Query-Competitive Sorting with Uncertainty

Magnús M. Halldórsson
ICE-TCS, Department of Computer Science, Reykjavik University, Iceland
mmh@ru.is

Murilo Santos de Lima[1]
ICE-TCS, Department of Computer Science, Reykjavik University, Iceland
mslima@ic.unicamp.br

Abstract

We study the problem of sorting under incomplete information, when queries are used to resolve uncertainties. Each of n data items has an unknown value, which is known to lie in a given interval. We can pay a query cost to learn the actual value, and we may allow an error threshold in the sorting. The goal is to find a nearly-sorted permutation by performing a minimum-cost set of queries.

We show that an offline optimum query set can be found in polynomial time, and that both oblivious and adaptive problems have simple query-competitive algorithms. The query-competitiveness for the oblivious problem is n for uniform query costs, and unbounded for arbitrary costs; for the adaptive problem, the ratio is 2.

We then present a unified adaptive strategy for uniform query costs that yields: (i) a 3/2-query-competitive randomized algorithm; (ii) a 5/3-query-competitive deterministic algorithm if the dependency graph has no 2-components after some preprocessing, which has query-competitive ratio $3/2 + \mathrm{O}(1/k)$ if the components obtained have size at least k; (iii) an exact algorithm if the intervals constitute a laminar family. The first two results have matching lower bounds, and we have a lower bound of 7/5 for large components.

We also show that the advice complexity of the adaptive problem is $\lfloor n/2 \rfloor$ if no error threshold is allowed, and $\lceil n/3 \cdot \lg 3 \rceil$ for the general case.

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

Keywords and phrases online algorithms, sorting, randomized algorithms, advice complexity, threshold tolerance graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.7

Funding Partially supported by Icelandic Research Fund grant 174484-051.

1 Introduction

Sorting is one of the most fundamental problems in computer science and an essential part of any system dealing with large amounts of data. High-performance algorithms such as QuickSort [19] have been known for decades, but the demand for fast sorting of huge amounts of data is such that improvements in sorting algorithms are still an active area of research; see, e.g., [26].

In a distributed application with dynamic data, it may not be feasible to maintain a precise copy of the information in each replica. In particular, to access a local cached information may be much cheaper, even though not as precise, than to query a master database or to run a distributed consensus algorithm. One approach is to maintain in the replicas, for each data item, an interval that bounds the actual value. These intervals can be updated much faster than to guarantee a strict consistency of the data. When higher

[1] Corresponding author.

precision is required, the system can query the master database for a more fine-grained interval or for the actual data value. Therefore a trade-off between data precision and system performance can be established. The TRAPP system, proposed by Olston and Widom [24], relies on this concept.

This idea has led to theoretical investigation on **uncertainty problems with queries** [7, 11, 12, 13, 14, 17, 22]. Such problems also appear in optimization scenarios in which an extra effort can be incurred in order to obtain more precise values of the input data, such as by investing in market research, which is expensive so its cost should be minimized. These works build upon more established frameworks of optimization with uncertainty, such as online [5], robust [3] and stochastic [4] optimization. In particular, the analysis of algorithms in terms of competitiveness against an adversary is inherited from the online optimization literature.

In this paper, we investigate the problem of sorting data items whose actual values are unknown, but for which we are given intervals on which the actual values lie. We can **query** an interval and then learn the actual value of the corresponding data item, but this incurs some cost. The goal, then, is to sort the items by performing a set of queries of minimum cost. Furthermore, the precision in the sorting may be relaxed, so that inversions may occur if the actual values are not too far apart.

We distinguish between two types of algorithms for uncertainty problems with queries. An **adaptive** algorithm may decide which queries to perform based on results from previous queries. An **oblivious** algorithm, however, must choose the whole set of queries to perform in advance; i.e., it must choose a set of queries that certainly allow the problem to be solved without any knowledge of the actual values. In this paper, both algorithms are compared with an offline optimum query set, i.e., a minimum-cost set of queries that proves the obtained solution to be correct.[2] An algorithm (either adaptive or oblivious) is **α-query-competitive** if it performs a total query cost of at most α times the cost of an offline optimum query set.

Another related problem is that of finding an **optimum query set**. Here we are given the actual data values, and want to identify a minimum-cost set of queries that would be sufficient to prove that the solution is correct. Solving this problem is useful, for example, to perform experimental evaluation of online algorithms, since we are actually finding the offline optimum solution for the uncertainty problem. This is also called the **verification version** of the corresponding uncertainty problem with queries [8, 11].

We are also interested in the **advice complexity** of the problems we study. In this setting, an online algorithm has access to an oracle that can give helpful information when making decisions. The advice complexity is the number of bits of advice that are sufficient and necessary for an online algorithm to solve the problem exactly. This is a research topic that has gained substantial attention; see [6] for a survey.

Our contribution. We begin by showing how to compute an optimum query set in polynomial time, and that both oblivious and adaptive problems have simple algorithms with matching deterministic lower bounds. The query-competitive ratio of the oblivious problem is n if we have uniform query costs, and unbounded for arbitrary costs; for the adaptive problem, the query-competitive ratio is 2. The optimal oblivious algorithm is trivial; for the

[2] This nomenclature differs to that used by Feder et al. [14]. They call an adaptive algorithm an **online** algorithm, and an oblivious algorithm an **offline** algorithm. We disagree with this nomenclature, since both types of algorithms are online in the standard sense of not knowing the data. Also, they compare an oblivious algorithm to an optimal oblivious strategy, and not to an offline optimum query set.

adaptive case, we have a simpler algorithm for uniform query costs, and a more sophisticated one for arbitrary query costs. If query costs are uniform and the error threshold is zero, then the simpler algorithm can be implemented as an oracle for any comparison-based sorting algorithm, preserving time complexity and stability.

Those results are rather simple and it seems like the query-competitiveness of the problem is settled. However, we present a unified adaptive strategy that attains different improvements for uniform query costs. First, we obtain a 3/2-query-competitive algorithm by using randomization. Second, if the error threshold is zero, and after some preprocessing the dependency graph has no 2-components, the strategy yields a deterministic 5/3-query-competitive algorithm; if the obtained graph has components of size at least k, then the same algorithm has query-competitive ratio $3/2 + \mathrm{O}(1/k)$. The first two results have a matching lower bound, and for large components we have a lower bound of 7/5. The problem can also be solved exactly if the intervals constitute a laminar family.

Finally, we show that the advice complexity for adaptive algorithms is exactly $\lfloor n/2 \rfloor$ bits if there is no error threshold, and exactly $\lceil n/3 \cdot \lg 3 \rceil$ bits for the general case.

Related work. The first work to investigate the minimum number of queries to solve a problem is by Kahan [20], who showed optimal oblivious strategies to find the minimum, maximum and median of n values in uncertainty intervals.

Olston and Widom [24] proposed the TRAPP system, a distributed database based on uncertainty intervals. The authors: (1) gave an optimal oblivious strategy for finding the minimum (and equivalently, the maximum) of a sequence of values within an error bound; (2) showed that it is NP-hard to find an optimum oblivious query set to compute the sum of a sequence of values within an error bound, with a reduction from the knapsack problem. The paper also discusses strategies for counting and finding the average of a sequence of values. Khanna and Tan [21] generalized these results for arbitrary query costs and different levels of precision.

Feder et al. [14] considered the uncertainty version of the problem of finding the k-th largest value on a sequence (i.e., the generalized median problem). The authors presented optimal oblivious and adaptive strategies for the problem, both running in polynomial time. Both strategies are optimal, and the ratio between the oblivious and the adaptive strategy (also called the **price of obliviousness**) is $\frac{2k-1}{k} < 2$ for uniform query costs, and k for arbitrary query costs.[3]

Bruce et al. [7] studied geometric problems where the points are given in uncertainty areas. The authors gave 3-query-competitive algorithms for finding the maximal points and the convex hull in a two-dimensional space. They also proposed the concept of **witness sets**, which has been used subsequently in various works on uncertainty problems with queries. Charalambous and Hoffman [8] showed that it is NP-hard to find an optimum query set for the maximal points problem.

Feder et al. [13] studied the uncertainty variant of the shortest path problem. They showed that to solve optimally the oblivious version of the problem is neither NP nor co-NP, unless NP = co-NP. Their paper also discusses the complexity of the problem for various particular cases.

Erlebach et al. [12] proved that the minimum spanning tree problem with uncertainty admits an adaptive 2-query-competitive algorithm, which is the best possible for a deterministic algorithm. Erlebach, Hoffmann and Kammer [11] studied a generalization called the

[3] The works cited up to this point do not evaluate the algorithms using the competitiveness framework.

cheapest set problem, for which there is an adaptive algorithm with at most $d \cdot \mathrm{opt} + d$ queries, where d is the maximum cardinality of a set. They also generalized the result in the previous work to obtain an adaptive 2-query-competitive algorithm for the problem of finding a minimum-weight base on a matroid.

Gupta, Sabharwal and Sen [18] studied various of the previous problems in the setting where a query may return a refined interval, instead of the exact value of the data item.

Megow, Meißner and Skutella [22] improved the result for the minimum spanning tree problem with a randomized adaptive algorithm, obtaining query-competitive ratio 1.7. (The problem has lower bound 1.5 for randomized algorithms.) They also considered non-uniform query costs and proved that their results can be extended to find a minimum-weight base on a matroid. Furthermore, they showed that an optimum query set and the actual value of the minimum spanning tree can be computed in polynomial time. Some experimental evaluation of those algorithms were presented in [15].

Ryzhov and Powell [25] investigated how to solve a linear program while minimizing the query cost when the coefficients of the objective function are uncertain. They presented a policy which is asymptotically optimal. Yamaguchi and Maehara [28] studied the variant with packing constraints and coefficients following a probability distribution, and showed how to apply this to stochastic problems such as matching, matroid and stable set problems.

Note that all the work cited so far deals with problems whose classical (offline) versions can be solved in polynomial time. Uncertainty versions with queries have been proposed for the knapsack problem [17] and the scheduling problem [2, 9]. Since those problems are NP-hard, we might include the query cost into the solution cost and look for a competitive algorithm if we are looking for a polynomial-time algorithm. Another option is to limit the maximum number of queries performed, and then to try to optimize the solution cost.

For a survey on the topic, see [10]. Other references for related problems are also cited in [11].

Another sorting problem with uncertainty was studied by Ajtain et al. [1]. In that problem, the values to be sorted are unknown, but their relative order can be tested by a comparison procedure. However, comparing values that are too close returns imprecise answers, so in principle we should compare all $\binom{n}{2}$ pairs to obtain a sorting with some error guarantee. The authors show how to solve the problem using only $\mathrm{O}(n^{3/2})$ comparisons.

Organization of the paper. In Section 2, we present the sorting problem with uncertainty and some basic facts, and in Section 3 we give algorithms to find an offline optimum query set and for the oblivious setting. We treat deterministic adaptive algorithms in Section 4. In Section 5, we show how to improve the adaptive result for uniform query costs by using a randomized algorithm, or by assuming some structure in the dependency graph. We investigate the advice complexity for adaptive algorithms in Section 6, and finally, in Section 7, we discuss possible future research directions.

2 Sorting with Uncertainty

In the sorting problem with uncertainty, there are n numbers $v_1, \ldots, v_n \in \mathbb{R}$ whose exact value is unknown. We are given n uncertainty intervals $I_1, \ldots, I_n$ with $v_i \in I_i = [\ell_i, r_i]$, a cost $w_i \in \mathbb{R}_+$ for querying interval I_i, and an error threshold $\delta \geq 0$. After querying I_i, we obtain the exact value of v_i; we can also say that we replace I_i with interval $I'_i = [v_i, v_i]$. The goal is to obtain a permutation $\pi : [n] \to [n]$ such that $v_i \leq v_j + \delta$ if $\pi(i) < \pi(j)$ by performing a minimum-cost set of queries.

We begin by defining the following dependency relation between intervals, which is essential to solve the problem.

▶ **Definition 2.1.** *Two intervals I_i and I_j such that $r_i - \ell_j > \delta$ and $r_j - \ell_i > \delta$ are **dependent**. Two intervals that are not dependent are **independent**.*

▶ **Lemma 2.2.** *The relative order between two intervals can be decided without querying either of them if and only if they are independent.*

Proof. Let I_i and I_j be such that $r_i - \ell_j \leq \delta$. Since $v_i \leq r_i$ and $v_j \geq \ell_j$, we have that $v_i \leq v_j + \delta$ and we can set $\pi(i) < \pi(j)$ without querying both I_i and I_j.

Conversely, let I_j and I_j be two dependent intervals. We cannot set $\pi(i) < \pi(j)$, because it may be the case that $v_i = r_i$ and $v_j = \ell_j$, thus $r_i - \ell_j > \delta$ implies that $v_i > v_j + \delta$. By a symmetric argument, we cannot set $\pi(j) < \pi(i)$, so we cannot decide the relative order between the intervals. ◀

The graphs defined by this dependency relation are exactly the **co-threshold tolerance** (co-TT) graphs [23]. $G = (V, E)$ is a **threshold tolerance** graph if there are functions $w : V \to \mathbb{R}$ and $t : V \to \mathbb{R}$ such that $uv \in E$ if and only if $w(u) + w(v) \geq \min(t(u), t(v))$. A co-TT graph is the complement of any threshold tolerance graph, or equivalently, $G = (V, E)$ is a co-TT graph if and only if there are functions $a : V \to \mathbb{R}$ and $b : V \to \mathbb{R}$ such that $uv \in E$ if and only if $a(u) < b(v)$ and $a(v) < b(u)$ [23]. The proof of Theorem 2.3 is omitted.

▶ **Theorem 2.3.** *The graphs defined by the dependency relation in Definition 2.1 are exactly the co-TT graphs.*

The following result will be useful.

▶ **Lemma 2.4** ([23])**.** *Every co-TT graph is chordal.*

When $\delta > 0$, it is useful to distinguish intervals of width smaller than δ, which we call **trivial** intervals. It is easy to check that two trivial intervals cannot be dependent, so when a trivial and a non-trivial interval are dependent, it is enough to query the non-trivial interval in order to decide their relative order. This does not mean, however, that trivial intervals should never be queried, and in particular adaptive algorithms may decide to do that.

It is also clear that the dependency graph is an interval graph when $\delta = 0$. This is also true when $\delta > 0$ and there are no trivial intervals, in which case we can simply replace each interval $I_i = [\ell_i, r_i]$ with $I_i^{(\delta)} = [\ell_i^{(\delta)}, r_i^{(\delta)}] := [\ell_i + \delta/2, r_i - \delta/2] \neq \emptyset$, and it is easy to check that I_i and I_j are dependent with error threshold δ if and only if $I_i^{(\delta)}$ and $I_j^{(\delta)}$ are dependent with error threshold 0. Note however that we cannot use this reduction to solve the sorting problem, since the precise values could fall outside of the given interval.

3 Warm-Up: Offline and Oblivious Algorithms

The first result we present concerns finding the optimum query set for a given set of intervals, assuming we know the actual values in each interval. I.e., given the intervals $I_1, \ldots, I_n$ and the actual values $v_1, \ldots, v_n$, find a minimum-cost set Q of intervals to query, such that Q is sufficient to prove an ordering for $I_1, \ldots, I_n$ without the knowledge of $v_1, \ldots, v_n$. Solving this problem is useful, for example, to perform experimental evaluation of algorithms, since we are actually finding the offline optimum solution for the online (either oblivious or adaptive) problem. The ideas we present here will also be useful when solving the online problem.

We show that the problem can be solved optimally in polynomial time. The key observations behind the algorithm are the following. In order to simplify notation, we write $I_i \supset I_j$ for intervals I_i and I_j if $\ell_i < \ell_j$ and $r_j < r_i$.

▶ **Proposition 3.1.** *Let I_i and I_j be intervals with actual values v_i and v_j. If $I_j \supset [v_i-\delta, v_i+\delta]$, then I_j is queried by every optimum solution.*

Proof. Even if we have queried I_i, we have to query I_j because we may have $v_j \in [\ell_j, v_i - \delta)$ or $v_j \in (v_i + \delta, r_j]$. ◀

▶ **Proposition 3.2.** *Let I_i and I_j be two dependent intervals, v_i the actual value in I_i and v_j the actual value in I_j. If $I_i \not\supset [v_j - \delta, v_j + \delta]$ and $I_j \not\supset [v_i - \delta, v_i + \delta]$, then it is enough to query either I_i or I_j to decide their relative order.*

Proof. If we query I_i, then $v_i \notin [\ell_j + \delta, r_j - \delta]$, so we can pick a reasonable order between I_i and I_j. The argument is symmetrical if we query I_j. ◀

The algorithm begins with a query set Q containing all intervals that satisfy the condition in Proposition 3.1. Due to Proposition 3.2, it is enough to complement Q with a minimum-cost vertex cover in the dependency graph defined by the remaining intervals, which can be found in polynomial time for chordal graphs [16].

▶ **Theorem 3.3.** *The problem of finding an optimum query set for the sorting problem with uncertainty can be solved optimally in polynomial time.*

Now we consider oblivious algorithms. In this case, all non-trivial intervals with some dependence must be queried, and clearly this is the best possible strategy. In the following theorem, we show that this implies a tight bound of n on the query-competitive ratio for the case with uniform costs, and that in the general case the query-competitive ratio is unbounded.

▶ **Theorem 3.4.** *If query costs are uniform, any oblivious algorithm for sorting with uncertainty has query-competitive ratio exactly n. For arbitrary costs, the query-competitive ratio is unbounded.*

Proof. For the upper bound with uniform costs, a naïve algorithm that queries all intervals and then sorts the numbers suffices.

For both lower bounds, we have $n-1$ independent intervals with length greater than 2δ, plus an interval I_n which contains all the other ones. Both an algorithm and the optimum solution must query I_n in order to decide where v_n fits in the order. If the algorithm does not query some I_i with $i < n$, then the adversary can set $v_n \in (\ell_i+\delta, r_i-\delta) \neq \emptyset$ and the algorithm cannot decide the order. Thus, without the knowledge of v_n, the algorithm must query all I_i with $i < n$. However, it may be the case that $v_n \notin I_i$ for all $i < n$, and querying I_n suffices to decide the order. This gives a lower bound of n on the query-competitive ratio for uniform query costs. For the general case, w_n can be arbitrarily small and the query-competitive ratio is unbounded. ◀

4 Deterministic Adaptive Algorithms

Now let us consider deterministic adaptive algorithms. We begin with a lower bound.

▶ **Lemma 4.1.** *Any deterministic adaptive algorithm for the sorting problem with uncertainty has query-competitive ratio at least 2, even if query costs are uniform and the dependency graph has large components.*

Proof. Consider intervals I_1 and I_2 with uniform query cost, $\ell_1 < \ell_2 < r_1 < r_2$ and $r_1 - \ell_2 > 2\delta$. If the algorithm queries I_1, then the adversary chooses $v_1 \in (\ell_2 + \delta, r_1 - \delta)$. The algorithm must also query I_2 to decide the order, but then the adversary can choose $v_2 \in [r_1 - \delta, r_2]$ and one query would be sufficient. The argument is symmetrical if the algorithm queries I_2 first, with $v_2 \in (\ell_2 + \delta, r_1 - \delta)$ and $v_1 \in [\ell_1, \ell_2 + \delta]$. To obtain a large component, make several independent copies of this structure and connected them by a large interval containing all the others. ◀

First we give a simple deterministic 2-query-competitive adaptive algorithm for the case with uniform query costs. It is inspired by the algorithm of Erlebach et al. [12] for the minimum spanning tree problem with uncertainty, and it relies on the following concepts, which were introduced in [7]. Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a set of intervals for the sorting problem with uncertainty. We say that a set $W \subseteq \mathcal{I}$ of intervals is a **witness set** if at least one of the intervals in W must be queried to decide the order of $\mathcal{I}$, even if all intervals except those in W are queried. Due to Lemma 2.2, any pair of dependent intervals constitute a witness set. A set of intervals $\mathcal{I}' = \{I'_1, \ldots, I'_n\}$ is a **refinement** of $\mathcal{I}$ if $\mathcal{I}'$ is obtained from $\mathcal{I}$ by performing a sequence of queries. Proposition 4.2 follows simply from $\mathcal{I}'$ having more information than $\mathcal{I}$.

▶ **Proposition 4.2.** *Let $\mathcal{I}'$ be a refinement of $\mathcal{I}$. If some set of intervals $W \subseteq \mathcal{I}' \cap \mathcal{I}$ is a witness set for $\mathcal{I}'$, then it is a witness set for $\mathcal{I}$.*

The algorithm, then, consists in the following. While there is some pair of dependent intervals, we query all intervals in this pair that have not been queried yet. When an interval I_i is queried, it is replaced by $[v_i, v_i]$. (Note that, even after querying I_i, it may still be dependent to a non-trivial interval.) Finally, intervals are sorted by breaking ties arbitrarily.[4]

For a better understanding of the algorithm, consider the examples in Figure 1, assuming $\delta = 0$. In Figure 1a, the optimum solution must query I_1 and I_3, since $v_1 \in I_3$ and $v_3 \in I_1$, and this is enough because I_2 will be independent after querying I_1. If the algorithm first queries I_1 and I_2, it must also query I_3. In Figure 1b it is enough to query I_1, but the algorithm will query a dependent pair, say, I_1 and I_2. Either way, the algorithm does not spend more than twice the optimum number of queries.

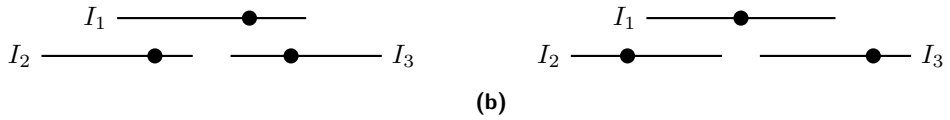

Figure 1 Example instances of the problem.

▶ **Theorem 4.3.** *The simple adaptive algorithm for sorting with uncertainty is 2-query-competitive for uniform query costs.*

Proof. Note that the optimum solution must query at least one interval in each witness set. For every pair $\{I_i, I_j\}$ of dependent intervals selected by the algorithm, we have that:

[4] If $\delta = 0$, then the algorithm can be implemented with stable sorting and in $O(n \lg n)$ time by running a standard stable sorting algorithm (e.g., MergeSort) and querying two intervals when MergeSort needs to know the relative order between them. It does not work, however, if $\delta > 0$, since the relation $v_i \leq v_j + \delta$ is not transitive.

(1) if both I_i and I_j have not been queried yet, the algorithm queries the witness set $\{I_i, I_j\}$; (2) if I_i has already been queried then, by Proposition 3.1, $\{I_j\}$ is a witness set, which is queried by the algorithm. We can conclude that the algorithm only queries disjoint witness sets of size at most 2, and thus it queries at most twice the minimum number of intervals. ◀

For arbitrary query costs, the problem also admits a 2-query-competitive deterministic adaptive algorithm, although not as simple. The algorithm first queries a minimum-cost vertex cover S_1 on the dependency graph. Then, it queries all non-trivial intervals that are still dependent after querying S_1, which we denote by the set S_2.

▶ **Theorem 4.4.** *The adaptive algorithm for sorting with uncertainty with arbitrary query costs is 2-query-competitive.*

Proof. Let Q be an optimum query set. The set of intervals not contained in Q must be independent. By the duality between independent sets and vertex covers, Q must be a vertex cover. Thus $w(S_1) \leq w(Q)$, since S_1 has minimum cost. Furthermore, note that every interval in S_2 is a singleton witness set, since S_2 is a set of independent intervals. Thus $w(S_2) \leq w(Q)$ as well, and $w(S_1 \cup S_2) \leq 2 \cdot w(Q)$. ◀

5 Improved Adaptive Algorithms for Uniform Query Costs

We now explore refined analysis of query-competitive sorting. We present a unified strategy that yields different improvements to Theorem 4.3, depending on what assumptions we make.

The core observation is that the bad 2-interval instance in the proof of Lemma 4.1 is the only structure that prevents an algorithm from performing better than twice the optimum. The first strategy that comes to mind, then, is to use randomization: a simple randomized strategy attains query-competitive factor 3/2 on the instance of Lemma 4.1. Before extending the algorithm to arbitrary instances, we give a lower bound for any randomized algorithm.

▶ **Lemma 5.1.** *Any randomized adaptive algorithm has query-competitive ratio at least* 3/2 *against an adversary that is oblivious to the randomized tosses, even for uniform query costs.*

Proof. Use the same bad instance as Lemma 4.1, set probability 1/2 for each of the two possible inputs and apply Yao's minimax principle. ◀

The algorithm is based on the following property of the dependency graph, whose proof we omit.

▶ **Lemma 5.2.** *If I_x is an interval with minimum r_x, then the vertex x is simplicial, i.e., its neighborhood is a clique.*

The algorithm begins by querying intervals that are singleton witness sets according to a generalization of the condition in Proposition 3.1. Then, if a component of the remaining dependency graph is an edge, the randomized strategy is applied. Else, the algorithm considers a non-queried vertex x with minimum r_x, a neighbor y of x with minimum r_y, and another neighbor z of x (or of y if y is the only neighbor of x) with minimum r_z. The algorithm first queries I_y. If x and y are still adjacent, or if x and z are adjacent, then we query both I_x and I_z. We repeat this strategy until the dependency graph has no edges.

A pseudocode is presented in Algorithm 1; we parameterize the probability p in the randomized strategy since the algorithm will be reused afterwards. We also maintain a set $\mathcal{V}$ of the values resulting of queried intervals.

Algorithm 1 Improved adaptive algorithm for the sorting problem with queries.

```
   Input: (I_1, ..., I_n, p)
1  V ← ∅;
2  while there are i, j with I_i ⊃ [ℓ_j − δ, r_j + δ] or I_i ⊃ [v_j − δ, v_j + δ] with v_j ∈ V do
3  |  query I_i, add v_i to V;
4  while there is some dependency do
5  |  if some component is an edge ij then
6  |  |  pick i with probability p (and j with probability 1 − p); assume i is picked;
7  |  |  query I_i, add v_i to V;
8  |  |  if I_j ⊃ [v_i − δ, v_i + δ] then
9  |  |  |  query I_j, add v_j to V;
10 |  else
11 |  |  let I_x non-queried with min r_x, and y be a neighbor of x with min r_y;
12 |  |  let z be another neighbor of x (or of y if x is a leaf), with min r_z;
13 |  |  query I_y, add v_y to V;
14 |  |  if I_x ⊃ [v_y − δ, v_y + δ] or I_x, I_z are dependent then
15 |  |  |  query I_x, add v_x to V;
16 |  |  |  query I_z, add v_z to V;
17 |  while there is I_i ⊃ [v_j − δ, v_j + δ] for some v_j ∈ V do
18 |  |  query I_i, add v_i to V;
```

▶ **Theorem 5.3.** *Algorithm 1 has expected query-competitive ratio* $3/2$ *if* $p = 1/2$.

Proof. We form a partition $V_1, \ldots, V_m$ of the set of input intervals with the following property. Let $a(V_i)$ be the number of intervals in V_i that are queried by the algorithm, and let $q(V_i) := |Q \cap V_i|$, where Q is an optimum query set. We show that $\mathbb{E}[a(V_i)/q(V_i)] \leq 3/2$ for every i, from which the theorem follows.

If the algorithm queries an interval I_i in Line 3 or Line 18, then $\{I_i\}$ is the next set in the partition. Due to Proposition 3.1, it is a singleton witness set, so $a(\{I_i\})/q(\{I_i\}) = 1$.

If the algorithm runs Lines 6–9 for edge ij, then $W = \{I_i, I_j\}$ is the next set in the partition. We consider the following cases.

1. If $I_i \supset [v_j - \delta, v_j + \delta]$ and $I_j \supset [v_i - \delta, v_i + \delta]$, then $q(W) = 2$ and $a(W) = 2$.
2. Otherwise, $q(W) \geq 1$ because this is a witness pair.
   a. If $I_i \supset [v_j - \delta, v_j + \delta]$ but $I_j \not\supset [v_i - \delta, v_i + \delta]$, then with probability $1/2$ the algorithm queries I_i and this is enough, and with probability $1/2$ it queries both, so $\mathbb{E}[a(W)] = 3/2$; the same holds for the symmetrical case.
   b. If $I_i \not\supset [v_j - \delta, v_j + \delta]$ and $I_j \not\supset [v_i - \delta, v_i + \delta]$, then Line 9 is not executed and $a(W) = 1$.

If the algorithm runs Lines 11–16 for x, y and z, then we have two cases.

1. If x and z are not neighbors, and x and y are not neighbors after Line 13, then $W = \{I_x, I_y\}$ is the next set in the partition. Since it is a witness set, $q(W) \geq 1$. But the algorithm will not query I_x because y is the only neighbor of x, so $a(W) = 1$.
2. Otherwise, $W = \{I_x, I_y, I_z\}$ is the next set in the partition. We have two subcases.
   a. If x and z are neighbors, then xyz is a clique by Lemma 5.2. So $q(W) \geq 2$, since otherwise a pair is unsolved.
   b. Otherwise, $I_x \supset [v_y - \delta, v_y + \delta]$ and $\{I_x\}$ is a singleton witness set. Since x and z are not neighbors, then y and z are neighbors and, by Lemma 2.2, $\{I_y, I_z\}$ is a witness set.

   Either way, $q(W) \geq 2$ and $a(W) = 3$.

We conclude that the expected query-competitive ratio is $3/2$. ◀

Our second strategy to obtain an improvement on Theorem 4.3 is, instead of using randomization, to assume that the graph does not have 2-components, i.e., components consisting of a single edge. This is not enough, however, since in Lemma 4.1 we have shown that we can have a large component. So our hypothesis is that $\delta = 0$ and, after executing the loop of Lines 2–3, the remaining dependency graph, which becomes a proper interval graph, has no 2-components. (Note that Theorem 5.3 is still true if we remove Lines 2–3 of the algorithm.) Let us prove a lower bound for this case.

▶ **Lemma 5.4.** *Any deterministic adaptive algorithm has query-competitive ratio at least* $5/3$*, even if* $\delta = 0$ *and the dependency graph is a proper interval graph with no* 2*-components.*

Proof. Consider five proper intervals I_a, I_b, I_c, I_d, I_e with $\ell_a < \ell_b < \ell_c < \ell_d < \ell_e$. The dependencies are defined by two triangles, *abc* and *cde*.

If the algorithm first queries I_c, then we set $v_c \in I_c \setminus (I_a \cup I_b \cup I_d \cup I_e)$, and we can make *ab* and *de* behave as the bad instance of Lemma 4.1.

If the algorithm first queries I_a, then we set $v_a \in I_b \cap I_c$, so the algorithm will be forced to query I_b and I_c, and we set $v_b, v_c \in (I_b \cup I_c) \setminus (I_a \cup I_d \cup I_e)$, so the optimum can avoid I_a. Then we can make *de* behave as the bad instance of Lemma 4.1. The argument is symmetric if the algorithm first queries I_e.

If the algorithm first queries I_b, then we set $v_b \in I_a \cap I_c$, so the algorithm will be forced to query I_a and I_c, and we set $v_a, v_c \in (I_a \cup I_c) \setminus (I_b \cup I_d \cup I_e)$, so the optimum can avoid I_b. Then we can make *de* behave as the bad instance of Lemma 4.1. The argument is symmetric if the algorithm first queries I_d. ◀

▶ **Theorem 5.5.** *Algorithm 1 (with* $p = 0$ *or* 1*) is* $5/3$*-query-competitive if* $\delta = 0$ *and the dependency graph has no* 2*-components after finishing the loop of Lines 2–3.*

Proof. The analysis is similar to that of Theorem 5.3. We will give a partition $V_1, \ldots, V_m$ of the set of intervals with the following property. Let $a(V_i)$ be the number of intervals in V_i that are queried by the algorithm, and let $q(V_i) := |Q \cap V_i|$, where Q is an optimum query set. We will have that $a(V_i)/q(V_i) \leq 5/3$ for every i, and then the theorem follows. The analysis for the cases of Lines 3, 11–16 and 18 are identical.

If the algorithm runs Lines 6–9 for edge ij, then let C be the component containing ij in the dependency graph after finishing the loop of Lines 2–3. We claim that i and j are the only vertices of C queried in Lines 6–9: Lines 11–16 force that intervals are queried from left to right; thus, since the dependency graph at this point is a proper interval graph, if an interval i' is queried in Line 18, then after that no interval j' with $r_{j'} < r_{i'}$ will have some dependency. Pick an arbitrary set W' of the partition consisting of vertices of C. We merge $\{I_i, I_j\}$ and W' into a single set W of the partition, and from the previous cases we have that $a(W)/q(W) \leq 5/3$. ◀

This proof indicates that the analysis can be improved if we require the graph to have *large* components after finishing the loop of Lines 2–3.

▶ **Theorem 5.6.** *Algorithm 1 (with* $p = 0$ *or* 1*) has query-competitive factor* $3/2 + \mathrm{O}(1/k)$ *if* $\delta = 0$ *and each component of the dependency graph has size at least* k *after finishing the loop of Lines 2–3.*

Proof. We only have to reconsider the case of Lines 6–9 in the proof of Theorem 5.5. If the algorithm runs Lines 6–9 for edge ij, then let C be the component containing ij in the dependency graph after finishing the loop of Lines 2–3. We merge $\{I_i, I_j\}$ and all partition

sets containing vertices of C into a single set W of the partition. Since i and j are the only vertices of C queried in Lines 6–9 and C has size at least k, from the other cases we have that $a(W)/q(W) \leq 3/2 + \mathrm{O}(1/k)$. ◀

The analysis is tight since we can have a chain of k triangles plus 1 edge, such that we can force the algorithm to query all intervals, while the optimum can avoid one interval in each triangle and one interval in the extra edge. For large components, we still have a lower bound of 7/5 for any deterministic algorithm.

▶ **Lemma 5.7.** *Any deterministic adaptive algorithm has query-competitive ratio at least* 7/5*, even if* $\delta = 0$ *and the dependency graph is a proper interval graph with large components.*

Proof. (Lemma 5.7.) Consider the graph of Figure 2, which has $7k + 2$ vertices. For $i = 0, \ldots, k-1$, vertices $7i + 3, \ldots, 7i + 7$ consist in a copy of the instance of Lemma 5.4. For $i = 0, \ldots, k$, vertices $x_i = 7i + 1$ and $y_i = 7i + 2$ are dependent, x_i is dependent to $7(i-1) + 7$ if $i > 0$, and y_i is dependent to $7i + 3$ if $i < k$. We set $v_{x_i}, v_{y_i} \in I_{x_i} \cap I_{y_i}$, so both the algorithm and the optimum must query I_{x_i} and I_{y_i}, but querying them gives us no information about the remaining vertices. From Lemma 5.4, we can force any deterministic algorithm to query all vertices in the graph, while the optimum solution can query only 3 vertices of $7i + 3, \ldots, 7i + 7$. ◀

Figure 2 Instance which attains the lower bound for proper interval graphs with large components.

It remains an open question to close the gap between the lower bound of 7/5 and the upper bound of $3/2 + \mathrm{O}(1/k)$. Finally, we note that the problem can be solved exactly for laminar families of intervals, since all queries will happen at Line 3 of the algorithm.

▶ **Theorem 5.8.** *Algorithm 1 obtains an optimum solution if* $\delta = 0$ *and the intervals constitute a laminar family.*

6 Advice Complexity for Adaptive Algorithms

In this section we investigate the advice complexity of solving the adaptive version of the problem. We assume arbitrary query costs, and for consistency that the oracle answers questions regarding a fixed optimum solution for the given instance.

First, we deal with the case when $\delta = 0$. Let n be the number of given intervals. We claim that $\lfloor n/2 \rfloor$ bits of advice are sufficient to solve the problem exactly, and that there are instances for which $\lfloor n/2 \rfloor$ bits are necessary.

▶ **Lemma 6.1.** *The advice complexity of the adaptive sorting problem with uncertainty is at least* $\lfloor n/2 \rfloor$*, where* n *is the number of intervals, even if* $\delta = 0$*.*

Proof. Assume n even and consider $n/2$ independent copies of the bad instance of Lemma 4.1. At least 1 bit of advice is necessary to decide the relative order between each pair. ◀

For an adaptive algorithm with a matching upper bound, we note that, if $\delta = 0$, then any triangle ijk contains a vertex j such that $I_j \subseteq I_i \cup I_k$ (just take i with minimum ℓ_i and k with maximum r_k). Thus, we can ask the oracle whether the optimum solution queries I_j; if not, then we must query all neighbors of j; otherwise, we query I_j, and since $I_j \subseteq I_i \cup I_k$,

we will know at least one of I_i and I_k that also must be queried. If the dependency graph contains no triangles, then it is a forest, because any cycle in a chordal graph must contain a triangle. Therefore, we can pick a leaf i and ask the oracle whether the optimum solution queries its neighbor j; if not, then we query all neighbors of j; otherwise, we query I_j and we will know if I_i must or not be queried. Since we decide at least two intervals with one bit of advice, then $\lfloor n/2 \rfloor$ bits are sufficient. We present a pseudocode in Algorithm 2.

Algorithm 2 An adaptive algorithm that finds an optimum solution with $\lfloor n/2 \rfloor$ bits of advice when $\delta = 0$.

```
   Input: (I_1, ..., I_n)
1  V ← ∅;
2  while there is some dependency do
3  |  if there is a triangle K then
4  |  |  let i ∈ K with minimum ℓ_i, k ∈ K with maximum r_k, and j ∈ K \ {i, k};
5  |  else let i be a leaf, and j be the neighbor of i ;
6  |  ask the oracle whether the optimum solution queries j;
7  |  if yes then query I_j, add v_j to V;
8  |  else foreach neighbor z of j do
9  |  |  query I_z, add v_z to V;
10 |  while there is I_i ⊃ [v_j − δ, v_j + δ] for some v_j ∈ V do
11 |  |  query I_i, add v_i to V;
```

▶ **Theorem 6.2.** *The advice complexity of the adaptive sorting problem with uncertainty is* $\lfloor n/2 \rfloor$ *when* $\delta = 0$*, where* n *is the number of intervals.*

Now we consider the case when $\delta > 0$. Here, we can improve the lower bound to $\lceil n/3 \cdot \lg 3 \rceil$ and still have an algorithm with matching upper bound. Both are based on the fact that to encode k distinct values amortized $\lg k$ bits are sufficient and necessary [27].

▶ **Lemma 6.3.** *The advice complexity of the adaptive sorting problem with uncertainty is at least* $\lceil n/3 \cdot \lg 3 \rceil$*, where* n *is the number of intervals.*

Proof. Assume n multiple of 3 and consider $n/3$ independent triangles; it suffices to bound the number of bits of advice necessary to solve each triangle. Suppose by contradiction that there is an algorithm that solves any triangle with one bit of advice, and consider the following instances $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3$. In each $\mathcal{I}_i$, the k-th triangle has intervals I_1, I_2, I_3 such that $\ell_1 < \ell_2 < \ell_3 - \delta$, $r_1 + \delta < r_2 < r_3$, $\ell_2 \leq \ell_1 + \delta$, $r_2 \geq r_3 - \delta$ and $r_1 - \ell_3 > 2\delta$. We have $v_j = r_j$ for all j in $\mathcal{I}_1$; in $\mathcal{I}_2$, $v_1 = \ell_1$, $v_2 \in (\ell_3 + \delta, r_1 - \delta)$, $v_3 = r_3$; and $v_j = \ell_j$ for all j in $\mathcal{I}_3$. The only optimum solution for $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3$ is not to query I_1, I_2, I_3, respectively. (See Figure 3.)

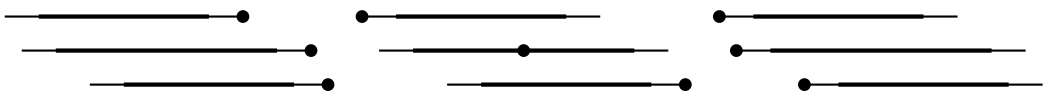

Figure 3 Instances for the lower bound on advice complexity when $\delta > 0$.

By the pigeonhole principle, the algorithm must have the same advice for at least two of those inputs. So, it suffices to prove that any deterministic algorithm fails in one instance of any subset with at least two of those instances. Since the intervals are structurally identical, any algorithm for a triangle performs no better than an algorithm in the following form, for

fixed $x, y \in \{1, 2, 3\}$, $x \neq y$: query I_x, and if no helpful information is given, query I_y. The instances are constructed in such a way that, for instance $\mathcal{I}_i$, the algorithm does not get any helpful information by querying I_x with $i \neq x$, so it fails on instances $\mathcal{I}_x$ and $\mathcal{I}_y$. Since one bit is not sufficient, at least three different values must be encoded in the advice for each triangle, so $\lceil n/3 \cdot \lg 3 \rceil$ bits are necessary for the whole instance. ◀

The algorithm that attains the upper bound relies on Lemma 5.2. It considers the clique K consisting of vertex x with minimum r_x and its neighborhood. Then it asks the oracle for the index of a vertex y in K that is not queried in the optimum solution or, if there is no such vertex in K, then the oracle must return $y = x$. Either way, the algorithm queries all intervals in $K \setminus \{y\}$, and if $y = x$ then the algorithm will know if y must also be queried after querying everyone else. So it uses $\lg |K|$ bits of advice to decide at least $|K|$ intervals, and the bound follows since $\lg k/k$ has its maximum at $k = 3$ when k is integer. A pseudocode is presented in Algorithm 3.

Algorithm 3 An adaptive algorithm that finds an optimum solution with $\lceil n/3 \cdot \lg 3 \rceil$ bits of advice.

```
Input: (I_1, ..., I_n)
V ← ∅;
while there is some dependency do
    let x with minimum r_x, and K be the clique consisting of x and its neighborhood;
    ask the oracle for a vertex y ∈ K not queried in the optimum solution, or y = x if
      there is no such vertex;
    foreach z ∈ K \ {y} do
        query I_z, add v_z to V;
    while there is I_i ⊃ [v_j − δ, v_j + δ] for some v_j ∈ V do
        query I_i, add v_i to V;
```

▶ **Theorem 6.4.** *The advice complexity of the adaptive sorting problem with uncertainty is* $\lceil n/3 \cdot \lg 3 \rceil$*, where* n *is the number of intervals.*

7 Future Work Directions

One interesting question is how the sorting problem can take advantage of queries with different levels of precision, as in [21]. A variety of other classical optimization problems could also be studied in similar uncertainty variants with queries.

References

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

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

3 H.-G. Beyer and B. Sendhoff. Robust optimization – a comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33-34):3190–3218, 2007. doi:10.1016/j.cma.2007.03.003.

4 J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research and Financial Engineering. Springer, 2011.

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

6 J. Boyar, L. M. Favrholdt, C. Kudahl, K. S. Larsen, and J. W. Mikkelsen. Online Algorithms with Advice: A Survey. *ACM Computing Surveys*, 50(2), 2017. doi:10.1145/3056461.

7 R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient Update Strategies for Geometric Computing with Uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005. doi:10.1007/s00224-004-1180-4.

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

9 C. Dürr, T. Erlebach, N. Megow, and J. Meißner. Scheduling with Explorable Uncertainty. In A. R. Karlin, editor, *ITCS 2018: 9th Innovations in Theoretical Computer Science Conference*, volume 94 of *Leibniz International Proceedings in Informatics*, pages 30:1–30:14, 2018. doi:10.4230/LIPIcs.ITCS.2018.30.

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

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

12 T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihal'Ák, and R. Raman. Computing minimum spanning trees with uncertainty. In *STACS'08: 25th International Symposium on Theoretical Aspects of Computer Science*, pages 277–288, 2008. URL: https://arxiv.org/abs/0802.2855.

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

14 T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32(2):538–547, 2003. doi:10.1137/S0097539701395668.

15 J. Focke, N. Megow, and J. Meißner. Minimum Spanning Tree under Explorable Uncertainty in Theory and Experiments. In C. S. Iliopoulos, S. P. Pissis, S. J. Puglisi, and R. Raman, editors, *SEA 2017: 16th International Symposium on Experimental Algorithms*, volume 75 of *Leibniz International Proceedings in Informatics*, pages 22:1–22:14, 2017. doi:10.4230/LIPIcs.SEA.2017.22.

16 F. Gavril. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph. *SIAM Journal on Computing*, 1(2):180–187, 1972. doi:10.1137/0201013.

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

18 M. Gupta, Y. Sabharwal, and S. Sen. The Update Complexity of Selection and Related Problems. *Theory of Computing Systems*, 59(1):112–132, 2016. doi:10.1007/s00224-015-9664-y.

19 C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962. doi:10.1093/comjnl/5.1.10.

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

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

22 N. Megow, J. Meißner, and M. Skutella. Randomization Helps Computing a Minimum Spanning Tree under Uncertainty. *SIAM Journal on Computing*, 46(4):1217–1240, 2017. doi:10.1137/16M1088375.

23 C. L. Monma, B. Reed, and W.T. Trotter Jr. Threshold Tolerance Graphs. *Journal of Graph Theory*, 12(3):343–362, 1988. doi:10.1002/jgt.3190120307.

24 C. Olston and J. Widom. Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data. In *VLDB 2000: 26th International Conference on Very Large Data Bases*, pages 144–155, 2000. URL: http://ilpubs.stanford.edu:8090/437/.

25 I. O. Ryzhov and W. B. Powell. Information Collection for Linear Programs with Uncertain Objective Coefficients. *SIAM Journal on Optimization*, 22(4):1344–1368, 2012. doi:10.1137/12086279X.

26 A. Salah, K. Li, and K. Li. Lazy-Merge: A Novel Implementation for Indexed Parallel k-Way In-Place Merging. *IEEE Transactions on Parallel and Distributed Systems*, 27(7):2049–2061, 2015. doi:10.1109/TPDS.2015.2475763.

27 C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

28 Y. Yamaguchi and T. Maehara. Stochastic Packing Integer Programs with Few Queries. In *SODA'18: Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 293–310, 2018. doi:10.1137/1.9781611975031.21.