Query Minimization under Stochastic Uncertainty*

Steven Chaplick[1], Magnús M. Halldórsson[2], Murilo S. de Lima[3], and Tigran Tonoyan[4]

[1] Lehrstuhl für Informatik I, Universität Würzburg, Germany and Department of Data Science and Knowledge Engineering, Maastricht University, the Netherlands
[2] ICE-TCS, Department of Computer Science, Reykjavik University, Iceland
[3] School of Informatics, University of Leicester, UK
[4] Computer Science Department, Technion Institute of Technology, Israel
`s.chaplick@maastrichtuniversity.nl`, `mmh@ru.is`, `mslima@ic.unicamp.br`, `ttonoyan@gmail.com`

Abstract. We study problems with stochastic uncertainty data on intervals for which the precise value can be queried by paying a cost. The goal is to devise an adaptive decision tree to find a correct solution to the problem in consideration while minimizing the expected total query cost. We show that sorting in this scenario can be performed in polynomial time, while finding the data item with minimum value seems to be hard. This contradicts intuition, since the minimum problem is easier both in the online setting with adversarial inputs and in the offline verification setting. However, the stochastic assumption can be leveraged to beat both deterministic and randomized approximation lower bounds for the online setting. Although some literature has been devoted to minimizing query/probing costs when solving uncertainty problems with stochastic input, none of them have considered the setting we describe. Our approach is closer to the study of query-competitive algorithms, and it gives a better perspective on the impact of the stochastic assumption.

Keywords: stochastic optimization · query minimization · sorting · selection · online algorithms

1 Introduction

Consider the problem of sorting n data items that are updated concurrently by different processes in a distributed system. Traditionally, one ensures that the data is strictly consistent, e.g., by assigning a master database that is queried by the other processes, or by running a distributed consensus algorithm. However, those operations are expensive, and we wonder if we could somehow avoid them. One different approach has been proposed for the TRAPP distributed database

* Partially supported by Icelandic Research Fund grant 174484-051 and by EPSRC grant EP/S033483/1. This work started while M.S.L. and T.T. were at Reykjavik University, during a research visit by S.C.

by Olston and Widom [15], and is outlined as follows. Every update is sent to the other processes, and each process maintains an interval on which each data item may lie. Whenever the precise value is necessary, a query on the master database can be performed. Some computations (e.g., sorting) can be performed without knowing the precise value of all data items, so one question that arises is how to perform these while minimizing the total query cost. Another scenario in which this type of problem arises is when market research is required to estimate the data input: a coarser estimation can be performed for a low cost, and more precise information can be obtained by spending more effort in research. The problem of sorting under such conditions, called the **uncertainty sorting problem with query minimization**, was recently studied by Halldórsson and de Lima [12].

The study of uncertainty problems with query minimization dates back to the seminal work of Kahan [13] and the TRAPP distributed database system by Olston and Widom [15], which dealt with simple problems such as computing the minimum and the sum of numerical data with uncertainty intervals. More recently, more sophisticated problems have been studied in this framework, such as geometric problems [1], shortest paths [6], minimum spanning tree and minimum matroid base [5, 14], linear programming [16, 19], and NP-hard problems such as the knapsack [8] and scheduling problems [3]. See [4] for a survey.

The literature describes two kinds of algorithms for this setting. Though the nomenclature varies, we adopt the following one. 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.

Two main approaches have been proposed to analyze both types of algorithms. In the first, an oblivious (adaptive) algorithm is compared to a hypothetical optimal oblivious (adaptive) strategy; this is the approach in [6, 13, 15]. However, for more complex problems, and in particular for adaptive algorithms, it usually becomes more difficult to understand the optimal adaptive strategy. A second (more robust) approach is competitive analysis, which is a standardized metric for online optimization. In this setting, both oblivious and adaptive algorithms are compared to an **optimum query set**, a minimum-cost set of queries that a clairvoyant adversary, who knows the actual values but cannot disclose them without performing a query, can use to prove the obtained solution to be correct. 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. This type of analysis is performed in [1, 5, 11–14]. For NP-hard problems, since we do not expect to find the "correct" solution in polynomial time, there are two approaches in the literature: either we have an objective function which combines query and solution costs (this is how the scheduling problem is addressed in [3]), or we have a fixed query budget and the objective function is based only on the solution cost (as for the knapsack problem in [8]).

Competitive analysis is, however, rather pessimistic. In particular, many problems such as minimum, sorting and spanning tree have a deterministic lower bound of 2 and a randomized lower bound of 1.5 for adaptive algorithms, and a simple 2-competitive deterministic adaptive algorithm, even if queries are allowed to return intervals [5, 11, 12, 14]. For the sorting problem, e.g., Halldórsson and de Lima [12] showed that there is essentially one structure preventing a deterministic adaptive algorithm from performing better than 2.

One natural alternative to competitive analysis is to assume stochastic inputs, i.e., that the precise value in each interval follows a known probability distribution, and we want to build a decision tree specifying a priority ordering for querying the intervals until the correct solution is found, so that the expected total query cost is minimized.[5] In this paper, we show that the adaptive sorting problem in this setting can be solved exactly in polynomial time. Very surprisingly, however, we have evidence that the problem of finding the data item with minimum value is hard, though it can be approximated very well.

Some literature is devoted to a similar goal of this paper, but we argue that there are some essential differences. One first line of work consists of the **stochastic probing problem** [7, 9, 10, 17], which is a general stochastic optimization problem with queries. Even though those works presented results for wide classes of constraints (such as matroid and submodular), they differ in two ways from our work. First, they assume that a solution can only contain elements that are queried, or that the objective function is based on the expectation of the non-queried elements. Second, the objective function is either a combination of the solution and query costs, or there is a fixed budget for performing queries. Since most of these variants are NP-hard [7], some papers [9, 17] focused on devising approximation algorithms, while others [7, 10] on bounding the ratio between an oblivious algorithm and an optimal adaptive algorithm (the **adaptive gap**). Another very close work is that of Welz [18, Section 5.3] and Yamaguchi and Maehara [19], which, like us, assume that a solution may contain non-queried items. Welz presented some results for the minimum spanning tree and traveling salesman problems, but they make strong assumptions on the probability distributions, while Yamaguchi and Maehara devised algorithms for a wide class of problems, which also yield improved approximation algorithms for some classical stochastic optimization problems. However, both works focus on obtaining approximate solutions, while we wish to obtain an exact one, and they only give asymptotic bounds on the number of queries performed, but do not compare this to the expected cost of an optimum query set. To sum up, our work gives a better understanding on how the stochastic assumption differs from the competitive analysis, since other assumptions are preserved and we use the same metric to analyze the algorithms: minimizing query cost while finding the correct solution.

Our results. We prove that, for the sorting problem with stochastic uncertainty, we can construct an adaptive decision tree with minimum expected query cost in polynomial time. We devise a dynamic programming algorithm which runs in time $O(n^3 d^3) = O(n^6)$, where d is the clique number of the interval graph

[5] Note that, unless some sort of nondeterminism is allowed, the stochastic assumption cannot be used to improve the oblivious results, so we focus on adaptive algorithms.

induced by the uncertainty intervals. We then discuss why simpler strategies fail, such as greedy algorithms using only local information, or relying on witness sets, which is a standard technique for solving query-minimization problems with adversarial inputs [1, 5]. We also discuss why we believe that the dynamic programming algorithm cannot be improved to something better than $O(n^3)$.

Surprisingly, on the other hand, we present evidence that finding an adaptive decision tree with minimum expected query cost for the problem of finding the data item with minimum value is hard, although the online version (with adversarial inputs) and the offline (verification) version of the problem are rather simple. If the leftmost interval is the first one to be queried, we know how to compute the decision tree with minimum expected query cost easily. This also implies that, for any other decision tree, one branch can be calculated easily. However, if the leftmost interval is not the first to be queried, we prove that it should be the last one to be considered in the decision tree. The hard part, then, is to find the order in which the other intervals are considered in the "hard branch" of the decision tree. We discuss why various heuristics fail to this case. A simple approximation result with factor $1 + 1/d_1$ for uniform query costs, where d_1 is the degree of the leftmost interval in the interval graph, follows from the online version with adversarial inputs [13]. For arbitrary query costs, we show that the stochastic assumption can be used to beat both deterministic and randomized lower bounds for the online version with adversarial inputs.

Organization of the paper. Section 2 is devoted to the sorting problem with stochastic uncertainty, and Section 3 to the problem of finding the minimum data item. We conclude the paper with future research questions in Section 4.

2 Sorting

The problem is to sort n numbers $v_1, \dots, v_n \in \mathbb{R}$ whose actual values are unknown. We are given n intervals $I_1, \dots, I_n$ such that $v_i \in I_i = [\ell_i, r_i]$. We can query interval I_i by paying a cost w_i, and after that we know the value of v_i. We want to find a permutation $\pi : [n] \to [n]$ such that $v_i \leq v_j$ if $\pi(i) < \pi(j)$ by performing a minimum-cost set of queries. We focus on adaptive algorithms, i.e., we can make decisions based on previous queries. We are interested in a stochastic variant of this problem in which v_i follows some known probability distribution on I_i. The only constraints are that (1) values in different intervals have independent probabilities, and (2) for any subinterval $(a, b) \subseteq I_i$, we can calculate $\mathbf{Pr}[v_i \in (a, b)]$ in constant time. The goal is to devise a strategy (i.e., a decision tree) to query the intervals so that the expected query cost is minimized. More precisely, this decision tree must tell us which interval to query first and, depending on where its value falls, which interval to query second, and so on.

Definition 1. *Two intervals I_i and I_j such that $r_i > \ell_j$ and $r_j > \ell_i$ are* **dependent**. *Two intervals that are not dependent are* **independent**.

The following lemma and proposition are proved in [12]. The lemma tells us that we have to remove all dependencies in order to be able to sort the numbers.

Lemma 1 ([12]). *The relative order between two intervals can be decided without querying either of them if and only if they are independent.*

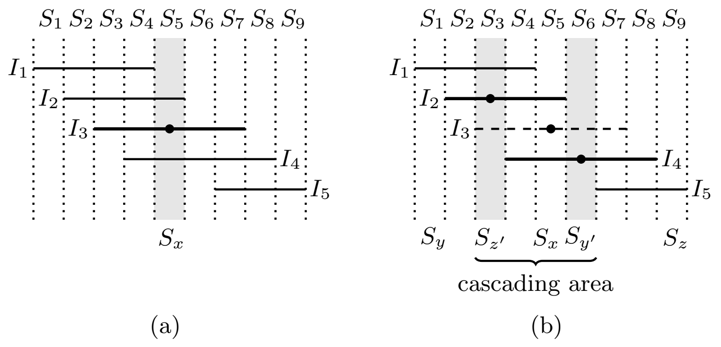
Proposition 1 ([12]). *Let I_i and I_j be intervals with actual values v_i and v_j. If $v_i \in I_j$ (and, in particular, when $I_i \subseteq I_j$), then I_j is queried by every solution.*

Note that the dependency relation defines an interval graph. Prop. 1 implies that we can immediately query any interval containing another interval, hence we may assume a proper interval graph. We may also assume the graph is connected, since the problem is independent for each component, and that there are no single-point intervals, as they would give a non-proper or disconnected graph.

An Optimal Algorithm. We describe a dynamic programming algorithm to solve the sorting problem with stochastic uncertainty. Since we have a proper interval graph, we assume intervals are in the natural total order, with $\ell_1 \leq \cdots \leq \ell_n$ and $r_1 \leq \cdots \leq r_n$. We also pre-compute the **regions** $S_1, \ldots, S_t$ defined by the intervals, where $t \leq 2n - 1$. A region is the interval between two consecutive points in the set $\bigcup_{i=1}^{n} \{\ell_i, r_i\}$; we assume that the regions are ordered. We write $S_x = (a_x, b_x)$ with $a_x < b_x$, and we denote by $\mathcal{I}_x(y,z) := \{i : S_x \subseteq I_i \subseteq (a_y, b_z)\}$ the indices of the intervals contained in (a_y, b_z) that contain S_x. For simplicity we assume that, for any interval I_i and any region S_x, $\mathbf{Pr}[v_i = a_x] = \mathbf{Pr}[v_i = b_x] = 0$; this is natural for continuous probability distributions, and for discrete distributions we may slightly perturb the distribution support so that this is enforced. Since the dependency graph is a connected proper interval graph, we can also assume that each interval contains at least two regions.

Before explaining the recurrence, we first examine how Prop. 1 limits our choices with an example. In Fig. 1(a), suppose we first decide to query I_3 and its value falls in region S_5. Due to Prop. 1, all intervals that contain S_5, namely I_2 and I_4, have to be queried as well. In Fig. 1(b), we assume that v_2 falls in S_3 and v_4 falls in S_6. This forces us to query I_1 but also results in a solution without querying I_5. Therefore, each time we approach a subproblem by first querying an interval I_i whose value falls in region S_x, we are forced to query all other intervals that contain S_x, and so on in a cascading fashion, until we end up with subproblems that are independent of current queried values. To find the best solution, we must pick a first interval to query, and then recursively calculate the cost of the best solution, depending on the region in which its value falls. Here, the proper interval graph can be leveraged by having the cascading procedure follow the natural order of the intervals.

We solve the problem by computing three tables. The first table, M, is indexed by two regions $y, z \in \{1, \ldots, t\}$, and $M[y,z]$ is the minimum expected query cost for the subinstance defined by the intervals contained in (a_y, b_z). Thus, the value of the optimum solution for the whole problem is $M[1,t]$. To compute $M[y,z]$, we suppose the first interval in (a_y, b_z) that is queried by the optimum solution is I_i. Then, for each region $S_x \subseteq I_i$, when $v_i \in S_x$, we are forced to query every interval I_j with $j \in \mathcal{I}_x(y,z)$ and this cascades, forcing other intervals to be queried depending on where v_j falls. So we assume that,



S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9
I_1
I_2
I_3
I_4
I_5
S_x
(a)
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9
I_1
I_2
I_3
I_4
I_5
S_y
$S_{z'}$
S_x
$S_{y'}$
S_z
cascading area
(b)

Fig. 1. A simulation of the querying process for a fixed realization of the values. (a) Querying I_3 first and assuming $v_3 \in S_5$. (b) Assuming $v_2 \in S_3$ and $v_4 \in S_6$.

for all $j \in \mathcal{I}_x(y,z)$, v_j falls in the area defined by regions $z', z'+1, \ldots, y'-1, y'$, with $z' \le x \le y'$, and that this area is minimal (i.e., some point is in $S_{z'}$, and some point is in $S_{y'}$). We call this interval $(a_{z'}, b_{y'})$ the **cascading area** of I_i in $\mathcal{I}_x(y,z)$. In Fig. 1(b), we have $i = 3$, $x = 5$, $z' = 3$ and $y' = 6$. As the dependency graph is a proper interval graph, the remaining intervals (which do not contain S_x) are split in two independent parts, whose value is computed by two tables, L and R, which we describe next. So the recurrence for $M[y,z]$ is

$$\begin{cases} 0, \quad \text{if } (a_y, b_z) \text{ contains less than 2 intervals; otherwise,} \\ \underbrace{\min_{I_i \subseteq (a_y,b_z)}}_{\substack{\text{first interval}\\ \text{to query}}} \underbrace{\sum_{S_x \subseteq I_i} \mathbf{Pr}[v_i \in S_x] \cdot}_{\text{where point } v_i \text{ falls}} \left(\overbrace{\sum_{j \in \mathcal{I}_x(y,z)} w_j}^{\text{cost of cascading}} + \sum_{\substack{z' \le x \\ y' \ge x}} \overbrace{p(y,z,i,x,z',y')}^{\text{cascading area}} \cdot \underbrace{\cdot \begin{pmatrix} L[y,z',\min \mathcal{I}_x(y,z)] + \\ +R[y',z,\max \mathcal{I}_x(y,z)] \end{pmatrix}}_{\text{cost of left/right subproblems}} \right) \end{cases},$$

where $p(y,z,i,x,z',y')$ is the probability that $(a_{z'}, b_{y'})$ is the cascading area of I_i in $\mathcal{I}_x(y,z)$. We omit the description of how to calculate this probability.

The definitions of L and R are symmetric, so we focus on L. For region indices y, z, z' with $z \ge z'$, let $I_{j'}$ be the leftmost interval contained in (a_y, b_z). Now, $L[y,z',j]$ is the minimum expected query cost of solving the subproblem consisting of intervals $I_{j'}, I_{j'+1}, \ldots, I_{j-1}$, assuming that a previously queried point lies in the region $S_{z'}$. We ensure that z' is the leftmost region in (a_y, b_z) that contains a queried point so that we query all intervals that contain some point. For example, in Fig. 1(b), after querying I_2, I_3 and I_4, the left subproblem has $z' = 3$ and $j = 2$. It holds that L can be calculated in the following way. If no interval before I_j contains $S_{z'}$, then the cascading is finished and we can refer to table M for regions $y, y+1, \ldots, z'-1$. Otherwise I_{j-1} must contain $S_{z'}$, we query it, and either v_{j-1} falls to the right of $\ell_{z'}$ and we proceed to the next

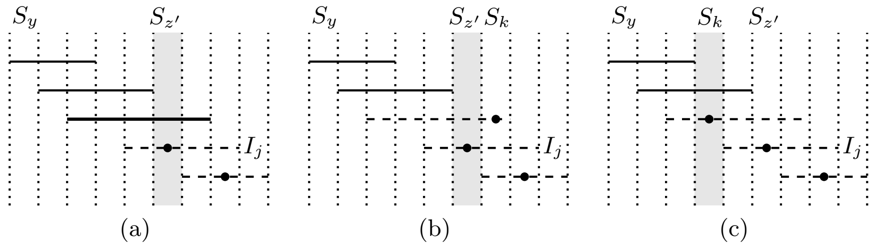


Fig. 2. An illustration of the definition of table L. (a) $L[y, z', j]$. (b) If $k \geq z'$, we recurse on $L[y, z', j-1]$. (c) If $k < z'$, we recurse on $L[y, k, j-1]$.

interval, or v_{j-1} falls in a region S_k with $k < z'$, and we proceed to the next interval with the leftmost queried point now being in S_k. Thus, we have

$$L[y, z', j] = \begin{cases} M[y, z'-1], & \text{if } j \leq 1 \text{ or } \ell_{j-1} < a_y \text{ or } I_{j-1} \not\supseteq S_{z'} \\ w_{j-1} + \displaystyle\sum_{S_k \subseteq I_{j-1}} \mathbf{Pr}[v_{j-1} \in S_k] \cdot L[y, \min(k, z'), j-1], & \text{otherwise.} \end{cases}$$

We illustrate this in Fig. 2. In Fig. 2(a), the subproblem contains $I_{j-1}, I_{j-2}, \ldots$, and the leftmost queried point is in $S_{z'}$. Since $S_{z'} \subseteq I_{j-1}$, we query I_{j-1} and assume v_{j-1} falls in a region S_k. In Fig. 2(b), we have that $k \geq z'$, so we recurse on $L[y, z', j-1]$; this will recurse on $M[y, z'-1]$ in its turn, since $S_{z'} \not\subseteq I_{j-2}$. In Fig. 2(c), we have that $k < z'$, so we recurse on $L[y, k, j-1]$, which in its turn will have to query I_{j-2}.

At this point it is not hard to see that the next theorem follows by a standard optimal substructure argument; we omit the proof.

Theorem 1. *$M[1, t]$ is the value of the minimum expected query cost to solve the stochastic sorting problem with uncertainty.*

The recurrences can be implemented in a bottom-up fashion that consumes time $O(n^6)$: if we precompute $p(y, z, i, x, z', y')$, then each entry of M is computed in $O(n^4)$, and each entry of L and R can be computed in linear time. It is possible to precompute $p(y, z, i, x, z', y')$ in time $O(n^4)$ (we omit how). A more careful analysis shows that the time consumption is $O(n^3 d^3)$, where d is the clique number of the interval graph. Note that, in a proper interval graph, an interval contains at most $2d - 1$ regions. Another simple fact is that $\mathcal{I}_x(y, z)$ contains at most d intervals, since every such interval contains S_x.

It seems difficult to improve this dynamic programming algorithm to something better than $O(n^3 \cdot \text{poly}(d))$. Note that the main information that the decision tree encodes is which interval should be queried first in a given independent subproblem (and there are $\Omega(n^2)$ such subproblems). We could hope to find an optimal substructure that would not need to test every interval as a first query, and that this information could somehow be inferred from smaller subproblems. However, consider $I_1 = (0, 100)$, $I_2 = (6, 105)$, and $I_3 = (95, 198)$, with uniform query costs and uniform probability distributions. The optimum solution for the first two intervals is to query first I_1, but the optimum solution for the whole instance is to start with I_2. Thus, even though I_2 is a suboptimal first query for the smaller subproblem, it is the optimal first query for the whole instance. This example could be adapted to a larger instance with more than d intervals, so that we need at least a linear pass in n to identify the best first query.

Simpler strategies that fail. It may seem that our dynamic programming strategy above is overly complex, and that a simpler algorithm may suffice to solve the problem. Below, we show sub-optimality of two such strategies.

We begin by showing that any greedy strategy that only takes into consideration local information (such as degree in the dependency graph or overlap area) fails. Consider a 5-path *abcde*, in which each interval has query cost 1 and an overlap of 1/3 with each of its neighbors, and the exact value is uniformly distributed in each interval. It can be shown by direct calculation that if we query I_b (or, equivalently, I_d) first, then we get an expected query cost of at most $29/9 = 3.2\bar{2}$, while querying I_c first yields an expected query cost of at least $11/3 = 3.6\bar{6}$. However, a greedy strategy that only takes into consideration local information cannot distinguish between I_b and I_c.

One technique that has been frequently applied in the literature of uncertainty problems with query minimization is the use of **witness sets**. A set of intervals W is a witness if a correct solution cannot be computed unless at least one interval in W is queried, even if all other intervals not in W are queried. Witness sets are broadly adopted because they simplify the design of query-competitive adaptive algorithms. If, at every step, an algorithm queries disjoint witness sets of size at most α, then this algorithm is α-query-competitive. This concept was proposed in [1]. For the sorting problem, by Lemma 1, any pair of dependant intervals constitute a witness set. However, we cannot take advantage of witness sets for the stochastic version of the problem, even for uniform query costs and uniform probability distributions, and even if we take advantage of the proper interval order. Consider the following intervals: $(0, 100), (95, 105), (98, 198)$. The witness set consisting of the first two intervals may lead us to think that either of them is a good choice as the first query. However, the unique optimum solution first queries the third interval. (The costs are $843/400 = 2.1075$ if we query first the first interval, $277/125 = 2.216$ if we query first the second interval, and $4182/2000 = 2.0915$ if we query first the third interval.)

3 Finding the Minimum

We also consider the problem of finding the minimum (or, equivalently, the maximum) of n unknown values $v_1, \ldots, v_n$. Assume that the intervals are sorted by the left corner, i.e., $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_n$. We may assume without loss of generality that $\ell_1 < \ell_2 < \cdots < \ell_n$. Let $\mathcal{I} = \{I_1, \ldots, I_n\}$. We begin by discussing some assumptions we can make. First, we can assume that the interval graph is a clique: with two independent intervals, we can remove the one on the right.

(However, we cannot assume a proper interval graph, as we did for sorting.) The second assumption is based on the following remark, whose proof we omit.

Remark 1. If I_1 contains some I_j, then I_1 is queried in every solution.

Thus we can assume that I_1 does not contain another interval; this implies that $r_1 = \min_i r_i$. It is also useful to understand how to find an optimum query set, i.e., to solve the problem assuming we know $v_1, \ldots, v_n$.

Lemma 2 ([13]). *The offline optimum solution either*

(a) queries interval I_i with minimum v_i and each interval I_j with $\ell_j < v_i$; or
(b) queries all intervals except for I_1, if v_1 is the minimum, $v_j > r_1$ for all $j > 1$, and this is better than option (a).

Option (b) can be better not only due to a particular non-uniform query cost configuration, but also with uniform query costs, when $v_1 \in I_2, \ldots, I_n$. Note also that I_1 is always queried in option (a). We omit the proof of this lemma.

We first discuss what happens if the first interval we query is I_1. In Fig. 3(a), we suppose that $v_1 \in S_3$. This makes I_2 become the leftmost interval, so it must be queried, since it contains v_1 and I_3. At this point we also know that we do not need to query I_4, since $v_1 < \ell_4$. After querying I_2, we have two possibilities. In Fig. 3(b), we suppose that $v_2 \in S_2$, so we already know that v_2 is the minimum and no other queries are necessary. In Fig. 3(c), we suppose that $v_2 \in S_6$, so we still need to query I_3 to decide if v_1 or v_3 is the minimum. Note that, once I_1 has been queried, we do not have to guess which interval to query next, since any interval that becomes the leftmost interval will either contain v_1 or will be to the right of v_1. Since this is an easy case of the problem, we formalize how to solve it. The following claim is clear: if we have already queried $I_1, \ldots, I_{i-1}$ and $v_1, \ldots, v_{i-1} \in I_i$, then we have to query I_i. (This relies on I_i having minimum ℓ_i among $I_i, \ldots, I_n$.) If we decide to first query I_1, then we are discarding option (b) in the offline solution, so all intervals containing the minimum value must be queried. The expected query cost is then $\sum_{i=1}^{n} w_i \cdot \mathbf{Pr}[I_i \text{ must be queried}]$. Given an interval I_i, it will not need to be queried if there is some I_j with $v_j < \ell_i$, thus the former probability is the probability that no value lies to the left of I_i. Since the probability distribution is independent for each interval, the expected query cost will be $\sum_{i=1}^{n} w_i \cdot \prod_{j<i} \mathbf{Pr}[v_j > \ell_i]$. This can be computed in $\mathrm{O}(n^2)$ time.

Now let us consider what happens if the optimum solution does not start by querying I_1, but by querying some I_k with $k > 1$. When we query I_k, we have two cases: (1) if v_k falls in I_1, then we have to query I_1 and proceed as discussed above, querying I_2 if $v_1 > \ell_2$, then querying I_3 if $v_1, v_2 > \ell_3$ and so on; (2) if $v_k \notin I_1$, then v_k falls to the right of ℓ_i, for all $i \neq k$, so essentially the problem consists of finding the optimum solution for the remaining intervals, and this value will be independent of v_k. Therefore, the cost of querying first I_k is $w_k + \mathbf{Pr}[v_k \notin I_1] \cdot \mathrm{opt}(\mathcal{I} \setminus \{I_k\}) + \mathbf{Pr}[v_k \in I_1] \cdot \sum_{i \neq k} w_i \cdot \prod_{j<i} \mathbf{Pr}[v_j > \ell_i | v_k \in I_1]$. Thus, we can see that a decision tree can be specified simply by a permutation of the intervals, since the last term in the last equation is fixed. More precisely, let

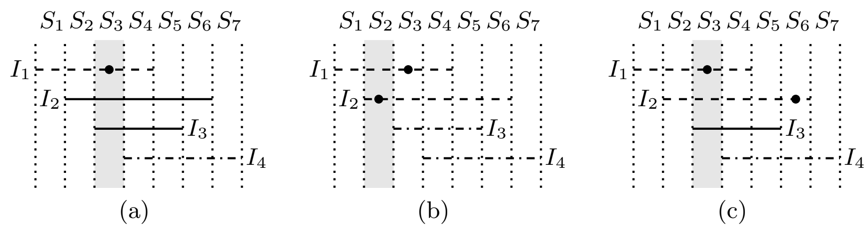


Fig. 3. A simulation of the querying process when we decide to query first I_1. (a) If $v_1 \in S_3$, I_2 must be queried, but not I_4. (b) If $v_2 \in S_2$, then v_2 is the minimum. (c) If $v_2 \in S_6$, then we still have to query I_3.

$a(1), \ldots, a(n)$ be a permutation of the intervals, where $a(k) = i$ means that I_i is the k-th interval in the permutation. We have two types of subtrees. Given a subset $X_k = \{a(k), \ldots, a(n)\}$ that contains 1, let $\hat{T}_k$ be the tree obtained by first querying I_1, then querying the next leftmost interval in X_k if it contains v_1 and so on. The second type of subtree T_k is defined by a suffix $a(k), \ldots, a(n)$ of the permutation. If $a(k) \neq 1$, then T_k is a decision tree with a root querying $I_{a(k)}$ and two branches. One branch, with probability $\mathbf{Pr}[v_{a(k)} \in I_1]$, consists of $\hat{T}_{k+1}$; the other branch, with probability $\mathbf{Pr}[v_{a(k)} \notin I_1]$, consists of T_{k+1}. If $a(k) = 1$, then $T_k = \hat{T}_k$, unless $k = n$, in which case T_n will be empty: I_1 does not need to be queried, because all other intervals have already been queried and their values fall to the right of I_1. We have that $\mathrm{cost}(T_k) =$

$$\begin{cases} 0, & \text{if } a(k) = a(n) = 1 \\ \mathrm{cost}(\hat{T}_k), & \text{if } a(k) = 1 \text{ but } k \neq n; \text{ otherwise,} \\ \mathbf{Pr}[v_{a(k)} \in I_1] \cdot \mathrm{cost}(\hat{T}_k | v_{a(k)} \in I_1) + \mathbf{Pr}[v_{a(k)} \notin I_1] \cdot (w_{a(k)} + \mathrm{cost}(T_{k+1})). \end{cases}$$

Note that in the last case we need to condition $\mathrm{cost}(\hat{T}_k)$ to the fact that $v_{a(k)} \in I_1$. The cost of $\hat{T}_k$ conditioned to E is $\sum_{i=k}^{n} w_i \cdot \prod_{\substack{j \geq k \\ a(j) < a(i)}} \mathbf{Pr}[v_{a(j)} > \ell_{a(i)} | E]$.

It holds that, if I_1 is not the last interval in a decision tree permutation, then it is always better to move I_1 one step to the beginning of the permutation. (We omit the proof due to space limitations.) Thus, by induction, the optimum solution either first queries I_1, or has I_1 at the end of the permutation. If I_1 is the last interval to be queried, then it does not have to be queried if all other values fall to its right. Thus it may be that, in expectation, having I_1 as the last interval is optimal.

We do not know, however, how to efficiently find the best permutation ending in I_1. Simply considering which interval begins or ends first, or ordering by $\mathbf{Pr}[v_i \in I_1]$ is not enough. To see this, consider the following two instances with uniform costs and uniform probabilities. In the first, $I_1 = (0, 100)$, $I_2 = (5, 305)$ and $I_3 = (6, 220)$; the best permutation is I_2, I_3, I_1 and has cost 2.594689. If we just extend I_2 a bit to the right, making $I_2 = (5, 405)$, then the best permutation is I_3, I_2, I_1, whose cost is 2.550467.

If there was a way to determine the relative order in the best permutation between two intervals $I_j, I_k \neq I_1$, simply by comparing some value not depending on the order of the remaining intervals (for example, by comparing the cost of $I_j I_k I_1 \cdots$ and $I_k I_j I_1 \cdots$), then we could find the best permutation easily. Unfortunately, the ordering of the permutations is not always consistent, i.e., given a permutation, consider what happens if we swap I_j and I_k: it is not always best to have I_j before I_k, or I_k before I_j. Consider intervals $I_1 = (0, 1000)$, $I_2 = (3, 94439)$, $I_3 = (8, 6924)$, and $I_4 = (9, 2493)$, with uniform query cost and uniform probability distributions. The best permutation is I_4, I_3, I_2, I_1, and the cost of the permutations ending in I_1 are as follows. Note that it is sometimes better that I_2 comes before I_3, and sometimes the opposite.

$$\begin{array}{lll} \text{cost}(4,2,3,1) = 3.48611 & \text{cost}(2,4,3,1) = 3.48715 & \text{cost}(2,3,4,1) = 3.48889 \\ \text{cost}(4,3,2,1) = 3.48593 & \text{cost}(3,4,2,1) = 3.48770 & \text{cost}(3,2,4,1) = 3.48859 \end{array}$$

This issue also seems to preclude greedy and dynamic programming algorithms from succeeding. It seems that it is not possible to find an optimal substructure, since the ordering is not always consistent among subproblems and the whole problem. We have implemented various heuristics and performed experiments on random instances, and could always find instances in which the optimum was missed, even for uniform query costs and uniform probabilities.

Another reason to expect hardness is that the following similar problem is NP-hard [7]. Given stochastic uncertainty intervals $I_1, \ldots, I_n$, costs $w_1, \ldots, w_n$, and a query budget C, find a set $S \subseteq \{1, \ldots, n\}$ with $w(S) \leq C$ that minimizes $\mathbf{E}[\min_{i \in S} v_i]$.

To conclude, we note that there are good approximation algorithms, which have been proposed for the online version [13]. If query costs are uniform, then first querying I_1 costs at most opt $+ 1$, which yields a factor $1 + 1/d_1$, where d_1 is the degree of I_1 in the interval graph. For arbitrary costs, there is a randomized 1.5-approximation algorithm using weighted probabilities in the two solutions stated in Lemma 2. Those results apply to the stochastic version of the problem simply by linearity of expectation.

Theorem 2. *The minimum problem admits a* $(1+1/d_1)$*-approximation for uniform query costs, and a randomized* 1.5*-approximation for arbitrary costs.*

Those results have matching lower bounds for the online setting, and for arbitrary query costs there is a deterministic lower bound of 2. We show that the stochastic assumption can be used to beat those lower bounds for arbitrary costs. First, the randomized 1.5-approximation algorithm can be derandomized, simply by choosing which solution has smaller expected query cost: either first querying I_1, or first querying all other intervals and if necessary querying I_1. We know how to calculate both expected query costs; the latter is $\sum_{i>1} w_i + w_1 \cdot \left(1 - \prod_{i>1} \mathbf{Pr}[v_i > r_1]\right)$. We omit the proof of the following theorem and the description of the randomized algorithm.

Theorem 3. *There is a deterministic* 1.5*-approximation algorithm and a randomized* 1.45*-approximation algorithm for arbitrary query costs.*

4 Further Questions

Can we extend our approach for sorting as to handle a dynamic setting, as in [2]? E.g., where some intervals can be inserted/deleted from the initial set. Updating the dynamic program should be faster than building it again from scratch.

Is the minimum problem NP-hard or can it be solved in polynomial time? If it is NP-hard, then so are the median and the minimum spanning tree problems. Can we devise polynomial time algorithms with better approximation guarantees than the best respective competitive online results?

References

1. Bruce, R., Hoffmann, M., Krizanc, D., Raman, R.: Efficient update strategies for geometric computing with uncertainty. Theory Comput. Syst. **38**(4), 411–423 (2005)
2. Busto, D., Evans, W., Kirkpatrick, D.: Minimizing interference potential among moving entities. In: SODA. pp. 2400–2418 (2019)
3. Dürr, C., Erlebach, T., Megow, N., Meißner, J.: Scheduling with explorable uncertainty. In: ITCS, LIPIcs, vol. 94, pp. 30:1–30:14 (2018)
4. Erlebach, T., Hoffmann, M.: Query-competitive algorithms for computing with uncertainty. Bull. EATCS **116**, 22–39 (2015)
5. Erlebach, T., Hoffmann, M., Krizanc, D., Mihal'ák, M., Raman, R.: Computing minimum spanning trees with uncertainty. In: STACS. pp. 277–288 (2008)
6. Feder, T., Motwani, R., O'Callaghan, L., Olston, C., Panigrahy, R.: Computing shortest paths with uncertainty. J. Algorithms **62**(1), 1–18 (2007)
7. Goel, A., Guha, S., Munagala, K.: Asking the right questions: model-driven optimization using probes. In: PODS. pp. 203–212 (2006)
8. Goerigk, M., Gupta, M., Ide, J., Schöbel, A., Sen, S.: The robust knapsack problem with queries. Comput. Oper. Res. **55**, 12–22 (2015)
9. Gupta, A., Nagarajan, V.: A stochastic probing problem with applications. In: IPCO, LNCS, vol. 7801, pp. 205–216 (2013)
10. Gupta, A., Nagarajan, V., Singla, S.: Algorithms and adaptivity gaps for stochastic probing. In: SODA. pp. 1731–1747 (2016)
11. Gupta, M., Sabharwal, Y., Sen, S.: The update complexity of selection and related problems. Theory Comput. Syst. **59**(1), 112–132 (2016)
12. Halldórsson, M.M., de Lima, M.S.: Query-competitive sorting with uncertainty. In: MFCS, LIPIcs, vol. 138, pp. 7:1–7:15 (2019)
13. Kahan, S.: A model for data in motion. In: STOC. pp. 265–277 (1991)
14. Megow, N., Meißner, J., Skutella, M.: Randomization helps computing a minimum spanning tree under uncertainty. SIAM J. Comput. **46**(4), 1217–1240 (2017)
15. Olston, C., Widom, J.: Offering a precision-performance tradeoff for aggregation queries over replicated data. In: VLBD. pp. 144–155 (2000)
16. Ryzhov, I.O., Powell, W.B.: Information collection for linear programs with uncertain objective coefficients. SIAM J. Optim. **22**(4), 1344–1368 (2012)
17. Singla, S.: The price of information in combinatorial optimization. In: SODA. pp. 2523–2532 (2018)
18. Welz, W.A.: Robot Tour Planning with High Determination Costs. Ph.D. thesis, Technischen Universität Berlin (2014)
19. Yamaguchi, Y., Maehara, T.: Stochastic packing integer programs with few queries. In: SODA. pp. 293–310 (2018)

A Calculating the Probability of a Cascading Area

Let us discuss how to calculate $p(y, z, i, x, z', y')$, which is the probability that, given that $v_i \in S_x$, each $k \in \mathcal{I}_x(y, z)$ has $v_k \in (a_{z'}, b_{y'})$, some $j \in \mathcal{I}_x(y, z)$ has $v_j \in S_{z'}$, and some $j' \in \mathcal{I}_x(y, z)$ has $v_{j'} \in S_{y'}$. We will assume that $z', y' \neq x$, and the other cases can be computed similarly.

Let us fix the values y, z, x, and denote $\mathcal{I} = \mathcal{I}_x(y, z)$, $\mathcal{I}_i = \mathcal{I}_x(y, z) \setminus \{i\}$, and $p(z', y', i) = p(y, z, i, x, z', y')$. Let us further denote by $q(z', y')$ the probability that each $k \in \mathcal{I}$ has $v_k \in (a_{z'}, b_{y'})$. Note that

$$q(z', y') = \prod_{j \in \mathcal{I}} \mathbf{Pr}[v_j \in (a_{z'}, b_{y'})],$$

and $q(z', y')$ can be computed in linear time. Let

$$q(z', y', i) = \frac{q(z', y')}{\mathbf{Pr}[v_i \in (a_{z'}, b_{y'})]}$$

denote the similar probability defined for the set of intervals $\mathcal{I}_i$, instead of $\mathcal{I}$.

Let $P(z', y', i)$ the event corresponding to $p(z', y', i)$, and $Q(z', y', i)$ be the event corresponding to $q(z', y', i)$. Then,

$$P(z', y', i) = Q(z', y', i) \setminus (Q(z'+1, y', i) \cup Q(z', y'-1, i)).$$

Since $Q(z'+1, y', i) \cup Q(z', y'-1, i) \subseteq Q(z', y', i)$,

$$\mathbf{Pr}[P(z', y', i)] = \mathbf{Pr}[Q(z', y', i)] - \mathbf{Pr}[Q(z'+1, y', i) \cup Q(z', y'-1, i)].$$

Furthermore,

$$\begin{aligned}\mathbf{Pr}[Q(z'+1, y', i) \cup Q(z', y'-1, i)] = \mathbf{Pr}[Q(z'+1, y', i)] + \mathbf{Pr}[Q(z', y'-1, i)] \\ -\mathbf{Pr}[Q(z'+1, y', i) \cap Q(z', y'-1, i)];\end{aligned}$$

the last term is clearly $\mathbf{Pr}[Q(z'+1, y'-1, i)]$. Thus, we have that

$$\begin{aligned} p(z', y', i) &= q(z', y', i) - q(z'+1, y', i) - q(z', y'-1, i) + q(z'+1, y'-1, i) \\ &= \frac{q(z', y')}{\mathbf{Pr}[v_i \in (a_{z'}, b_{y'})]} - \frac{q(z'+1, y')}{\mathbf{Pr}[v_i \in (a_{z'+1}, b_{y'})]} \\ &\quad - \frac{q(z', y'-1)}{\mathbf{Pr}[v_i \in (a_{z'}, b_{y'-1})]} + \frac{q(z'+1, y'-1)}{\mathbf{Pr}[v_i \in (a_{z'+1}, b_{y'-1})]}. \end{aligned}$$

Since $q(z', y')$ does not depend on i, for each fixed x, y, z, it can be computed in linear time, so all $q(z', y')$ can be computed for all x, y, z in time $\mathrm{O}(n^6)$. Given this precomputation, all values $p(z', y', i)$ for all y, z, i, x, z', y' can be pre-computed in $\mathrm{O}(n^6)$ time.

We can further improve the runtime to $\mathrm{O}(n^5)$ as follows. Here, we fix z', y' and y, z, and compute $q(z', y')$ for various values of x. Therefore let us now denote $q(x) = q(z', y')$. Recall that $q(x) = \prod_{j \in \mathcal{I}_x(y,z)} \mathbf{Pr}[v_j \in (a_{z'}, b_{y'})]$. We compute $q(x)$ sequentially from $x = y$ to $x = z$. Given $q(x)$ for some x, $q(x+1)$ is computed by removing from the product intervals in $\mathcal{I}_x(y,z) \setminus \mathcal{I}_{x+1}(y,z)$ and including intervals in $\mathcal{I}_{x+1}(y,z) \setminus \mathcal{I}_x(y,z)$. The remainder of the product is reused. Observe that during this computation, each interval between y, z is included in (and removed from) the product exactly once. Therefore, the computation corresponding to fixed values of y, z, a, b can be done in time proportional to the number of positions x between y and z plus the number of intervals between y and z, that is, in $\mathrm{O}(n)$ time. Thus, the computation of the whole table takes $\mathrm{O}(n^5)$ time.

Finally, the preprocessing time can be further reduced (still having constant time computation of $p(\dots)$ after preprocessing), by decomposing the table for q into two parts, based on the following observation. The first table is indexed by y, x, z', y', while the second one by z, x, z', y'. Consider the value $q(y, z, x, z', y')$. This is a product of probabilities ranging over intervals $\mathcal{I}_x(y,z)$. The observation is that $\mathcal{I}_x(y,z) = \mathcal{I}_x(y,t) \setminus \mathcal{J}_x(z,t)$, where $\mathcal{J}_x(z,t)$ is the set of intervals that intersect both x and z and do not end at z. Thus,

$$q(y, z, x, z', y') = \frac{q(y, t, x, z', y')}{q'(z, t, x, z', y')}$$

if the denominator is non-zero, and otherwise $q(y, z, x, z', y') = q(y, t, x, z', y')$, where q' is defined similarly as q, except it ranges over $\mathcal{J}_x(z,t)$. The sub-tables q and q' can be computed in $\mathrm{O}(n^4)$ time using the observations in the previous paragraph.

B Proof of Theorem 1

We prove that, for any $1 \leq y \leq z \leq t$, the value $M[y, z]$ is the expected query cost of a best decision tree for the subproblem defined by the intervals totally contained in (a_y, b_z). The proof is by induction on the number of intervals contained in (a_y, b_z). If it contains less than two intervals, then no query has to be done to solve this subproblem and the claim follows, so let us assume it contains at least two intervals.

Let us define more precisely how the decision tree for a subproblem is structured. Let $\mathcal{I}$ be a collection of intervals and queried points, with at least two dependent elements, and let $T(\mathcal{I})$ be a best decision tree for solving the subproblem defined by $\mathcal{I}$. The root of the tree indicates which interval I_i to query first. Then, for each region S_x contained in I_i, the tree has a branch which is the decision tree for the remaining intervals, conditioned to the fact that $v_i \in S_x$; we can write this subtree as $T((\mathcal{I} \setminus I_i) \cup \{v_x\})$, for some $v_x \in S_x$. Note that, for any $v_x \in S_x$, the cost of $T((\mathcal{I} \setminus I_i) \cup \{v_x\})$ is the same, since v_x will be dependent to the same intervals, and the dependencies between other intervals do not change. The expected cost of the solution encoded by $T(\mathcal{I})$ is then

$$\mathrm{cost}(T(\mathcal{I})) = w_i + \sum_{S_x \subseteq I_i} \mathbf{Pr}[v_i \in S_x] \cdot \mathrm{cost}(T((\mathcal{I} \setminus I_i) \cup \{v_x\})).$$

The leaves of the tree will correspond to collections of independent intervals, which will have cost zero.

If a subtree $T(\mathcal{I})$ contains a queried value v_x and a non-queried interval I_j with $v_x \in I_j$, then Prop. 1 says that any solution for $\mathcal{I}$ must query I_j. This implies that I_j is queried in the path between the root and any leaf of the tree. Thus, it is easy to see that there is a solution with same the cost for this subproblem in which the first query is I_j. If more than one interval contains a queried value v_x, then we can query them before other intervals, and in any order, so we can actually query all of them at the same time, and have a root with branches for each combination of regions in which the values fall.

The algorithm starts by querying an interval I_i and, depending on the region S_x in which v_i falls, queries all intervals that contain S_x. Since we have a proper interval graph, the remaining intervals are divided into two independent suproblems. Also, if the minimal area containing the regions in which the values fall is the same, then the cost of the subtree is the same, since the same intervals will contain a point queried at this time; this implies that each cascading area is a single disjoint event. Given a cascading area $(a_{z'}, b_{y'})$, the remaining problem consists of finding the best solution for two subproblems: one considering that the intervals to the left of S_x have not been queried and that the leftmost queried point is in $S_{z'}$, and another that the intervals to the right of S_x have not been queried and that the rightmost queried point is in $S_{y'}$. This is precisely the definition of tables L and R; thus, if the definition of tables L and R is correct, then the theorem follows by an optimal substructure argument.

So let us prove that the definition of table $L[y, z', j]$ is correct; the proof for table R is analogous. Let us recall the definition: $L[y, z', j]$ is the minimum expected cost of the subinstance of (a_y, b_z) (for some $z \geq z'$) consisting of intervals $I_{j'}, I_{j'+1}, \ldots, I_j$, where $I_{j'}$ is the leftmost interval contained in (a_y, b_z), assuming that the leftmost queried point is contained in $S_{z'}$. If $j \leq 1$, then $(a_y, b_{z'-1})$ contains no interval and therefore $M[y, z'-1]$ is zero. If $\ell_{j-1} < a_y$, then no interval to the left of I_j is contained in (a_y, ∞), so $(a_y, b_{z'-1})$ contains no interval and $M[y, z'-1]$ is zero. If $j > 1$ and $\ell_{j-1} \geq a_y$, but $I_{j-1} \not\supseteq S_{z'}$, then all intervals in $I_j, I_{j+1}, \ldots$ have a value to the right of $a_{z'}$, and thus any solution to the intervals totally in contained $(a_y, b_{z'-1})$ is feasible to complement the current decision tree. Thus, by an optimal substructure argument, $L[y, z', j] = M[y, z'-1]$. If $j > 1$, $\ell_{j-1} \geq a_y$, and I_{j-1} contains $S_{z'}$, then there is some queried point in $S_{z'}$, so Prop. 1 implies that I_{j-1} must be queried in any solution of the subproblem, and thus can be the first interval queried in this subproblem. When querying I_{j-1}, we ensure that the leftmost region with a queried point is updated correctly, so the last equation in the definition of $L[y, z', j]$ is correct by an optimal substructure argument. $\square$

C Proof of Lemma 2

First let us consider the case when v_1 is the minimum. If $v_j < r_1$ for some $j > 1$, then I_1 has to be queried even if all other intervals have already been queried, due to Remark 1. Thus, the only situation in which I_1 may not be queried is when $v_j > r_1$ for all $j > 1$, and in this case clearly we have to query all other intervals, since otherwise we cannot decide who is the minimum.

Now we prove that, if v_i is the minimum with $i \neq 1$, then I_i must be queried. Since v_i is the minimum, all other values fall to the right of ℓ_i. In particular, $v_1 \in I_i$, since I_1 has minimum r_1. Thus, even if all other intervals have already been queried, I_i must be queried due to Remark 1.

It remains to prove that, for any $i \geq 1$, if v_i is minimum and I_i is queried, then all intervals with $\ell_j < v_i$ must also be queried; we actually prove that $I_1, \ldots, I_j$ must be queried, by induction on j. The base case is $j = 1$, and if $i \neq 1$ the claim follows from Remark 1, since $v_i < r_1$ and $\ell_1 \leq \ell_i$. So assume $j > 1$; note that $\ell_{j-1} < \ell_j < v_i$ thus, by induction hypothesis, $I_1, \ldots, I_{j-1}$ must be queried. Since v_i is the minimum, $v_k \geq v_i > \ell_j$, for $k = 1, \ldots, j-1$. Therefore, after $I_1, \ldots, I_{j-1}$ are queried, I_j is the leftmost interval, and must be queried due to Remark 1, since it contains v_i. □

D "I_1 Should Be First or Last" Lemma

Lemma 3. *Given a decision tree permutation $I_k I_1 I_{k'} \cdots$ of a subset S, with $|S| \geq 3$ and $I_1 \in S$, it costs at least as much as the cost of $I_1 I_k I_{k'} \cdots$.*

Proof. Let $\mathbf{1}[A]$ be the indicator variable of the event A, i.e., $\mathbf{1}[A] = 1$ if A is true, and zero otherwise. The cost of $I_k I_1 I_{k'} \cdots$ is

$$\begin{aligned}
& w_k + \mathbf{Pr}[v_k \notin I_1] \cdot \sum_{i \in S \setminus I_k} w_i \cdot \prod_{j<i, j \neq k} \mathbf{Pr}[v_j > \ell_i] \\
& \quad + \mathbf{Pr}[v_k \in I_1] \cdot \sum_{i \in S \setminus I_k} w_i \cdot \prod_{j<i} \mathbf{Pr}[v_j > \ell_i | v_k \in I_1] \\
&= w_k + \mathbf{Pr}[v_k \notin I_1] \cdot \sum_{i \in S \setminus I_k} w_i \cdot \prod_{j<i, j \neq k} \mathbf{Pr}[v_j > \ell_i] \\
& \quad + \mathbf{Pr}[v_k \in I_1] \cdot \sum_{i \in S \setminus I_k} w_i \cdot \max(1 - \mathbf{1}[k < i], \mathbf{Pr}[v_k > \ell_i | v_k \in I_1]) \cdot \prod_{j<i, j \neq k} \mathbf{Pr}[v_j > \ell_i] \\
&= w_k + \mathbf{Pr}[v_k \notin I_1] \cdot \sum_{i \in S \setminus I_k} w_i \cdot \max(1 - \mathbf{1}[k < i], \mathbf{Pr}[v_k > \ell_i | v_k \notin I_1]) \cdot \prod_{j<i, j \neq k} \mathbf{Pr}[v_j > \ell_i] \\
& \quad + \mathbf{Pr}[v_k \in I_1] \cdot \sum_{i \in S \setminus I_k} w_i \cdot \max(1 - \mathbf{1}[k < i], \mathbf{Pr}[v_k > \ell_i | v_k \in I_1]) \cdot \prod_{j<i, j \neq k} \mathbf{Pr}[v_j > \ell_i] \\
&= w_k + (\mathbf{Pr}[v_k \notin I_1] \cdot \max(1 - \mathbf{1}[k < i], \mathbf{Pr}[v_k > \ell_i | v_k \notin I_1]) \\
& \quad + \mathbf{Pr}[v_k \in I_1] \cdot \max(1 - \mathbf{1}[k < i], \mathbf{Pr}[v_k > \ell_i | v_k \in I_1])) \cdot \sum_{i \in S \setminus I_k} w_i \cdot \prod_{j<i, j \neq k} \mathbf{Pr}[v_j > \ell_i] \\
&= w_k + \max(1 - \mathbf{1}[k < i], \mathbf{Pr}[v_k > \ell_i]) \cdot \sum_{i \in S \setminus I_k} w_i \cdot \prod_{j<i, j \neq k} \mathbf{Pr}[v_j > \ell_i] \\
&= w_k + \sum_{i \in S \setminus I_k} w_i \cdot \prod_{j<i} \mathbf{Pr}[v_j > \ell_i],
\end{aligned}$$

where the first equality holds since the probability distribution is independent for each interval, so $\mathbf{Pr}[v_j > \ell_i | v_k \in I_1] = \mathbf{Pr}[v_j > \ell_i]$ unless $j = k$. The second equality holds since $\ell_i < r_1$ for all i, so $\mathbf{Pr}[v_k > \ell_1 | v_k \notin I_1] = 1$.

On the other hand, if we swap I_k and I_1, then the cost is

$$\begin{aligned}\sum_{i\in S} w_i \cdot \prod_{j<i} \mathbf{Pr}[v_j > \ell_i] &= w_k \cdot \prod_{j<k} \mathbf{Pr}[v_j > \ell_k] + \sum_{i\in S\setminus I_k} w_i \cdot \prod_{j<i} \mathbf{Pr}[v_j > \ell_i] \\ &\le w_k + \sum_{i\in S\setminus I_k} w_i \cdot \prod_{j<i} \mathbf{Pr}[v_j > \ell_i],\end{aligned}$$

since $\prod_{j<k} \mathbf{Pr}[v_j > \ell_k] \le 1$, and the last value is precisely the cost of the former permutation. ◻

E Better Approximations for the Minimum Problem

E.1 Proof of Theorem 3

Let $\mathcal{V}$ be the set of realizations of the values, and assume $\mathcal{V}$ is finite. (Otherwise finiteness can be attained by grouping realizations into equivalence classes based on the partition into regions.) For each $V \in \mathcal{V}$, let $C_1(V)$ be the cost of first querying I_1, then querying I_2 if $v_1 \in I_2$, and so on, and let $C_R = \sum_{i>1} w_i$. We partition $\mathcal{V}$ in sets $\mathcal{V}_1$ and $\mathcal{V}_R$, where $V \in \mathcal{V}_1$ if $\text{opt}(V) = C_1(V)$, and $V \in \mathcal{V}_R$ if $\text{opt}(V) = C_R$; note that Lemma 2 guarantees that this is indeed a partition.

Let $\rho = \mathbf{Pr}[V \in \mathcal{V}_R]$ and $\mathcal{C}_1 = \sum_{V'\in\mathcal{V}_1} C_1(V') \cdot \mathbf{Pr}[V = V']$. If opt^* is the expected query cost of the best permutation for the stochastic problem, then

$$\begin{aligned}\text{opt}^* &\ge \sum_{V'\in\mathcal{V}} \text{opt}(V') \cdot \mathbf{Pr}[V = V'] \\ &= \sum_{V'\in\mathcal{V}_1} C_1(V') \cdot \mathbf{Pr}[V = V'] + \sum_{V'\in\mathcal{V}_R} C_R \cdot \mathbf{Pr}[V = V'] = \mathcal{C}_1 + \rho \cdot C_R,\end{aligned}$$

where the inequality holds by bounding opt^* via a fractional solution.

Now consider the solution for the stochastic problem obtained by the algorithm. Let ALG_1 be the cost of first querying I_1, and let ALG_R be the cost of first querying all other intervals. Then

$$\text{ALG}_1 \le \mathcal{C}_1 + \rho \cdot (w_1 + C_R) \le \mathcal{C}_1 + \rho \cdot C_R + \rho \cdot \frac{\mathcal{C}_1}{1-\rho} \le \text{opt}^* + \frac{\rho}{1-\rho} \cdot \mathcal{C}_1,$$

where the second inequality holds because $\mathbf{Pr}[V \in \mathcal{V}_1] = 1 - \rho$ and $C_1(V') \ge w_1$ for any V'. On the other hand,

$$\text{ALG}_R \le (1-\rho) \cdot (C_R + w_1) + \rho \cdot C_R \le C_R + \mathcal{C}_1 \le \text{opt}^* + (1-\rho) \cdot C_R.$$

The solution returned by the algorithm costs

$$\begin{aligned}\min(\mathrm{ALG}_1, \mathrm{ALG}_R) &\le \mathrm{opt}^* + \min\left(\frac{\rho}{1-\rho}\cdot \mathcal{C}_1, (1-\rho)\cdot C_R\right)\\ \le \mathrm{opt}^* + \sqrt{\frac{\rho}{1-\rho}\cdot \mathcal{C}_1 \cdot (1-\rho)\cdot C_R} &= \mathrm{opt}^* + \sqrt{\mathcal{C}_1\cdot\rho\cdot C_R}\\ \le \mathrm{opt}^* + \frac{\mathcal{C}_1 + \rho\cdot C_R}{2} &\le \frac{3}{2}\cdot \mathrm{opt}^*,\end{aligned}$$

where the second and third inequalities hold by the properties of the geometric mean. □

E.2 Randomized Algorithm

Let a denote the cost of I_1 and b denote the total cost of R, where R is the set of intervals other than I_1.

Let p denote the probability that R *hits* I_1, that is, some interval in R has its value v in I_1. We have different algorithms, depending on the costs of intervals.

Case 1. Each element of R has cost less than $3b/4$. We choose a subset $G \subseteq R$ such that G has cost at least $\alpha \cdot b$ and has probability at least $\alpha \cdot p$ of hitting I_1, and $\alpha \in (1/4, 3/4)$.

To that end, let G' be a subset of cost between $b/2$ and $3b/4$ (which exists by our assumption). Let βb be the total cost of G'. If G' hits I_1 w.p. at least βp then we let $G = G'$ and $\alpha = \beta$. Otherwise, the complement $R \setminus G'$ has cost $(1-\beta)b$ and probability at least $(1-\beta)p$, so we let $G = R \setminus G'$ and $\alpha = 1-\beta$. Note that $\beta \in (1/2, 3/4)$ and $1-\beta \in (1/4, 1/2)$, so $\alpha \in (1/4, 3/4)$.

The algorithm is as follows. Let $x \in (0,1)$ be a probability parameter. Below, "query X", for a subset X of intervals, means querying all intervals in X.

1. with probability x, query I_1 and cascade
2. with the remaining probability $1-x$, do:
   (a) query G
   (b) if it hits I_1 then query I_1 and cascade
   (c) otherwise, query R and if it hits I_1 then query I_1 too.

Analysis. If R hits I_1 then the optimal strategy is to query I_1 and cascade, as discussed in Sec. 3. Conditioned on this event, the expected approximation we get is

$$\begin{aligned}x\cdot 1 + (1-x)\cdot\left[\alpha\cdot\frac{a+\alpha b}{a} + (1-\alpha)\cdot\frac{a+b}{a}\right] &= 1 + (1-x)\cdot(\alpha^2+1-\alpha)b/a\\ &\le 1 + (1-x)\cdot 13b/(16a),\end{aligned}$$

where the first occurrence of α corresponds to the probability that G hits I_1, conditioned on the event that R hits I_1, and the last inequality follows from the fact that $\alpha \in (1/4, 3/4)$.

On the other hand, if R does not hit I_1 then the optimum is to query R only. The expected approximation is then

$$x \cdot \frac{a+b}{b} + (1-x) \cdot 1 = 1 + x \cdot a/b,$$

Thus, it remains to choose x so as to minimize the maximum of $1 + xa/b$ and $1 + (1-x) \cdot 13b/(16a)$. We do this by equating the two expressions: we obtain $x = 13b^2/(13b^2 + 16a^2)$, giving us the expression $1 + \frac{z}{1+16z^2/13}$ with $z = a/b$. The latter is maximized at $z = \sqrt{13/16} \approx 0.9$ and the maximum is $1 + \sqrt{13/64} < 1.4507$.

Case 2. R contains an element J of cost at least $3b/4$. Let $G' = R \setminus J$. Then we postpone querying G', optimally solve the two-interval instance $\{I_1, J\}$, and query/cascade G' only when necessary. First, observe that the cost of solving the reduced instance is at most the cost of the original instance. Next, note that we query G' either when I_1 is queried and it hits G', in which case we perform optimally, or if we query J and it does not hit I_1, in which case querying G' increases the cost we paid by only $1/3$ fraction. Thus, in this case we have an expected $4/3$ approximation.

The approach above can be used with fine-tuned parameters to obtain 1.444 approximation, but it cannot give better than 1.433 approximation, which is the best one can obtain in Case 1, even if not restricted by Case 2.