslash \mathcal{I}' = \emptyset$), that querying Q suffices to solve the instance, and that $|Q| \leq \mathrm{opt}_1 + i - 1$. Clearly, every interval in $L \cup X \cup M^*$ comes before all the intervals in $R \backslash R^*$ in the ordering considered. It also holds that every interval in R^* comes before all the intervals in $R \backslash R^*$ in the ordering, since otherwise an interval in R^* not satisfying this condition could be removed from OPT_1. Furthermore, querying all intervals in Q is enough to solve the instance, because every interval in $R \backslash R^*$ is to the right of v^*, and the optimum solution can decide the problem without querying them. Thus it suffices to bound the size of Q. Note then that $|L|+|X| \leq i-1$ since, after querying OPT_1, the i-th smallest interval is in M, and any interval in $L \cup X$ has a left endpoint to the left of v^*. Therefore,

$$|Q| \leq |L| + |X| + |M^*| + |R^*| \leq i - 1 + |M^*| + |R^*| \leq \mathrm{opt}_1 + i - 1,$$

which concludes the proof. □

The upper bound of $\left\lceil \frac{\mathrm{opt}_1+i-1}{k} \right\rceil$ is best possible, because we can construct a lower bound of $\mathrm{opt}_1 + i - 1$ queries to solve the problem. It uses the same instance as described in [21] for the problem of identifying an i-th smallest element (but not necessarily finding its precise value). We include a description of the instance for the sake of completeness. Consider $2i$ intervals, comprising i copies of $(0, 5)$ and i copies of $\{3\}$. For the first $i-1$ intervals $(0, 5)$ queried by the algorithm, the adversary returns a value of 1, so the algorithm also needs to query the final interval of the form $(0, 5)$ to decide the problem. Then the adversary sets the value of that interval to 4, and querying only that interval would be sufficient for determining that 3 is the i-th smallest value. Hence any deterministic algorithm makes at least i queries, while $\mathrm{opt}_1 = 1$.

5.2 Finding All Elements with Value *v**

Now we focus on the task of finding v^* as well as identifying all intervals in $\mathcal{I}$ whose precise value equals v^*. For this problem variant, closed intervals do not cause any difficulties. 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. A complete proof is included in Appendix B.

Let us begin by observing that the optimal query set is easy to characterize.

Lemma 5.2 *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.*

Proof If a non-trivial interval I_j containing v^* is not queried, one cannot determine whether the precise value of I_j is equal to v^* or not. Thus, every feasible query set contains all non-trivial intervals that contain v^*.

Furthermore, it is easy to see that the non-trivial intervals containing v^* constitute a feasible query set: Once these intervals are queried, one can determine for each interval whether its precise value is smaller than v^*, equal to v^*, or larger than v^*. □

Let ℓ be the i-th smallest left endpoint, and let u be the i-th smallest right endpoint. Then it is clear that v^* must lie in the interval $[\ell, u]$, which we call the *target area*. The following lemma was essentially shown by Kahan [24]; we include a proof for the sake of completeness.

Lemma 5.3 [24] *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 $[\ell, u]$, i.e., satisfies $\ell_j \leq \ell$ and $u_j \geq u$.*

Proof First, assume that the target area is trivial, i.e., $\ell = u = v^*$. If there is no non-trivial interval in $\mathcal{I}$ that contains v^*, then the instance is already solved, a contradiction.

Now, assume that the target area is non-trivial. Assume that no interval in $\mathcal{I}$ contains the target area. Then all intervals I_j with $\ell_j \leq \ell$ have $u_j < u$. There are at least i such intervals (because ℓ is the i-th smallest left endpoint), and hence the i-th smallest right endpoint must be strictly smaller than u, a contradiction to the definition of u. □

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 [24] for ElementSelection.) 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 $\ell_j \leq \ell$ and $u_j \geq u$; they *contain* the target area;
(2) b intervals $[\ell_j, u_j]$ with $\ell_j > \ell$ and $u_j < u$; they *are strictly contained* in the target area and contain neither endpoint of the target area;
(3) c intervals $[\ell_j, u_j]$ with $\ell_j \leq \ell$ and $u_j < u$; they intersect the target area on the *left*;
(4) d intervals $[\ell_j, u_j]$ with $\ell_j > \ell$ and $u_j \geq u$; 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 5.4 *At the start of any round, $a \geq 1$ and $b \leq a - 1$.*

Proof Lemma 5.3 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 $u_j < \ell$. 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. As ℓ 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. As u 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 $b \leq a-1$. □

Lemma 5.5 *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^*.*

Proof Assume for a contradiction that the algorithm queries at least one interval of categories (3) or (4), and that it queries all intervals of categories (3) and (4) that contain v^*. Observe that the algorithm also queries all intervals in categories (1) and (2), as otherwise it would not have started to query intervals of categories (3) and (4). Thus, the algorithm has queried all intervals that contain v^* and hence solved the problem, a contradiction to the current round not being the last one. □

Theorem 5.6 *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 5.2 and 5.3, $|A|=|A^*| \geq 1$. Since the algorithm prioritizes category (1), by Proposition 5.4 we have $|B| \leq |A|-1$, and thus $|A \cup B| \leq 2 \cdot |A|-1 = 2 \cdot |A^*|-1 \leq 2(|A^*|+|B^*|)-1$.

To bound 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 5.5, 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 5.7 *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.*

Proof Consider the instance with $i-1$ copies of interval $[0, 3]$ (called *left-side* intervals), $i-1$ copies of interval $[5, 8]$ (called *right-side* intervals), and one interval $[2, 6]$ (called *middle* interval). The precise values are always 1 for the left-side intervals, and 7 for right-side intervals. The value of the middle interval depends on the behavior of the algorithm, but in all cases it will be the i-th smallest element. If the algorithm does not query the middle interval in the first round, then we set its value to 4, so we have $\mathrm{opt}_1 = 1$ and the algorithm performs at least $\mathrm{opt}_1 + i = (i+1) \cdot \mathrm{opt}_1$ queries. So assume that the algorithm queries the middle interval in the first round. If it queries more left-side than right-side intervals, then we set the value of the middle interval to 5.5, so all right-side intervals must be queried (and all queries of left-side intervals are wasted); otherwise, we set the middle value to 2.5. In either case, we have $\mathrm{opt}_1 = i$ and the algorithm wastes at least $\lceil (i-1)/2 \rceil$ queries. □

5.3 Element Selection

If we do not need to output the value v^*, we can use the algorithms in the previous sections, and we waste at most one extra query. Thus, if we want to find one element whose value is v^*, the algorithm in Sect. 5.1 performs at most $\mathrm{opt}_1 + i$ queries, which is the best possible [21], and therefore we use at most $\mathrm{opt}_k + \lceil i/k \rceil$ rounds and this is the best possible. If we want to find all elements whose value is v^*, then the algorithm from Theorem 5.6 uses at most $2 \cdot \mathrm{opt}_k + 1$ rounds, which can be proved as follows: Let OPT'_1 be the optimal query set for Selection (finding v^* and all elements with value v^*) and let OPT_1 be the optimal query set for ElementSelection (identifying all elements whose value is v^*). If $|\mathrm{OPT}_1| = |\mathrm{OPT}'_1|$, then it is immediate that the algorithm of Theorem 5.6 uses at most $2 \cdot \mathrm{opt}_k$ rounds. The case $|\mathrm{OPT}_1| \neq |\mathrm{OPT}'_1|$ can only arise if there is a single element e with value v^* and if the optimal solution for identifying e does not query e. In this case, we can assume $\mathrm{OPT}_1 = \mathrm{OPT}'_1 \setminus \{e\}$. The proof of Theorem 5.6 shows the following: If the total number of queries made in all rounds except the final one is z, then at least $z/2$ of those queries are in OPT'_1. If e is not among these z queries, then at least $z/2$ of them are also in OPT_1. This shows $z \leq 2\mathrm{opt}_1$, and thus the number of rounds used by the algorithm is at most $\lceil z/k \rceil + 1 \leq \lceil 2\mathrm{opt}_1/k \rceil + 1 \leq 2\mathrm{opt}_k + 1$. If e is among the z queries, then at least $z/2 - 1$ of them are in OPT_1. Furthermore, the final round must also contain at least one query from OPT_1 (otherwise, $\mathrm{OPT}_1 \cup \{e\} = \mathrm{OPT}'_1$ would have been queried and thus the problem solved before the final round, a contradiction). Therefore, $\mathrm{opt}_1 \geq z/2$. Again, we have $z \leq 2\mathrm{opt}_1$ and get that the number of rounds is at most $\lceil 2\mathrm{opt}_1/k \rceil + 1 \leq 2\mathrm{opt}_k + 1$.

6 Relationship with the Parallel Model by Meißner

In [29, 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 used, 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 uses at most r batches, then there is an algorithm that uses 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 using 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 $\alpha = 1$).

Theorem 6.2 *Assume that there is an α-round-competitive algorithm for rounds of k queries, with α independent of k. Then there is, for every positive integer x, an algorithm that uses at most r batches and has query-competitive ratio $\mathrm{O}(\alpha \cdot n^{\lfloor \alpha \rfloor/(r-1)})$, where $r = \lfloor \alpha \rfloor \cdot x + 1$. In particular, for $x \geq \lg n$, the query-competitive factor is $\mathrm{O}(\alpha)$.*

Proof Given an α-round-competitive algorithm A for rounds of k queries, we construct an algorithm B that uses at most r batches. We group them into sequences of $\lfloor \alpha \rfloor$ batches. For the i-th sequence, for $i = 1, \ldots, x$, algorithm B runs algorithm A for $\lfloor \alpha \rfloor$ rounds with $k = n^{(i-1)/x}$, until the problem is solved. If the problem is not solved after $\lfloor \alpha \rfloor \cdot x$ batches, then algorithm B queries all the remaining intervals in one final batch.

To determine the query-competitive ratio, consider the number i of sequences of $\lfloor \alpha \rfloor$ batches the algorithm executes. If the problem is solved during the i-th sequence, then algorithm B performs at most $\lfloor \alpha \rfloor \cdot (\sum_{j=0}^{i-1} n^{j/x}) = \lfloor \alpha \rfloor \cdot \Theta(n^{(i-1)/x})$ queries. (If the problem is solved during the final batch, it performs at most $n \leq \lfloor \alpha \rfloor \cdot n^{x/x}$ queries.) On the other hand, we claim that, if the problem is not solved after the $(i-1)$-th sequence, then the optimum solution queries at least $n^{(i-2)/x}$ intervals. This is because algorithm A is α-round-competitive, so whenever the algorithm executes a sequence of $\lfloor \alpha \rfloor$ rounds for a certain value of k and does not solve the problem, it follows that the optimum solution requires more than one round for this value of k, and hence more than k queries. Thus, the query-competitive ratio is at most $\lfloor \alpha \rfloor \cdot \Theta(n^{1/x}) = \Theta(\alpha \cdot n^{\lfloor \alpha \rfloor/(r-1)})$. □

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 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.

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

Funding This research was supported by the Engineering and Physical Sciences Research Council (EPSRC), grants EP/S033483/1 and EP/S033483/2.

Declarations

Competing interests The authors have no conflicts of interest to declare that are relevant to the content of this article.

A Lower Bound for MINIMUMELEMENT with Closed Intervals

For completeness, we include her the argument from [21] showing that no deterministic algorithm can be better than n-query-competitive for the MINIMUMELEMENT problem with a single set of n elements if we allow closed intervals in the input. Consider an instance consisting of n identical closed intervals $I_1 = I_2 = \cdots = I_n = [1, 3]$. For a deterministic algorithm that queries the intervals one by one in some order, the adversary returns a value of 2 for the first $n - 1$ queries and a value of 1 for the last query. Let the interval of the last query be I_ℓ. The algorithm can identify I_ℓ with value $v_\ell = 1$ as a minimum element only after n queries, while the optimal solution only needs to execute a single query on I_ℓ to prove that it is a minimum element.

This lower bound also shows that no deterministic algorithm can be better than $\frac{n}{k}$-round-competitive for the MinimumElement problem with a single set of n closed intervals if k queries per round are allowed.

B Selection with Arbitrary Intervals

In this section we prove that Theorem 5.6 also holds for arbitrary intervals.

An instance of the Selection problem is given by a set $\mathcal{I}$ of n intervals and an integer i, $1 \leq i \leq n$. The i-th smallest value in the set of n precise values is denoted by v^*. The task is to find v^* as well as identify all intervals in $\mathcal{I}$ whose precise value equals v^*.

We allow arbitrary intervals as input: trivial intervals containing a single value, open intervals, closed intervals, and intervals that are closed on one side and open on the other.

Lemma 5.2 and its proof hold also for arbitrary intervals without any changes.

We call the left endpoint ℓ_i of an interval I_i an *open left endpoint* if the interval does not contain ℓ_i and a *closed left endpoint* otherwise. The definitions of the terms *open right endpoint* and *closed right endpoint* are analogous. When we order the left endpoints of the intervals in non-decreasing order, equal left endpoints are ordered as follows: closed left endpoints come before open left endpoints. When we order the right endpoints of the intervals in non-decreasing order, equal right endpoints are ordered as follows: open right endpoints come before closed right endpoints. Equal left endpoints that are all open can be ordered arbitrarily, and the same holds for equal left endpoints that are all closed, for equal right endpoints that are all open, and for equal right endpoints that are all closed. Informally, the order of left endpoints orders intervals in order of the "smallest" values they contain, and the order of right endpoints orders intervals in order of the "largest" values they contain. We call the resulting order of left endpoints $\preceq_L$ and the resulting order of right endpoints $\preceq_U$. We say that a right endpoint u_{i_1} *strictly precedes* a right endpoint u_{i_2} if either $u_{i_1} < u_{i_2}$ or $u_{i_1} = u_{i_2}$ and u_{i_1} is an open right endpoint and u_{i_2} is a closed right endpoint.

Let I_{j_1} be the interval with the i-th smallest left endpoint (i.e., the i-th left endpoint in the order $\preceq_L$), and let I_{j_2} be the interval with the i-th smallest right endpoint (i.e, the i-th right endpoint in the order $\preceq_U$). Then it is clear that v^* must lie in the interval I_{ta}, which we call the *target area* and define as follows:

- If ℓ_{j_1} is an open left endpoint of I_{j_1} and u_{j_2} is an open right endpoint of I_{j_2}, then $I_{\text{ta}} = (\ell_{j_1}, u_{j_2})$.
- If ℓ_{j_1} is an open left endpoint of I_{j_1} and u_{j_2} is a closed right endpoint of I_{j_2}, then $I_{\text{ta}} = (\ell_{j_1}, u_{j_2}]$.

- If ℓ_{j_1} is a closed left endpoint of I_{j_1} and u_{j_2} is an open right endpoint of I_{j_2}, then $I_{\text{ta}} = [\ell_{j_1}, u_{j_2})$.
- If ℓ_{j_1} is a closed left endpoint of I_{j_1} and u_{j_2} is a closed right endpoint of I_{j_2}, then $I_{\text{ta}} = [\ell_{j_1}, u_{j_2}]$.

The following lemma was essentially shown by Kahan [24]; for the sake of completeness, we give a proof for arbitrary intervals.

Lemma B.1 (Kahan [24]; version of Lemma 5.3 for arbitrary intervals) *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_{ta}.*

Proof First, assume that the target area is trivial, i.e., $I_{\text{ta}} = \{v^*\}$. If there is no non-trivial interval in $\mathcal{I}$ that contains v^*, then the instance is already solved, a contradiction.

Now, assume that the target area I_{ta} is non-trivial. Assume that no interval in $\mathcal{I}$ contains the target area. Then all intervals I_j whose left endpoint is not after ℓ_{j_1} in the order of left endpoints must have a right endpoint that strictly precedes u_{j_2}. There are at least i such intervals (because ℓ_{j_1} is the i-th smallest left endpoint), and hence the i-th smallest right endpoint must strictly precede u_{j_2} in the order of right endpoints, a contradiction to the definition of u_{j_2}. □

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

(1) a non-trivial intervals that *contain* the target area;
(2) b intervals that *are strictly contained* in the target area such that the target area contains at least one point to the left of the interval and at least one point to the right of the interval.
(3) c intervals that contain some part of I_{ta} at the left end and do not contain some part of I_{ta} on the right end. Formally, an interval I_i with closed right endpoint u_i is in this category if $u_i \in I_{\text{ta}}$, $I_i \cap I_{\text{ta}} = \{v \in I_{\text{ta}} \mid v \leq u_i\}$ and $\{v \in I_{\text{ta}} \mid v > u_i\} \neq \emptyset$. Moreover, an interval I_i with open right endpoint u_i is in this category if $u_i \in I_{\text{ta}}$ and $I_i \cap I_{\text{ta}} = \{v \in I_{\text{ta}} \mid v < u_i\} \neq \emptyset$.
(4) d intervals that contain some part of I_{ta} at the right end and do not contain some part of I_{ta} on the left end. Formally, an interval I_i with closed left endpoint ℓ_i is in this category if $\ell_i \in I_{\text{ta}}$, $I_i \cap I_{\text{ta}} = \{v \in I_{\text{ta}} \mid v \geq \ell_i\}$ and $\{v \in I_{\text{ta}} \mid v < \ell_i\} \neq \emptyset$. Moreover, an interval I_i with open left endpoint ℓ_i is in this category if $\ell_i \in I_{\text{ta}}$ and $I_i \cap I_{\text{ta}} = \{v \in I_{\text{ta}} \mid v > \ell_i\} \neq \emptyset$.

We propose the following algorithm for rounds with k queries. Each round is filled with as many 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 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. More precisely, intervals of category (3) are chosen according to the reverse of the order $\preceq_U$ of their right endpoints, and intervals of category (4) are chosen according to the order $\preceq_L$ of their left endpoints. When a round is filled, it is queried, and the algorithm restarts, calculating a new target area and redistributing the intervals into the categories.

Proposition B.2 (Version of Proposition 5.4 for arbitrary intervals) *At the start of any round, $a \geq 1$ and $b \leq a - 1$.*

Proof Lemma B.1 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 I_{ta} (and have empty intersection with I_{ta}). Similarly, let R be the set of intervals that lie to the right of I_{ta} (and have empty intersection with I_{ta}). 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 not after ℓ_{j_1} in the order $\preceq_L$. As ℓ_{j_1} is the i-th left endpoint in that order, 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 not before u_{j_2} in the order $\preceq_U$. As u_{j_2} is the i-th smallest right endpoint in that order, or equivalently the $(n - i + 1)$-th largest right endpoint in that order, 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 $b \leq a - 1$. □

Lemma 5.5 and its proof hold for arbitrary intervals without any changes.

Theorem B.3 *There is a 2-round-competitive algorithm for* Selection *even if arbitrary intervals are allowed as input.*

Proof The proof is identical to the proof of Theorem 5.6, except that Lemma B.1 and Proposition B.2 are used in place of Lemma 5.3 and Proposition 5.4, respectively. □

References

1. Ajtai, M., Feldman, V., Hassidim, A., Nelson, J.: Sorting and selection with imprecise comparisons. ACM Trans. Algorithms **12**(2), 19:1-19:19 (2016). https://doi.org/10.1145/2701427
2. Albers, S., Eckl, A.: Explorable uncertainty in scheduling with non-uniform testing times. In: Kaklamanis, C., Levin, A. (eds.) WAOA 2020: 18th International Workshop on Approximation and Online Algorithms, Lecture Notes in Computer Science, vol. 12806, pp. 127–142. Springer (2020). https://doi.org/10.1007/978-3-030-80879-2_9
3. Arantes, L., Bampis, E., Kononov, A.V., Letsios, M., Lucarelli, G., Sens, P.: Scheduling under uncertainty: a query-based approach. In: IJCAI 2018: 27th International Joint Conference on Artificial Intelligence, pp. 4646–4652 (2018). https://doi.org/10.24963/ijcai.2018/646
4. Beerliova, Z., Eberhard, F., Erlebach, T., Hall, A., Hoffmann, M., Mihalák, M., Ram, L.S.: Network discovery and verification. IEEE J. Sel. Areas Commun. **24**(12), 2168–2181 (2006). https://doi.org/10.1109/JSAC.2006.884015
5. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
6. Braverman, M., Mossel, E.: Sorting from noisy information (2009). arXiv:0910.1191
7. Bruce, R., Hoffmann, M., Krizanc, D., Raman, R.: Efficient update strategies for geometric computing with uncertainty. Theory Comput. Syst. **38**(4), 411–423 (2005). https://doi.org/10.1007/s00224-004-1180-4
8. Canonne, C.L., Gur, T.: An adaptivity hierarchy theorem for property testing. Comput. Complex. **27**, 671–716 (2018). https://doi.org/10.1007/s00037-018-0168-4
9. Charalambous, G., Hoffmann, M.: Verification problem of maximal points under uncertainty. In: Lecroq, T., Mouchard, L. (eds.) IWOCA 2013: 24th International Workshop on Combinatorial Algo-

rithms, Lecture Notes in Computer Science, vol. 8288, pp. 94–105. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-45278-9_9

10. Dürr, C., Erlebach, T., Megow, N., Meißner, J.: An adversarial model for scheduling with testing. Algorithmica (2020). https://doi.org/10.1007/s00453-020-00742-2
11. Erlebach, T., Hoffmann, M.: Minimum spanning tree verification under uncertainty. In: Kratsch, D., Todinca, I. (eds.) WG 2014: International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, vol. 8747, pp. 164–175. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-12340-0_14
12. Erlebach, T., Hoffmann, M.: Query-competitive algorithms for computing with uncertainty. Bull. EATCS **116**, 22–39 (2015)
13. Erlebach, T., Hoffmann, M., Kammer, F.: Query-competitive algorithms for cheapest set problems under uncertainty. Theoret. Comput. Sci. **613**, 51–64 (2016). https://doi.org/10.1016/j.tcs.2015.11.025
14. Erlebach, T., Hoffmann, M., Krizanc, D., Mihal'ák, M., Raman, R.: Computing minimum spanning trees with uncertainty. In: Albers, S., Weil, P. (eds.) STACS'08: 25th International Symposium on Theoretical Aspects of Computer Science, Leibniz International Proceedings in Informatics, vol. 1, pp. 277–288. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2008). https://doi.org/10.4230/LIPIcs.STACS.2008.1358
15. Erlebach, T., Hoffmann, M., de Lima, M.S., Megow, N., Schlöter, J.: Untrusted predictions improve trustable query policies (2020). arXiv:2011.07385
16. Feder, T., Motwani, R., O'Callaghan, L., Olston, C., Panigrahy, R.: Computing shortest paths with uncertainty. J. Algorithms **62**(1), 1–18 (2007). https://doi.org/10.1016/j.jalgor.2004.07.005
17. Feder, T., Motwani, R., Panigrahy, R., Olston, C., Widom, J.: Computing the median with uncertainty. SIAM J. Comput. **32**(2), 538–547 (2003). https://doi.org/10.1137/S0097539701395668
18. Focke, J., Megow, N., Meißner, J.: Minimum spanning tree under explorable uncertainty in theory and experiments. In: Iliopoulos, C.S., Pissis, S.P., Puglisi, S.J., Raman, R. (eds.) SEA 2017: 16th International Symposium on Experimental Algorithms, Leibniz International Proceedings in Informatics, vol. 75, pp. 22:1–22:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2017). https://doi.org/10.4230/LIPIcs.SEA.2017.22
19. Gavril, F.: Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. SIAM J. Comput. **1**(2), 180–187 (1972). https://doi.org/10.1137/0201013
20. Goerigk, M., Gupta, M., Ide, J., Schöbel, A., Sen, S.: The robust knapsack problem with queries. Comput. Oper. Res. **55**, 12–22 (2015). https://doi.org/10.1016/j.cor.2014.09.010
21. Gupta, M., Sabharwal, Y., Sen, S.: The update complexity of selection and related problems. Theory Comput. Syst. **59**(1), 112–132 (2016). https://doi.org/10.1007/s00224-015-9664-y
22. Halldórsson, M.M., de Lima, M.S.: Query-competitive sorting with uncertainty. In: Rossmanith, P., Heggernes, P., Katoen, J.P. (eds.) MFCS 2019: 44th International Symposium on Mathematical Foundations of Computer Science, Leibniz International Proceedings in Informatics, vol. 138, pp. 7:1–7:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.MFCS.2019.7
23. Hoorfar, A., Hassani, M.: Inequalities on the Lambert W function and hyperpower function. J. Inequal. Pure Appl. Math. **9**(2), 51:1-51:5 (2008)
24. Kahan, S.: A model for data in motion. In: STOC'91: 23rd Annual ACM Symposium on Theory of Computing, pp. 265–277 (1991). https://doi.org/10.1145/103418.103449
25. Khanna, S., Tan, W.C.: On computing functions with uncertainty. In: PODS'01: 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 171–182 (2001). https://doi.org/10.1145/375551.375577
26. Lekkerkerker, C., Boland, J.: Representation of a finite graph by a set of intervals on the real line. Fundamenta Mathematicae **51**(1), 45–64 (1962)
27. Maehara, T., Yamaguchi, Y.: Stochastic packing integer programs with few queries. Math. Program. **182**, 141–174 (2020). https://doi.org/10.1007/s10107-019-01388-x
28. Megow, N., Meißner, J., Skutella, M.: Randomization helps computing a minimum spanning tree under uncertainty. SIAM J. Comput. **46**(4), 1217–1240 (2017). https://doi.org/10.1137/16M1088375
29. Meißner, J.: Uncertainty exploration: algorithms, competitive analysis, and computational experiments. Ph.D. thesis, Technische Universität Berlin (2018). https://doi.org/10.14279/depositonce-7327

30. Merino, A.I., Soto, J.A.: The minimum cost query problem on matroids with uncertainty areas. In: Baier, C., Chatzigiannakis, I., Flocchini, P., Leonardi, S. (eds.) ICALP 2019: 46th International Colloquium on Automata, Languages, and Programming, Leibniz International Proceedings in Informatics, vol. 132, pp. 83:1–83:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.83
31. Olston, C., Widom, J.: Offering a precision-performance tradeoff for aggregation queries over replicated data. In: VLDB 2000: 26th International Conference on Very Large Data Bases, pp. 144–155 (2000). http://ilpubs.stanford.edu:8090/714/
32. Ryzhov, I.O., Powell, W.B.: Information collection for linear programs with uncertain objective coefficients. SIAM J. Optim. **22**(4), 1344–1368 (2012). https://doi.org/10.1137/12086279X
33. van der Hoog, I., Kostitsyna, I., Löffler, M., Speckmann, B.: Preprocessing ambiguous imprecise points. In: Barequet, G., Wang, Y. (eds.) SoCG 2019: 35th International Symposium on Computational Geometry, Leibniz International Proceedings in Informatics, vol. 129, pp. 42:1–42:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.SoCG.2019.42
34. Welz, W.A.: Robot tour planning with high determination costs. Ph.D. thesis, Technische Universität Berlin (2014). https://www.depositonce.tu-berlin.de/handle/11303/4597