Result-Sensitive Binary Search with Noisy Information

Narthana S. Epa
School of Computing and Information Systems, The University of Melbourne, Victoria, Australia
nepa@student.unimelb.edu.au

Junhao Gan
School of Computing and Information Systems, The University of Melbourne, Victoria, Australia
junhao.gan@unimelb.edu.au

Anthony Wirth
School of Computing and Information Systems, The University of Melbourne, Victoria, Australia
awirth@unimelb.edu.au

Abstract

We describe new algorithms for the *predecessor* problem in the *Noisy Comparison* Model. In this problem, given a sorted list L of n (distinct) elements and a query q, we seek the *predecessor* of q in L: denoted by u, the largest element less than or equal to q. In the Noisy Comparison Model, the result of a comparison between two elements is non-deterministic. Moreover, multiple comparisons of the same pair of elements might have different results: each is generated *independently*, and is correct with probability $p > 1/2$. Given an overall error tolerance Q, the cost of an algorithm is measured by the total number of noisy comparisons; these must guarantee the predecessor is returned with probability at least $1 - Q$. Feige et al. showed that predecessor queries can be answered by a modified binary search with $\Theta(\log \frac{n}{Q})$ noisy comparisons.

We design result-sensitive algorithms for answering predecessor queries. The query cost is related to the index, k, of the predecessor u in L. Our first algorithm answers predecessor queries with $O(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q})$ noisy comparisons, for an arbitrarily large constant c. The function $\log^{*(c)} n$ iterates c times the iterated-logarithm function, $\log^* n$. Our second algorithm is a genuinely result-sensitive algorithm whose *expected* query cost is bounded by $O(\log \frac{k}{Q})$, and is guaranteed to terminate after at most $O(\log \frac{\log n}{Q})$ noisy comparisons.

Our results strictly improve the state-of-the-art bounds when $k \in \omega(1) \cap o(n^{\varepsilon})$, where $\varepsilon > 0$ is some constant. Moreover, we show that our result-sensitive algorithms immediately improve not only predecessor-query algorithms, but also *binary-search-like* algorithms for solving key applications.

2012 ACM Subject Classification Theory of computation → Sorting and searching; Theory of computation → Predecessor queries

Keywords and phrases Fault-tolerant search, random walks, noisy comparisons, predecessor queries

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.60

Funding *Anthony Wirth*: Funded by the Melbourne School of Engineering.

Acknowledgements We thank our anonymous reviewer for directing us to the work of Karp and Kleinberg [8].

1 Introduction

Let U be a totally ordered universe of elements. Consider a sorted (abstract) list L of n (distinct) elements from U, in ascending order, and indexed starting from 1. Denote by $L[j]$ the jth element in L for $j = 1, 2, \dots, n$. Given a query element q, the *predecessor* problem on L is to return the index $k = \max\{j \mid L[j] \le q\}$; if the set $\{j \mid L[j] \le q\}$ is empty, return 0. As the 0 case can be identified easily, without loss of generality, we assume that the predecessor always exists in L. The predecessor query is one of the most fundamental and important

© Narthana S. Epa, Junhao Gan, and Anthony Wirth;
licensed under Creative Commons License CC-BY

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

primitives in computer science; it is a crucial building block for a large number of data structures and algorithms. Any improvements (even a small constant) in the efficiency of answering predecessor queries would immediately benefit all these algorithms, improving practicality. In fact, the predecessor query has been widely studied in various computation models. For example, it is well known that a predecessor query can be answered with $\Theta(\log n)$ comparisons by *binary search* in the (usual, non-noisy) Comparison Model.

In this paper, we focus on the *Noisy Comparison Model*, which was proposed by Feige et al. [7]. It has two parameters: (i) the *probability of a correct comparison*, $p > 1/2$, and (ii) the *overall error tolerance* $Q \in (0, 1/2]$. In this model, each comparison between a pair of elements is "noisy", in the sense that the comparison result is not deterministic. Instead, each comparison is answered *correctly* by a comparison oracle *independently* with probability p. In other words, for two different comparisons queries between the *same* pair of elements, the oracle may return different results, completely independently. The correct answer is returned with probability p, the incorrect answer with probability $1 - p$.

In this paper, we seek algorithms that minimize the number of calls to the *comparison oracle*. The *cost* of an algorithm is measured by the total *number* of noisy comparisons. We require that the algorithm solves the predecessor problem correctly with probability at least $1 - Q$.

1.1 Applications

The Noisy Comparison Model is particularly useful for modelling application scenarios where comparisons between elements are difficult and costly. For example, in a crowd-sourcing scenario, a human worker may make mistakes when comparing two given objects such as images, and each such comparison incurs some cost, e.g., a dollar. As a result, an efficient algorithm in the Noisy Comparison Model nicely trades off budget with accuracy. Another example is in employee recruitment, where the comparison result between two applicants may be noisy ("incorrect") due to lack of familiarity with or even bias in relation to certain applicants. Potentially, a noisy comparison algorithm could help guide a process to improve fairness with limited resources.

1.2 Previous Innovation

To answer a predecessor query under the Noisy Comparison Model, we must return the correct answer with probability $1 - Q$. To achieve this, a naive approach is to replace each deterministic comparison in an algorithm (in the usual model) with a sequence of repeated comparisons between the same two elements in the Noisy Comparison Model. If we want the probability of each such comparison being correct to be $1 - \delta$, each such sequence comprises $O(\log_p(1/\delta))$ repeated noisy comparisons and returns the majority answer. Moreover, to account for the $O(\log n)$ comparisons in traditional binary search, by setting $\delta = O(\frac{Q}{\log n})$, we solve the predecessor problem with probability at least $1 - Q$, making $O(\log n \cdot \log(\frac{\log n}{Q}))$ noisy comparisons.

While the traditional binary search is optimal under the (deterministic) Comparison Model, this naive adaptation of binary search is actually far from optimal under the Noisy Comparison Model. As Feige et al. show [7], the worst-case lower bound on answering a predecessor query under the Noisy Comparison Model is only $\Omega(\log(n/Q))$. Indeed, Feige et al. introduce a binary-search based algorithm whose noisy comparison cost matches this lower bound. Some of our methods build on this algorithm, which we hence refer to as Feige.

1.3 The Open Question

In general, Feige is worst-case optimal. However, when the index k of the predecessor is $O(1)$, the brute-force algorithm which compares q naively with elements of L one-by-one, in ascending order, performs better. The total number of noisy comparisons is bounded by $O(\log(1/Q))$. Hence the state-of-the-art result is $O(\log \frac{1}{Q})$ when $k = O(1)$, and $O(\log \frac{n}{Q})$ when $k = \omega(1)$. The subtle, but crucial, question is: Can we bridge these two bounds smoothly over the entire spectrum $k \in \{1, \dots, n\}$?

With this motivation, in this paper, we design *result-sensitive* algorithms for answering predecessor queries, and solving related problems, in the Noisy Comparison Model. That is, the costs of the algorithms should depend on the result index, k.

1.4 Our Contributions

We develop result-sensitive algorithms for the predecessor problem under noisy comparisons, then apply these to Range Count, Stabbing Count, and Shortlisting problems. We start with a function definition. The function $\log^{*(c)} n$ iterates the iterated-logarithmic function c times. That is, $\log^{*(1)} n = \log^* n$ and $\log^{*(c)} n = \log^*(\log^{*(c-1)} n)$.

▶ **Theorem 1.** *Let $c \geq 1$ be an arbitrarily large integer constant. There exists an algorithm that, on every predecessor query, answers correctly with probability at least $1 - Q$, and makes $O(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q})$ noisy comparisons.*

Of course, $\log^* n$ grows very slowly, e.g., $\log^* n = 6$ for $n = 2^{2^{32}}$. However, $\log^{*(c)} n$, the bound in Theorem 1, is unfortunately *not* genuinely result sensitive as it depends on n, not k. Nonetheless, the analysis of this algorithm, supporting Theorem 1, is relatively simple, building on the Feige algorithm [7]. With a more careful algorithm design and analysis, we show that:

▶ **Theorem 2.** *There exists an algorithm that, on every predecessor query, answers correctly with probability at least $1 - Q$, and makes $O(\log \frac{k}{Q})$ noisy comparisons in expectation.*

As a result, our algorithm in Theorem 2 has bridged the noisy-comparison bounds, of $O(\log \frac{1}{Q})$ for $k = O(1)$, and of $O(\log \frac{n}{Q})$ for $k = \omega(1)$, over the whole spectrum of k. Moreover, we highlight that for $k \in \omega(1) \cap o(n^\varepsilon)$, with ε being an arbitrarily small constant, our algorithm strictly improves both of the two state-of-the-art bounds; for the $O(1)$ and $\Omega(n^\varepsilon)$ ranges of k, our algorithm matches the better of the two bounds.

Our Contribution to Applications

Our result-sensitive algorithm also immediately improves two types of noisy-comparison algorithms for solving certain problems:

Type-I. algorithms that include predecessor search as a black box; and
Type-II. algorithms that generalise the comparison oracle in predecessor search

Type-I algorithms. We consider the following two example problems:

▶ **Problem 1** (Range Count Query). *Given a sorted list L of n elements from U, and a query range $(a, b]$, a* range count query *returns the number of elements in L falling into $(a, b]$.*

▶ **Problem 2** (Stabbing Count Query). *Consider a set S of n closed intervals, each of which appears in two sorted lists. One list comprises S sorted by left endpoints; the other list comprises S sorted by right endpoints. Given a query value q, a stabbing count query returns the number of intervals of S that contain q.*

In addition, by Theorem 2, we have:

▶ **Corollary 3.** *There exists an algorithm that, on every range count query, answers correctly with probability at least* $1 - Q$*, and makes* $O(\log \frac{k_a+1}{Q} + \log \frac{count+1}{Q})$ *noisy comparisons in expectation. The value* k_a *is the index of the predecessor of* a *in* L*, while count is the number of elements in* L *falling in* $(a, b]$*.*

▶ **Corollary 4.** *There exists an algorithm that, on every stabbing count query, answers correctly with probability at least* $1-Q$*, and makes* $O(\log \frac{k_r+1}{Q} + \log \frac{count+1}{Q})$ *noisy comparisons in expectation. The value* k_r *is the number of right endpoints no larger than* q*, while count is the number of intervals in* S *stabbed by* q*.*

When $k_a, k_r, count \in \omega(1) \cap o(n^\epsilon)$, the algorithms in these two corollaries strictly improve the results implied by the state-of-the-art Feige algorithm.

Type-II algorithms. We consider this problem:

▶ **Problem 3** (Shortlisting)**.** *Given two sorted lists,* A *and* B*, with* n_A *and* n_B *elements, respectively, that are disjoint, and a value* m*,* $1 \le m \le n_A + n_B$*, return the* m *smallest elements (i.e., a shortlist of size* m*) from the* conceptual *merged sorted list of* A *and* B*.*

By generalising the comparisons in a predecessor search, we prove:

▶ **Theorem 5.** *There exists an algorithm for the Shortlisting problem that, on every input, with probability at least* $1-Q$*, returns a set of the* m *smallest elements,* not necessarily sorted*, and makes* $O\left(\log \frac{\min\{k, m-k\}+1}{Q}\right)$ *noisy comparisons in expectation, where* k *is the number of elements from* A *being in the shortlist.*

2 Related Work

An early model for computing with errors was the Rényi-Ulam game, where the player must identify an element of a finite set of integers by asking questions from an adversary who may lie a bounded number of times [15, 18]. Variants have been considered where the errors were bounded, either globally or as a running total. Binary-search style algorithms to play this game under such models where put forward by Rivest et al. [16], Saks and Wigderson [17], and Borgstrom and Kosaraju [4].

The Noisy Comparison Model used in this paper is different, but related to the Rényi-Ulam game [11, 7]. In this model, errors are random, independent and transient, meaning that repeated comparisons can be used to bound the correctness probability for the result. Feige et al. [7] consider the problems of binary search, sorting, merging, and ranked selection under this model. Their algorithm for binary search is a basic result that we rely on for much of this paper. Algorithms for the problems of sorting, merging and ranked selection with related error models have been more recently considered by Ravikumar for merge sorting with bounded errors [14], Luecci and Liu for minimum selection with noisy errors [10], and Chen et al. for ranked selection with noisy errors [5]. For a more complete history and summary of the various error models for fault tolerant searching, refer to the survey paper by Pelc [12] or the book of Cicalese [6].

Karp and Kleinberg [8] pursued an alternative comparison model. There, the oracle's probability of reporting "less than" is an increasing function of the index in the list L. In that the probability p is fixed, the model presented here is a special case of that model. Unlike Karp and Kleinberg [8], however, we succeed only if we find the exact predecessor; they permit an approximately close answer. Applying the algorithm of Karp and Kleinberg [8] to our scenario would find an approximate predecessor with probability at least 3/4.

The predecessor problem may be solved with an unbounded search algorithm to achieve a result-sensitive comparison bound. It is straightforward to apply an unbounded search algorithm to solve the predecessor problem with an output-sensitive running time of $O(\log k)$, where k is the index of the predecessor. The unbounded search problem was established, for the deterministic case, by Bentley and Yao [3]. Further lower and upper bounds have been established due to the work of Raoult and Vuilleman [13], Knuth [9], and Beigel [2]. Pelc [11] also addresses the problem of unbounded search in the Noisy Comparison Model that is used in this paper. However, if we treat the tolerance, Q, as a constant, letting k be the index of the result, when the probability of an correct individual comparison, p, is in $[\frac{1}{3}, \frac{1}{2})$, these algorithms make $O(\log^2 k)$ comparisons [11]. However, in a survey article [12], Pelc notes that for the case of linearly bounded errors, Aslam and Dhagat [1] improved the comparison bound to $O(\log k)$ for all $p < \frac{1}{2}$. Furthermore, Pelc observed that this implies the existence of an unbounded search algorithm in the Noisy Comparison Model that performs $O(\log k)$ for every $p < \frac{1}{2}$. However this asymptotic result on the number of comparisons is due to varying the tolerance, Q. In our algorithms, we add a $\log(1/Q)$ term to the number of comparisons; the algorithms Pelc refers to have a $1/(1 - \sqrt{1 - Q})$ term, which grows much faster than $\log(1/Q)$ as $Q \to 0$.

3 Almost Result-Sensitive Algorithms

The predecessor *index* of a query element q is k: we denote the actual predecessor, $L[k]$, by u. Moreover, we assume that $k \neq 0$. For simplicity, we also assume that n is a tower-of-two number, namely, $n = 2^{2^{\cdot^{\cdot^{2}}}}$. Otherwise, we could pad L with dummy elements of value ∞, increasing n to be a tower-of-two. As a result, for every integer i with $1 \leq i \leq \log^* n$, $\log^{(i)} n$ is an integer, where $\log^{(1)} n = \log n$ and $\log^{(i)} n = \log(\log^{(i-1)} n)$.

In this section, we prove Theorem 1, showing an almost result-sensitive predecessor algorithm, with

$$O\left(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q}\right)$$

noisy comparisons. The basic idea is to incorporate Feige as a black box. As the first step, in Section 3.1 we describe an algorithm with a slightly worse asymptotic bound on the number of noisy comparisons.

3.1 An $O\left(\log \frac{\log^* n}{Q} + \log \frac{k}{Q}\right)$-Algorithm

Denote by $L[a, b)$ the sub-list of L of a *contiguous* index range $\{a, \ldots, b-1\}$ with $1 \leq a \leq b \leq n$. Suppose we identify a sub-list $L[a, b)$ that contains the predecessor u and whose length, $b - a$, is bounded by $O(k^{c'})$, for some constant c'. By applying Feige, we can answer the query on $L[a, b)$ with $O(\log \frac{k}{Q})$ noisy comparisons. The basic idea of the algorithm proposed in this subsection is to use binary search with noisy comparisons to identify such a sub-list $L[a, b)$ with length bounded by $O(k)$.

To ease the presentation, setting $L_0 \equiv L$, we let $L_i[j]$ denote the item in location j of list L_i, while $I_i[u]$ is the *dual*, the index of item u in list L_i. Meanwhile, $Pred_i[q]$ is the *index* of the predecessor of q in list L_i. For this purpose, we define two *conceptual* lists:

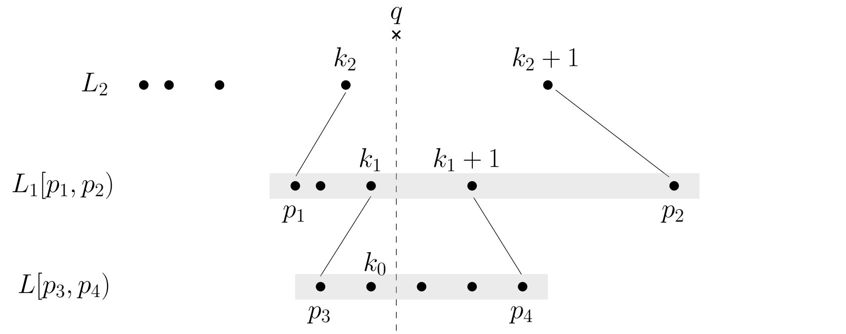

Figure 1 Our three-level binary search with noisy comparisons. Searching for q in L_2 first, we then "zoom in" to a sub-list of L_1, i.e., $L_1[p_1, p_2)$. Then, via the predecessor of q in $L_1[p_1, p_2)$, we further "zoom in" to a sub-list of L, i.e., $L[p_3, p_4)$, in which we can find the predecessor of q in L.

Power-of-two List L_1: This is the sorted sub-list of L comprising the elements at indexes that are *powers of 2*. It has $1 + \log n$ elements:

$$L[1], L[2], L[4], L[8], \ldots, L[n] = L[2^{\log n}].$$

Iterated-log List L_2: This is the sorted sub-list of L comprising the elements at the *iterated-logarithmic* indexes. It has $1 + \log^* n$ elements:

$$L[1] = L[\log^{(\log^* n)} n], L[\log^{(\log^* n-1)} n], L[\log^{(\log^* n-2)} n], \ldots,$$
$$L[\log^{(2)} n], L[\log n], L[n] = L[\log^{(0)} n].$$

Our algorithm adopts a three-level binary search with noisy comparisons, in which the search range is narrowed, by searching at finer and finer granularities. Figure 1 illustrates this principle: while L_2 has $O(\log^* n)$ elements, both the lengths of $L_1[p_1, p_2)$ and $L[p_3, p_4)$ are bounded by $O(k)$. As a result, the total number of noisy comparisons is bounded by $O(\log \frac{\log^* n}{Q} + \log \frac{k}{Q})$, as claimed. At each level, the noisy binary search has error tolerance $Q/3$; by the union bound, the overall error tolerance is Q.

The details of the algorithm are as follows. Observe that the indexes p_1, p_2, p_3, p_4, and k_2 are in some sense absolute, whereas the indexes k_0 and k_1 are relative to the sublist in which they are defined.

Feige on L_2: Let $k_2 = Pred_2[q]$. If $k_2 = 1 + \log^* n$, then return n as the answer, as the largest element of L_2 is in fact $L[n]$. Otherwise, let $u_2 = L_2[Pred_2[q]]$ and let $p'_1 = I_0[u_2]$. Then $p'_1 = \log^{(\alpha+1)} n$, where $\alpha = \log^* n - k_2$. Let $p'_2 = \log^{(\alpha)} n$. Furthermore, define $p_1 = \lceil \log p'_1 \rceil + 1$ and $p_2 = \lfloor \log p'_2 \rfloor + 1$. It can be verified that the elements in the sub-list $L_1[p_1, p_2)$ consists of all the elements in L_1 whose indices in L are in $[p'_1, p'_2)$.

Feige on $L_1[p_1, p_2)$: Denote by k_1 the (relative) index of the predecessor u_1 of q in sublist $L_1[p_1, p_2)$. Let $p_3 = I_0[u_1]$, so we have $p_3 = 2^{k_1+p_1-2}$, and also let $p_4 = 2 \cdot p_3$.

Feige on $L[p_3, p_4)$: Let k_0 be the (relative) index of the predecessor u of q in sublist $L[p_3, p_4)$. Finally, return $k = p_3 + k_0 - 1$ as the index of u in L.

Number of Noisy Comparisons

The first search, in L_2, makes $O(\log \frac{\log^* n}{Q})$ noisy comparisons, as the length of L_2 is bounded by $O(\log^* n)$. In the second search, $L_1[p_1, p_2)$ contains at most $O(\log \frac{p_2'}{p_1'})$ elements. Since $p_1' = \log p_2'$, the length of $L_1[p_1, p_2)$ is bounded by $O(\log p_2') = O(p_1') = O(k)$. Hence the second search makes $O(\log \frac{k}{Q})$ noisy comparisons. Finally, as $L[p_3, p_4)$ has length at most $p_4 - p_3 = p_3 \le k$, the number of noisy comparisons in the third search is bounded by $O(\log \frac{k}{Q})$. Therefore, the overall number of comparisons is $O(\log \frac{\log^* n}{Q} + \log \frac{k}{Q})$.

3.2 An $O\left(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q}\right)$-Algorithm

Observe that when $k \in \Omega(\log^* n)$, the comparison cost of the three-level algorithm becomes $O(\log \frac{k}{Q})$; but when $k \in o(\log^* n)$, the most expensive part, asymptotically, is Feige on L_2. To improve the $\log^* n$ term in the bound, we bootstrap our algorithm, replacing Feige on L_2 with our three-level binary search algorithm itself, setting the tolerance $Q/6$ for each run of Feige. That is, we treat L_2 as an input to a new predecessor query of q, and solve it with $O(\log \frac{\log^* |L_2|}{Q} + \log \frac{k_2}{Q})$ noisy comparisons. Combining with the search costs on sub-lists L_1 and L, the overall bound is improved to $O(\log \frac{\log^* \log^* n}{Q} + \log \frac{k}{Q})$.

In fact, we can repeat this process, bootstrapping c times. By setting the tolerance of each run of Feige to $Q/(3c)$, by the union bound, the tolerance of the overall algorithm is still at most Q. The total number of noisy comparisons is bounded by $O(\log \frac{\log^{*(c)} n}{Q} + c \cdot \log \frac{c \cdot k}{Q})$. As long as c is a constant, the bound is $O(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q})$, proving Theorem 1.

4 Genuinely Result-Sensitive Algorithms

Incorporating Feige as a black box, as we did in Section 3, admits clean analysis. Unfortunately, the bound on the number of noisy comparisons is not quite result sensitive. In this section, we introduce truly result-sensitive asymptotic behavior, by describing an algorithm comprising two phases:

Phase 1: Find $u_1 = L_1[Pred_1[q]]$, with tolerance $Q' = Q/2$.

Phase 2: Let $p_1 = I_0[u_1]$. Find the predecessor of q in the sub-list $L[p_1, 2 \cdot p_1)$ with tolerance $Q/2$. Let k_0 be the (relative) index, and return $k = p_1 + k_0 - 1$.

By the union bound, the overall tolerance of this two-phase algorithm is at most Q. Moreover, since $k \ge p_1$, the length of $L[p_1, 2 \cdot p_1)$ is at most k. Hence the cost of Feige in Phase 2 is $O(\log \frac{k}{Q})$ noisy comparisons. Therefore, in the rest of this section, we focus on designing an algorithm which completes Phase 1 with $O(\log \frac{k}{Q})$ noisy comparisons *in expectation*.

As the first step, we propose a Phase-1 method that can, with probability at least $1 - Q$, find the correct predecessor of q in L_1, with $O(\log \frac{k}{Q})$ noisy comparisons (for convenience, we write Q, rather than Q', locally). Unfortunately, there is no guarantee that this method terminates, and hence there is no worst-case bound on the number of noisy comparisons. Nonetheless, in Section 4.2 we refine the method so that it *always* halts, has error tolerance Q, and *in expectation* makes at most $O(\log \frac{k}{Q})$ noisy comparisons. For convenience, we assume $(\log n)/Q$ is an integer.

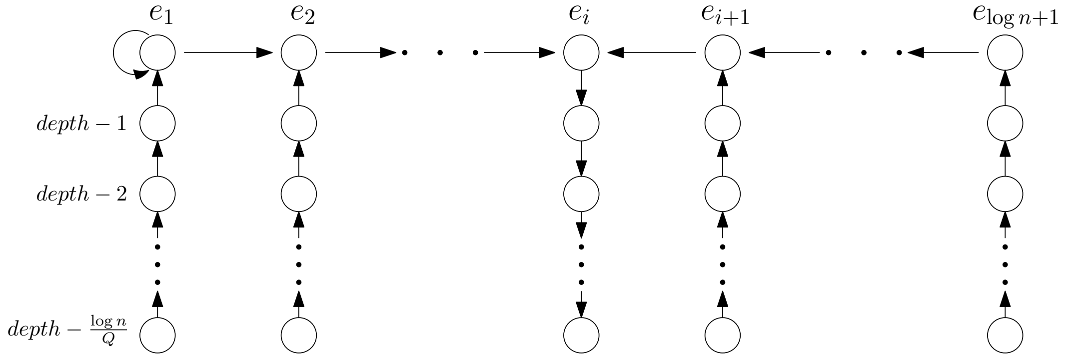

Figure 2 Comparison tree T, including directed edges displayed for oriented comparison tree T_q^o, with item $e_i = L_1[Pred_1[q]]$.

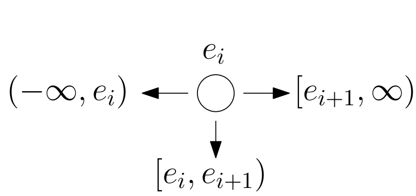

Figure 3 The horizontal node of e_i: its three edges partition the search space.

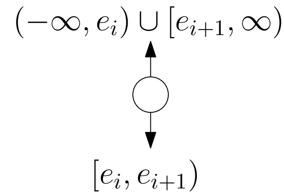

Figure 4 Non-leaf vertical node of e_i: its two edges partition the search space.

4.1 A Random-Walk Phase-1 Algorithm

The algorithm proposed in this subsection is based on random walks on a *conceptual comparison tree*, denoted by T, which is defined as follows and shown in Figure 2 (ignoring the edge directions).

- There is a *horizontal path* with $1 + \log n$ nodes in T, where the ith node, $i \in [1, \log n + 1]$, corresponds to $e_i = L_1[i]$, i.e., $L_0[2^{i-1}]$. Each node in this path is called a *horizontal node*.
- For each e_i, linked to its horizontal node, there is a *vertical extended path* with $(\log n)/Q$ nodes. The jth node of this vertical extended path ($j \in [1, \frac{\log n}{Q}]$) is at *depth-$j$*. Each node in this extended path is called a *vertical node* of e_i.
- For each e_i, its horizontal node has three edges which correspond to a partition of the search space, as shown in Figure 3. Specifically, (i) the left edge (in fact, a self loop for e_1) corresponds to $(-\infty, e_i)$; (ii) the right edge corresponds to $[e_{i+1}, \infty)$; and (iii) the downward edge corresponds to $[e_i, e_{i+1})$ (in fact, for $e_{1+\log n}$, the right and downward edges are the same). When a walk (with respect to query q) is at the horizontal node e_i, if the noisy comparisons indicate that q falls in one of the three partitions, then the walk should move along the corresponding edge.
- A vertical node at depth-$(\log n)/Q$ has only an upward edge. Every other vertical node has two edges, corresponding to a partition of the solution space, as shown in Figure 4. The upward edge corresponds to $(-\infty, e_i) \cup [e_{i+1}, \infty)$, while the downward edge corresponds to $[e_i, e_{i+1})$. As with the horizontal nodes, when a walk for a query q is at the current vertical node, it should move along the edge corresponding to the range where q falls, as indicated by the comparisons.

Consider the query q in the *Deterministic* Comparison Model. Each edge in T can be *conceptually* oriented according to the result of a deterministic comparison on query q. We call this *the oriented comparison tree* with respect to q, denoted by T_q^o, as shown in Figure 2. We emphasize that both T and T_q^o are conceptual: neither is materialized.

Analysis

Referring to Figure 2, starting at an arbitrary node s in T_q^o, the walk that follows the edge directions, reaches the deepest vertical node t, the leaf at depth-$(\log n)/Q$, of the predecessor of q in L_1. Moreover, such a walk is the shortest path (in terms of the number of moves) from s to t in T_q^o.

Our first Phase-1 method, Algorithm 4.1, performs a random walk on T_q^o, starting at the horizontal node of e_1. It comprises a sequence of moves, each of which the result of the appropriate number of repeated noisy comparisons so that with probability, 2/3, the move follows the direction of the edge in T_q^o. Such a move is called *forward*, whereas a move in the oppositve direction is *backward*. (Given $p > 1/2$, we can boost the probability of a forward move to at least 2/3 with a constant-factor blowup in noisy comparisons [7].)

Algorithm 4.1 A Random-Walk Phase-1 Algorithm.

```
    Let c be some constant to be fixed.
    r ← 1                                              ▷ r is the number of rounds
    Perform c · log_3 (1/Q) moves, starting from the horizontal node of e_1
    while true do
5:      for j ← 1; j ≤ c; j ← j + 1 do
            move
            if The walk is at a vertical node of e_i at depth-(r + log_3 (1/Q)) then
                Return I_1[e_i]
        r ← r + 1                                      ▷ Increase the round number
```

▶ **Lemma 6.** *For every r, the probability of Algorithm 4.1 returning a wrong answer in round r is at most $Q/3^r$. Overall, the algorithm returns a wrong answer with probability at most $Q/2$.*

Proof. Observe that Algorithm 4.1 stops only when the random walk reaches a vertical node at depth-$(r + \log_3 \frac{1}{Q})$. If the answer is wrong in round r, the walk must have made at least $r + \log_3 \frac{1}{Q}$ backward moves in some vertical path. The probability of this, in round r, is at most $(1/3)^{r+\log_3(1/Q)} \leq Q/3^r$. Summing over all rounds and applying a union bound, the overall failure probability at most $\sum_{r=1}^{\infty} Q/3^r \leq Q/2$. ◀

▶ **Theorem 7.** *With probability at least $1 - Q$, Algorithm 4.1 halts, having made $O(\log \frac{k}{Q})$ noisy comparisons, and returns $Pred_1[q]$.*

Proof. Let e_{i^*} be $L_1[Pred_1[q]]$. By Lemma 6, the probability of Algorithm 4.1 returning a wrong answer is at most $Q/2$. Next, we show the algorithm returns a correct answer with probability at least $1 - Q/2$ and makes $O(\log \frac{k}{Q})$ noisy comparisons. Then by the union bound, the theorem holds.

Consider the round $r = i^*$, where the random walk has made $c \cdot (i^* + \log_3 \frac{1}{Q})$ moves. Among these, let m_f and m_b be the numbers of forward moves and backward moves, respectively. In order to return a correct answer in this round, the random walk must satisfy

$$m_f - m_b \geq i^* + i^* + \log_3 \frac{1}{Q}\,. \tag{1}$$

Here, the first i^* arises from horizontal forward moves to the horizontal node of e_{i^*}, while the $i^* + \log_3 \frac{1}{Q}$ component is the condition of returning an answer in round $r = i^*$. By the Chernoff-Bound argument of Feige et al. [7], $c' \cdot (2 \cdot i^* + \log_3 \frac{1}{Q})$ moves suffice to satisfy inequality (1) with probability at least $1 - Q/2$. Therefore, by setting $c = 2 \cdot c'$, we have $c \cdot (i^* + \log_3 \frac{1}{Q}) \geq c' \cdot (2 \cdot i^* + \log_3 \frac{1}{Q})$ and thus, the algorithm returns the correct answer in round i^* with probability at least $1 - Q/2$. Moreover, the total number of noisy comparisons is $O(i^* + \log \frac{1}{Q}) = O(\log \frac{2^{i^*}}{Q}) = O(\log \frac{k}{Q})$, from the definition of e_{i^*}. ◀

Although Theorem 7 shows that Algorithm 4.1 is result sensitive with error tolerance at most $Q/2$, it is a Las Vegas algorithm. That is, for every integer K, there is a non-zero (albeit negatively exponential in K), probability that the query cost of Algorithm 4.1 exceeds K. Nonetheless, we show in the next subsection that Algorithm 4.1 can be refined such that it *always* terminates with at most $O(\log \frac{\log n}{Q})$ noisy comparisons, while the expected number of noisy comparisons is $O(\log \frac{k}{Q})$.

4.2 The Ultimate Phase-1 Algorithm

The refined algorithm, which we call our Ultimate Phase-1 Algorithm, is as simple as this:

Algorithm 4.2 The Ultimate Algorithm.

Run Algorithm 4.1 for at most $R = \log \log n$ rounds
if Algorithm 4.1 has not yet returned an answer **then**
  Run Feige on L_1, with tolerance Q

Since Feige algorithm always terminates, Algorithm 4.2 always terminates. Observe that after R rounds of Algorithm 4.1, $O(\log \log n + \log \frac{1}{Q}) = O(\log \frac{\log n}{Q})$ comparisons are made. Running Feige on L_1, the cost is bounded by $O(\log \frac{\log n}{Q})$. As a result, in the worst case, Algorithm 4.2 makes $O(\log \frac{\log n}{Q})$ noisy comparisons. For those queries whose correct result k satisfies $k = \Omega(\log n)$, this bound is within $O(\log \frac{k}{Q})$. We next show that for $k = o(\log n)$, the *expected* number of noisy comparisons is bounded by $O(\log \frac{k}{Q})$.
We state explicitly the Chernoff-Hoeffding bound that is the backbone of our proof.

▶ **Fact 1** (Chernoff-Hoeffding Bound). *Let $X_1, X_2, \ldots, X_N$ be random variables such that $\alpha \leq X_i \leq \beta$ for all $i \in [1, N]$. Let $X = \sum_{i=1}^{N} X_i$ and set $\mu = \mathbb{E}(X)$. Then for all $\delta > 0$, we have:*

$$\Pr[X \leq (1-\delta) \cdot \mu] \leq \exp\left(-\frac{\delta^2 \mu^2}{N(\beta - \alpha)^2}\right).$$

In our application of the bound, X_i is a random variable $\in \{+1, -1\}$ indicating the ith move is forward ($X_i = 1$) or backward ($X_i = -1$), so that $X = m_f - m_b$. As a result, we have $\alpha = -1$, $\beta = 1$, $\mu = N/3$, and by setting $\delta = 1 - \tau/\mu$, with τ being the threshold for $m_f - m_b$ signifying termination, we have:

$$\Pr[X \leq \tau] \leq \exp\left(-\frac{(\mu - \tau)^2}{4N}\right).$$

Consider a query q with $e_{i^*} = L_1[Pred_1[q]]$. Let $M = i^* + \log \frac{1}{Q}$ and $\tau = 2 \cdot i^* + \log \frac{1}{Q}$. As shown in the proof of Theorem 7 and by this Chernoff-Hoeffding Bound, for some sufficiently large constant c, when $N = c \cdot M$,

$$\Pr[X \le \tau] \le \exp\left(-\frac{(\mu-\tau)^2}{4N}\right) \le Q\,,$$

and so Algorithm 4.1 stops in the $(r^* = i^*)^{\text{th}}$ round with probability at least $1 - Q$.

Next, we show the *expectation* of the cost of running all $R = \log\log n$ rounds in Algorithm 4.1, plus the cost for Feige is bounded by $O(N) = O(M) = O(\log\frac{k}{Q})$.

▶ **Lemma 8.** *Let $\tau^{(0)} = \tau$, $\mu^{(0)} = \mu$ and $N^{(0)} = N$ be the parameters in the r^{*th} round. In the $(r^* + i)^{th}$ round for $i \in [1, R - r^*]$, the probability of Algorithm 4.1 not terminating is at most $Q^{1+i\cdot\varepsilon}$, where $\varepsilon = 1/M$.*

Proof. In round $r^* + i$, we denote the correct-stop threshold by $\tau^{(i)}$, the number of moves by $N^{(i)}$, and the expected value of X by $\mu^{(i)}$. According to Algorithm 4.1, in each round, the "correct-stop" threshold, τ, gets increased by 1 and N gets increased by c. Therefore, we have:

$$\tau^{(i)} = \tau^{(0)} + i = \tau^{(0)} + \frac{i}{M}\cdot M \le (1 + \frac{i}{M})\cdot\tau^{(0)} = (1 + i\cdot\varepsilon)\cdot\tau^{(0)}\,,$$

$$N^{(i)} = N^{(0)} + i\cdot c = N^{(0)} + \frac{i}{M}\cdot c\cdot M = (1 + \frac{i}{M})\cdot N^{(0)} = (1 + i\cdot\varepsilon)\cdot N^{(0)}\,,$$

$$\mu^{(i)} = (1/3)N^{(i)} = (1 + i\cdot\varepsilon)\cdot(1/3)N^{(0)} = (1 + i\cdot\varepsilon)\cdot\mu^{(0)}\,.$$

Substituting into the Chernoff-Hoeffding Bound, we have the probability of *not* stopping and returning the correct value being

$$\begin{aligned}\Pr[X \le \tau^{(i)}] &\le \exp\left(-\frac{(\mu^{(i)} - \tau^{(i)})^2}{4N^{(i)}}\right)\\ &\le \exp\left(-(1 + i\cdot\varepsilon)\cdot\frac{(\mu^{(0)} - \tau^{(0)})^2}{4N^{(0)}}\right)\\ &\le Q^{1+i\cdot\varepsilon}\,.\end{aligned}$$ ◀

Denote by $\mathbb{E}[\mathit{Cost}]$ the expectation of the cost of Algorithm 4.2: that is, running all R rounds of Algorithm 4.1, plus the cost of Feige on L_1.

▶ **Lemma 9.** *If the predecessor of a query q in L is at index $k = o(\log n)$, the expected query cost of Algorithm 4.2, i.e., $\mathbb{E}[\mathit{Cost}]$, is $O(\log\frac{k}{Q})$.*

Proof. We partition $\mathbb{E}[\mathit{Cost}]$ into three parts, and bound each:

- Cost_A: the expected cost of the first r^* rounds of Algorithm 4.1,
- Cost_B: the expected cost of the rounds from $r^* + 1$ to R of Algorithm 4.1,
- Cost_C: the expected cost of Feige on L_1.

As the comparisons in the first r^* rounds are mandatory, $\mathit{Cost}_A = N$, where N is the number of comparisons in the first r^* rounds. In order to bound Cost_B, let us consider the following *conceptual* process, which starts after r^* rounds:

Algorithm 4.3 A Conceptual Batch Process.

```
    r ← r*
    while r < R = log log n do
        run the next M rounds as a batch, without stopping
        if the depth-based stopping condition has been met in one of these M rounds then
5:          return the answer corresponding to when the stopping condition was first met
        r ← r + M
```

Clearly, the cost of this conceptual batch process, Algorithm 4.3, is no less than Algorithm 4.1. The latter executes each round separately and terminates immediately when the stopping condition is met. By Lemma 8, for the ℓth batch of M rounds, the probability of the algorithm *not stopping* during that batch is at most $Q^{1+\ell}$, for $\ell = 1, 2, \ldots, z = \lceil \frac{R-r^*+1}{M} \rceil$. That is, the probability of the ℓth batch running is Q^{ℓ}.

Since the cost of each batch of M rounds is $c \cdot M = N$, the expected cost of the whole conceptual batch process, which is an upper bound on $Cost_B$, is (because $Q \leq 1/2$ and $\mathbb{E}[Y] = \sum_{i=1}^{\infty} \Pr[Y \geq i]$):

$$Cost_B \leq \sum_{\ell=1}^{z} Q^{\ell} \cdot N \leq N \cdot \sum_{l=1}^{z} \frac{1}{2^l} < N \,.$$

Finally, the probability that Algorithm 4.2 has not stopped in the first $R = \log\log n$ rounds is at most Q^{1+z}. So we have: $Cost_C \leq Q^{1+z} \cdot \gamma \cdot \log \frac{\log n}{Q}$, for some constant γ.

In total, the expected cost of the first $R = \log\log n$ rounds is at most $2N$. Moreover, the probability of Feige on L_1 occurring is at most Q^{1+z}, which is at most the probability of each noisy comparison made in the first $R = \log\log n$ rounds. Therefore the expected value of $Cost_C$ is at most the expected value of $Cost_A + Cost_B$, up to a constant factor. Putting everything together, we have $\mathbb{E}[Cost] = Cost_A + Cost_B + Cost_C \leq O(N) = O(M) = O(\log \frac{k}{Q})$. ◀

Proof of Theorem 2. Since the correctness of Algorithm 4.2 is easy to verify, combining with Lemma 9, Theorem 2 holds. ◀

5 Conclusion

In this paper, we have designed result-sensitive algorithms under the Noisy Comparison Model for answering predecessor queries. Specifically, our algorithm in Section 3 correctly answers with probability at least $1 - Q$, and makes $O(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q})$ noisy comparisons in the worst case, where k is the index of the predecessor in the sorted list. Incorporating Feige as a black box leads to a clean analysis of this algorithm.

In Section 4, we present a genuinely result-sensitive algorithm such that for the queries with $k = \Omega(\log n)$, the cost is $O(\log \frac{k}{Q})$ in the worst case, and for those queries with $k = o(\log n)$, its expected query cost is bounded by $O(\log \frac{k}{Q})$. Our algorithm nicely bridges the state-of-the-art known bounds – $O(\log \frac{1}{Q})$ for $k = O(1)$ and $O(\log \frac{n}{Q})$ for $k = \omega(1)$ – over the whole spectrum of $k = 1, \ldots, n$. In particular, for $k \in \omega(1) \cap o(n^{\epsilon})$, our algorithm strictly improves the better of the two bounds.

Finally, by solving some key range-query and shortlisting problems, we illustrate how the benefit of our result-sensitive algorithm.

References

1 Javed A Aslam and Aditi Dhagat. Searching in the presence of linearly bounded errors. In *STOC*, pages 486–93, 1991.

2 R. Beigel. Unbounded Searching Algorithms. *SIAM Journal on Computing*, 19(3):522–537, 1990.

3 Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–7, 1976.

4 Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based Search in the Presence of Errors. In *STOC*, pages 130–6, 1993.

5 Xi Chen, Sivakanth Gopi, Jieming Mao, and Jon Schneider. Competitive analysis of the top-K ranking problem. In *SODA*, pages 1245–64, 2017.

6 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms*. Springer, 2016.
7 U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with Noisy Information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.
8 Richard M. Karp and Robert Kleinberg. Noisy Binary Search and Its Applications. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 881–890, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
9 Donald E. Knuth. Supernatural Numbers. In David A. Klarner, editor, *The Mathematical Gardner*, pages 310–25. Springer US, 1981.
10 Stefano Leucci and Chih-Hung Liu. A Nearly Optimal Algorithm for Approximate Minimum Selection with Unreliable Comparisons. *arXiv*, 2018. arXiv:1805.02033.
11 Andrzej Pelc. Searching with known error probability. *Theoretical Computer Science*, 63(2):185–202, 1989.
12 Andrzej Pelc. Searching games with errors—fifty years of coping with liars. *Theoretical Computer Science*, 270(1):71–109, 2002.
13 J. C. Raoult and J. Vuillemin. Optimal unbounded search strategies. In Jaco de Bakker and Jan van Leeuwen, editors, *ICALP*, pages 512–530, 1980.
14 B. Ravikumar. A Fault-Tolerant Merge Sorting Algorithm. In Oscar H. Ibarra and Louxin Zhang, editors, *Computing and Combinatorics*, pages 440–447, 2002.
15 Alfréd Rényi. On a problem of information theory. *MTA Mat. Kut. Int. Kozl. B*, 6:505–516, 1961.
16 R.L. Rivest, A.R. Meyer, D.J. Kleitman, K. Winklmann, and J. Spencer. Coping with errors in binary search procedures. *Journal of Computer and System Sciences*, 20(3):396–404, 1980.
17 M. Saks and A. Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees. In *FOCS*, pages 29–38, 1986.
18 Stanisław M Ulam. *Adventures of a Mathematician*. Univ of California Press, 1991.

A Appendix: Applications

As mentioned in the Introduction, our result-sensitive algorithm for predecessor queries immediately improves two types of algorithms for related problems. We provide details here, proving Corollaries 3 and 4 and Theorem 5.

A.1 Direct Applications of Predecessor Search

Predecessor search appears as a black box in this type of application. We discuss two example problems: (i) Range Count Query, and (ii) Stabbing Count Query. Specifically, we prove Corollaries 3 and 4.

Proof of Corollary 3 for Range Count Query. For the given query range $(a, b]$, we can first apply our Algorithm 4.2 to find the index k_a of the predecessor of a in L, with error tolerance $Q/2$. Then apply Algorithm 4.2 again on the sub-list $L(k_a, n]$ to find the (relative) index of k_b of the predecessor of b, with error tolerance $Q/2$. Since all the elements in $L[1, k_a]$ must not be in the range $(a, b]$, it can be verified that k_b is exactly the number *count* of elements of L falling in $(a, b]$. The correctness of the algorithm is obvious and, allowing for $count = 0$, the cost is bounded by $O(\log \frac{k_a+1}{Q} + \log \frac{count+1}{Q})$ in expectation. ◀

Proof of Corollary 4 for Stabbing Count Query. Denote by L_ℓ (respectively, L_r) the list of the intervals of S sorted by their left (respectively, right) endpoints. Let k_ℓ and k_r be the indexes of the predecessors of the query value q in L_ℓ and L_r, respectively. Observe that the number of intervals stabbed by q can be computed as $count = k_\ell - k_r$. Thus, we can first apply Algorithm 4.2 on L_r to find the index k_r of $Pred_r[q]$, and then apply the algorithm

on the sub-list $L_\ell[k_r+1,n]$ to find the index k'_ℓ of $Pred_\ell[q]$. Thus, $k'_\ell = k_\ell - k_r = count$. By setting the error tolerances of each search to $Q/2$, the correctness of the algorithm is straightforward, and the cost is bounded by $O(\log \frac{k_r+1}{Q} + \log \frac{count+1}{Q})$ in expectation. ◀

A.2 Predecessor Search with Generalised Comparison Oracle

In this subsection, we illustrate how our result-sensitive algorithm can be applied to improve Type-II algorithms by generalising the comparison oracle. As an example, we solve the Shorlisting Problem.

Recall that in the Shortlisting Problem, there are two disjoint sorted lists L_A and L_B, the goal is to return the *set* of m smallest elements (the "shortlist") in the conceptual merged sorted list of L_A and L_B. Obviously, actually merging A and B Feige– making $O(N \log \frac{N}{Q})$ noisy comparisons (where $N = n_A + n_B$) – immediately solves the problem. However, our goal is to solve the problem with significantly fewer comparisons (in expectation) than this merge sledgehammer. Instead, we aim for $O(\log \frac{k}{Q})$ noisy comparisons, where k is the number of elements from L_A in the shortlist.

The crucial observation is that if we can determine the number k, then we can output the whole shortlist without further comparisons. As we illustrate below, we can compute k with a binary-search-like algorithm. Without loss of generality, we assume $n_A = n_B = m$ since all the elements with indices greater than m in either of the lists can be safely pruned.

The Comparison Oracle for Shortlisting

▶ **Observation 1.** *For $1 \le i \le m$, $L_A[i]$ is in the shortlist if and only if $L_A[i] < L_B[m-i+1]$.*

Proof. First, suppose that $L_A[i]$ is in the shortlist; then there are at most $m-i$ elements from L_B in the shortlist. Thus, $L_B[m-i+1]$ cannot be in the shortlist and therefore, $L_A[i] < L_B[m-i+1]$.

Conversely, suppose that $L_A[i] < L_B[m-i+1]$. As a result, if $L_B[m-i+1]$ is in the shortlist, then $L_A[i]$ must be in the shortlist. However, this forces the shortlist to have size at least $m+1$. Therefore, there are no more than $m-i$ elements from L_B in the shortlist, implying that there are at least i elements from L_A in the shortlist. Hence, $L_A[i]$ is in the shortlist. ◀

By Observation 1, we can decide whether $L_A[\lfloor m/2 \rfloor]$ is in the shortlist by comparing it with $L_B[m - \lfloor m/2 \rfloor + 1]$. We thus determine which half of the list L_A should be further considered, which is a binary-search-like algorithm. Since checking the condition in Observation 1 only takes one (deterministic) comparison, by the same analysis of Feige [7], we know that we can identify the largest element $L_A[k]$ from L_A in the shortlist with $O(\log \frac{m}{Q})$ noisy comparisons, with an error tolerance Q.

The Comparison Oracle for Our Result-Sensitive Algorithm

We adapt the two-phase algorithm proposed in Section 4 to obtain a result-sensitive algorithm for the Shortlisting Problem. Again, the second phase can be solved by binary search, which we just designed efficiently. We focus on Phase 1, where our goal is to identify the largest element in the power-of-two list of L_A, denoted by L_{A1}, which should be in the shortlist. We simply design the search-space partitions for both the horizontal nodes and the vertical nodes, as shown in Figures 3 and 4. The subsequent analysis follows immediately.

For a horizontal node $e_i = L_{A1}[i] = L_A[2^{i-1}]$, the space partition for its edges is as follows:

- the left edge corresponds to the case that e_i is not in the shortlist: $e_i > L_B[m - 2^{i-1} + 1]$;
- the right edge corresponds to the case that e_{i+1} is in the shortlist: $e_{i+1} < L_B[m - 2^i + 1]$;
- the downward edge corresponds to the case that e_i is in the shortlist, but e_{i+1} is not.

For a vertical node of e_i, the space partition for its two edges is as follows:

- the downward edge corresponds to the case that e_i is in the shortlist, but e_{i+1} is not;
- the upward edge corresponds to the other case.

By plugging the above comparison oracle to Algorithm 4.2, we have a result-sensitive Phase-1 algorithm for the Shortlisting Problem.

Proof of Theorem 5. By the above analysis, and allowing for $k = 0$, we obtain an algorithm to find k in L_A with $O(\log \frac{k+1}{Q})$ noisy comparisons in expectation. By symmetry, we can find the dual index, $m - k$, for L_B with an expected cost $O(\log \frac{m-k+1}{Q})$. Therefore, by running both of the algorithms for L_A and L_B *simultaneously* and stopping as soon as either terminates, the Shortlisting Problem can be solved with $O(\log \frac{\min\{k, m-k\}+1}{Q})$ noisy comparisons in expectation. The answer is correct with probability at least $1 - Q$. ◀