Author/s:
Gan, J;Wirth, A;Zhang, X

Title:
An Almost Optimal Algorithm for Unbounded Search with Noisy Information

Date:
2022-06-29

Citation:
Gan, J., Wirth, A. & Zhang, X. (2022). An Almost Optimal Algorithm for Unbounded Search with Noisy Information. Czuma, A (Ed.) Xin, Q (Ed.) Leibniz International Proceedings in Informatics, 227, pp.25:1-25:15. Dagstuhl Publishing. https://doi.org/10.4230/LIPIcs.SWAT.2022.25.

Persistent Link:
https://hdl.handle.net/11343/311310

An Almost Optimal Algorithm for Unbounded Search with Noisy Information

Junhao Gan
School of Computing and Information Systems, The University of Melbourne, Australia

Anthony Wirth
School of Computing and Information Systems, The University of Melbourne, Australia

Xin Zhang
School of Computing and Information Systems, The University of Melbourne, Australia

Abstract

Given a sequence of integers, $\mathcal{S} = s_1, s_2, \ldots$ in ascending order, called the *search domain*, and an integer t, called the *target*, the *predecessor problem* asks for the *target index* N such that s_N is the largest integer in $\mathcal{S}$ satisfying $s_N \leq t$. We consider solving the predecessor problem with the least number of queries to a *binary comparison oracle*. For each query index i, the oracle returns whether $s_i \leq t$ or $s_i > t$. In particular, we study the predecessor problem under the UnboundedNoisy setting, where (i) the search domain $\mathcal{S}$ is *unbounded*, i.e., $n = |\mathcal{S}|$ is unknown or infinite, and (ii) the binary comparison oracle is *noisy*. We denote the former setting by Unbounded and the latter by Noisy. In Noisy, the oracle, for each query, *independently* returns a *wrong* answer with a fixed constant probability $0 < p < 1/2$. In particular, even for two queries on the same index i, the answers from the oracle may be different. Furthermore, with a noisy oracle, the goal is to correctly return the target index with probability at least $1 - Q$, where $0 < Q < 1/2$ is the *failure probability*.

Our first result is an algorithm, called **NoS**, for Noisy that improves the previous result by Ben-Or and Hassidim [FOCS 2008] from an expected query complexity bound to a worst-case bound. We also achieve an expected query complexity bound, whose leading term has an *optimal* constant factor, matching the lower bound of Ben-Or and Hassidim. Building on **NoS**, we propose our **NoSU** algorithm, which correctly solves the predecessor problem in the UnboundedNoisy setting. We prove that the query complexity of **NoSU** is $\sum_{i=1}^{k}(\log^{(i)} N)/(1 - H(p)) + o(\log N)$ when $\log Q^{-1} \in o(\log N)$, where N is the target index, $k = \log^* N$, the iterated logarithm, and $H(p)$ is the entropy function. This improves the previous bound of $O(\log(N/Q)/(1 - H(p)))$ by reducing the coefficient of the leading term from a large constant to 1. Moreover, we show that this upper bound can be further improved to $(1-Q)\sum_{i=1}^{k}(\log^{(i)} N)/(1-H(p)) + o(\log N)$ in expectation, with the constant in the leading term reduced to $1 - Q$. Finally, we show that an information-theoretic lower bound on the expected query cost of the predecessor problem in UnboundedNoisy is at least $(1 - Q)(\sum_{i=1}^{k} \log^{(i)} N - 2k)/(1 - H(p)) - 10$. This implies the constant factor in the leading term of our expected upper bound is indeed optimal.

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

Keywords and phrases Fault-tolerant search, noisy binary search, query complexity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.25

Funding *Junhao Gan*: J.G. is supported in part by Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA) DE190101118.
Anthony Wirth: A.W. is supported by the Faculty of Engineering and Information Technology at The University of Melbourne.
Xin Zhang: X. Z.'s research is supported by an Australian Government Research Training Program (RTP) Scholarship.

Acknowledgements We thank Przemysław Uznański and Dariusz Dereniowski for the discussions on noisy search, particularly for highlighting the nuances of the binary and ternary comparison models.

1 Introduction

Consider a sequence of integers $\mathcal{S} = s_1, s_2, \ldots$ sorted in ascending order. The *predecessor problem* for a given integer, t, asks for the index N such that s_N is the largest integer in $\mathcal{S}$ satisfying $s_N \leq t$. We assume s_N always exists. We refer to $\mathcal{S}$ as the *search domain* and to N as the *target index*. The goal of the predecessor problem is to find the target index with the least number of *queries* of a given *binary comparison oracle*, F. Specifically, let $n = |\mathcal{S}|$; F is a function from the *index domain* of $\mathcal{S}$, i.e., $\{1, 2, \ldots, n\}$, to a binary domain $\{1, -1\}$, such that $F(i) = -1$, if $s_i \leq t$, but $F(i) = 1$, if $s_i > t$. Clearly, when n is known and the oracle F always returns correct answers, the well-known binary search algorithm solves the predecessor problem optimally with $\lceil \log_2 n \rceil$ queries.

Arguably, the predecessor problem is one of the most important and classic problems in computer science. Departing from the standard setting, there is much research on variants of binary search [2, 23, 31]. We are particularly interested in two settings of the predecessor problem, Unbounded and Noisy, as well as the combination of them.

The Unbounded Setting. In the Unbounded setting, the size of $\mathcal{S}$, i.e., n, is either *unknown* or *unbounded*, i.e., infinite. The Unbounded problem was studied by Bentley and Yao [5] and has many applications, e.g., in range-count queries and in shortlisting [16]. Several algorithms solving the Unbounded problem are *result sensitive*: their query complexity does not depend on the (possibly infinite) size of the search domain, but rather the location of the result (i.e., the target index N) in $\mathcal{S}$, a property desirable in the context of local algorithms [21]. Bentley and Yao [5] wrote the state-of-the-art algorithm, **BY**, with query complexity $\sum_{1 \leq i \leq k} \lfloor \log^{(i)}(N) \rfloor + 5 + 2k$, where $\log^{(i)} N$ is the nested logarithm and $k = \log^* N$ is the iterated logarithm of N.

The Noisy Setting. In the Noisy setting, the comparison oracle might not always respond correctly, or truthfully: it could return incorrect results. In this case, the comparison oracle is said to be *noisy*. This setting captures the fact that real-world information can be noisy and, hence, the results of comparisons might not be correct. There has been a long line of work for dealing with faulty information or uncertainties in searching problems, from the 1965 Rényi-Ulam game [27] to some more recent work [15, 11, 16]. In this paper, we focus on the *probabilistic error model* [27, 29, 17, 11, 15], where the oracle F behaves as follows.

▶ **Definition 1.** *For a specified fixed constant* error probability p*, with* $0 < p < 1/2$*, in the* ***probabilistic error model****, on each query the comparison oracle* F independently *returns a wrong answer with probability* p*.*

The goal of the predecessor problem under Noisy is to return the *correct* target index with probability at least $1 - Q$, where $0 < Q < 1/2$ is the *failure probability*.

The UnboundedNoisy Setting. The main focus of this paper is on algorithms solving the predecessor problem in the UnboundedNoisy setting. This setting combines both Unbounded and Noisy, and UnboundedNoisy has the following characteristics:

- First, due to the unbounded search domain, the anticipated query complexity is a function of the target index, N, rather than the unknown or unbounded n;
- Second, inheriting from Noisy, the query complexity should also be related to both the error probability, p, and the failure probability, Q. On the one hand, aligned with previous work [17, 4, 15, 14, 16], parameter p is treated as a constant. On the other hand, as the

target index N depends on the target integer t, it no longer makes sense to aim at "high success probability" in terms of N. We require that $1/Q$ is far less than N; otherwise, one can turn this problem into the bounded NOISY case by searching the first c/Q elements for some constant c. In particular, we assume throughout that $\log Q^{-1} \in o(\log N)$. As we see shortly, in this case, the *leading* term in the query complexity is $\log_2 N$, which is typically the most important term in the query cost. We hence focus on reducing the hidden constant in such a leading term.

1.1 Our contributions

Bounded search

Our first contribution is an improved algorithm, **NoS**, for the NOISY setting. It achieves the same asymptotic query complexity as the state-of-the-art algorithm by Ben-Or and Hassidim [4]. However, our query bound is more powerful by being *worst-case*, while theirs is in expectation. Comparing the constant factors, when $\log Q^{-1} \in o(\log n)$, the leading term in their query bound is $(1-Q)/(1-H(p))\log_2 n$, where $H(p) = -p\log p - (1-p)\log(1-p)$ is the well-known entropy function. We show that **NoS** can achieve the same leading term in the query bound, in expectation, but the randomness stems purely from the algorithm mechanism rather than the assumption on the target index distribution. Furthermore, according to the lower bound of Ben-Or and Hassidim [4], the constant factor on this leading term in the query bounds of both their algorithm and **NoS** is indeed optimal.

▶ **Theorem 2.** *Our **NoS** algorithm solves the predecessor problem in the NOISY setting, with constant error probability $0 < p < 1/2$, and a failure probability $0 < Q < 1/2$, with worst-case query complexity:*

$$\frac{1}{1-H(p)}\left(\log_2 n + O(\log\log n) + O(\sqrt{\log n \log Q^{-1}} \cdot \log\frac{\log n}{\log Q^{-1}}) + O(\log Q^{-1})\right). \quad (1)$$

▶ **Corollary 3.** *When $\log Q^{-1} \in o(\log n)$, **NoS** achieves* expected *query complexity:*

$$\frac{1-Q}{1-H(p)}\left(\log_2 n + o(\log n)\right). \quad (2)$$

According to the lower bound of Ben-Or and Hassidim [4], we have the following:

▶ **Fact 4** ([4]). *The expected query complexity for solving the predecessor problem under NOISY, parameterized by $0 < p < 1/2$ and $0 < Q < 1/2$, is at least:*

$$\frac{1-Q}{1-H(p)}\log_2 n - 10.$$

Therefore, the constant factor of the leading term in the query complexity of **NoS** is tight, when $\log Q^{-1} \in o(\log n)$. Very recently, and concurrently with the technical development of this paper, Dereniowski et al. [10] also proposed improvements over the results of Ben-Or and Hassidim, achieving a query bound with the same leading term as ours. We compare the two contributions further in Section 2.

Unbounded search

Building upon Theorem 2, our second contribution is a new algorithm, **NoSU**, which improves the query complexity bound for UNBOUNDEDNOISY.

▶ **Theorem 5.** *Our **NoSU** algorithm solves the predecessor problem in the* UnboundedNoisy *setting, parameterized by a constant error probability $0 < p < 1/2$ and a failure probability $0 < Q < 1/2$, with worst-case query complexity:*

$$\frac{1}{1-H(p)}\left(\sum_{i=1}^{k}\log_2^{(i)} N + O(\log\log N) + O(\sqrt{\log N \log Q^{-1}}\cdot\log\frac{\log N}{\log Q^{-1}}) + O(k\log\frac{k}{Q})\right),$$

where $k = \log^* N$ is the iterated logarithm of N.

▶ **Corollary 6.** *When $\log Q^{-1} \in o(\log N)$, **NoSU** has* expected *query complexity:*

$$\frac{1-Q}{1-H(p)}\left(\sum_{i=1}^{k}\log_2^{(i)} N + o(\log N)\right). \tag{3}$$

Our **NoSU** algorithm significantly reduces the hidden constant factor in the state-of-the-art bound by Epa et al. [16] and Dereniowski et al. [11], i.e., $O(\log(N/Q)/(1-H(p)))$. The coefficients of the leading terms in these bounds are required to be sufficiently large to invoke the Chernoff bound. In contrast, by Corollary 6, the constant factor in the leading term, $\log_2 N$, of **NoSU**, is just $\frac{1-Q}{1-H(p)}$.

Our final contribution is a lower bound on the expected query complexity for the predecessor problem under UnboundedNoisy.

▶ **Theorem 7.** *The expected query complexity for solving the predecessor problem under* UnboundedNoisy*, parameterized by $0 < p < 1/2$ and $0 < Q < 1/2$, is at least:*

$$\frac{1-Q}{1-H(p)}\left(\sum_{i=1}^{k}\log_2^{(i)} N - 2k\right) - 10. \tag{4}$$

Combining Theorem 7 and Corollary 6, the leading term in the expected query complexity of **NoSU** is thus optimal.

1.2 Applications of UnboundedNoisy

The UnboundedNoisy setting naturally produces algorithms whose costs are result-sensitive: the query complexity relies on the target index rather than the size of the search domain. Epa et al. [16] listed a range of applications that solves the predecessor in a noisy setting, including, for instances, counting the number of elements from a sorted list that fall into a range, and obtaining the top-ranked elements from two sorted lists. When applying **NoSU** to bounded domains, **NoSU** is an improvement to the algorithm by Epa et al. [16] in the result-sensitive setting.

The unbounded domain also arises from the context of local algorithms [21], where computing units in a distributed environment have access to local, but not some global information, e.g., the total size of the domain. Since the search domain is distributed, it is hard to discern the total size and run the normal binary search algorithm. Kim and Winston [20] adapted **BY** [5], an unbounded binary search algorithm to the problem maximum power point tracking, processing a large volume of data created from the voltage change in logarithmic time and avoiding a linear scan of the input. Since unbounded domains are often a consequence of large-scale, automatically generated, or distributed datasets, it is natural to consider the comparisons on those data points with the presence of errors [33]. In software

testing, there often is a breakpoint for certain resource, be it time, space or workload [32]. For instance, finding the number of requests of services that break a load-balancer requires a tester to check the status of the software under a range of different request numbers. Due to the randomness commonly existing in those tests, it is reasonable to assume that the search domain (possible breakpionts) is not only unbounded but also noisy.

2 Related Work

Computing with faulty or uncertain information has long been active field of research. An early model that deals with uncertainty is the Rényi-Ulam game [26, 28, 34]. In this two-player game, Player 1's goal is to identify an element in a finite set by posing questions to Player 2. In answering these questions, Player 2 tries to stop Player 1 from meeting their goal by lying sometimes. One standard model for is *fixed lies* [11, 25, 7, 9], where the number of lies Player 2 can tell is bounded. Another common *error model* is *linearly bounded lies* [13, 1], where the number of lies the adversary is allowed to tell is at all times linear in the number of queries. The error model in this paper is the *probabilistic model* [17], where for each query the adversary (a.k.a., oracle) independently lies with fixed probability. Feige et al. [17] proposed algorithms for not just searching, but also sorting, ranking and merging with the probabilistic model. They refer to this model as the noisy comparison tree: at each "node" (query), the result leads us in two different directions, and the configurations of the query responses naturally form a binary tree. When searching an integer domain, algorithms are often based on binary search [6, 29, 30].

There are well-known connections between these models. We consider the **MWU** algorithm by Dereniowski et al. [12], who studied a graph search framework proposed by Emamjomeh-Zadeh et al. [15]. Though designed for the fixed-lies model, with carefully selected parameters, it could return the correct answer with desirable guarantees under the probabilistic error model, which is the setting of Noisy.

Bentley and Yao first introduced the Unbounded setting for binary search and the definitive algorithm, **BY** [5], detailed in Section 4. Invoking results from prefix codes, they also established a lower bound for Unbounded predecessor search, which is at the heart of our lower bound in Section 6. Their bounds [5] were later improved by Beigel [3] on the non-leading terms. As our analysis focuses on the leading term, we still consider **BY** as the state of the art.

Combining the two settings, UnboundedNoisy was studied by Pelc [24] and then by Aslam and Dhagat [1], who achieved a bound of $O(\log N)$ for the linearly bounded lies model. Dereniowski et al. [11] and Epa et al. [16] recently studied the problem under the same setting as ours, but our algorithm **NoSU** has a better constant on the leading term.

Table 1 displays state-of-the-art results for those problems and the query complexity.

Although the hidden constant on the leading term $\log n$ of **Feige**'s bound is not as small as that of **BH**, a nice feature of **Feige** is that it is a Monte-Carlo algorithm and provides a worst-case query complexity. We incorporate **Feige** as a subroutine in our algorithm, **NoSU**.

Simultaneously with the technical development of our work, Dereniowski et al. [10] applied the Bayesian-update technique, further developing and improving the result of Ben-Or and Hassidim [4], to sorted integers as well as graphs. Similar to **NoS**, their algorithms also improve the previous bounds [17, 4, 10] in Noisy. Their algorithm **PŁU** and our **NoS** (stated in Theorem 2) achieve the same query complexity bound in the leading term. Upon closer inspection, **PŁU** is tighter on the second dominating term ($O(\sqrt{\log n \log Q^{-1}})$) than ours. While we are interested in constant error probability, **PŁU** is able to handle the

Table 1 Summary of complexity results for predecessor search in a variety of settings.

Problem	*Algorithm*	*Upper Bound Complexity*	*Source*
BOUNDED	**BinarySearch**	$\lceil \log n \rceil$	Folklore
UNBOUNDED	**BY**	$\sum_{1 \le i \le \log^* N} \lfloor \log^{(i)}(N) \rfloor + 5 + 2\log^* N$	[5]
NOISY	**BH**	Expected $(1 - H(p))^{-1} \cdot (\log n + O(\log\log n) + O(\log Q^{-1}))$	[4]
NOISY	**Feige**	$(1 - H(p))^{-1} O(\log n/Q)$	[17]
NOISY	**PŁU**	$(1 - H(p))^{-1} \cdot (\log n + O(\sqrt{\log n \log Q^{-1}}) + O(\log Q^{-1}))$	[10]
UNBOUNDEDNOISY	**Unbounded**	$(1 - H(p))^{-1} O(\log(N/Q))$	[11]
UNBOUNDEDNOISY	**EGW**	Expected $(1 - H(p))^{-1} O(\log(N/Q))$	[16]

case when p is not a constant For this work, this is not an issue as we already require that p be a constant. Although both their and our algorithms share similar ideas of selecting items by adjusting the pre-assigned weights, interestingly, the actual details are substantially different: while **PŁU** processes the items in epochs with repeated queries, our **NoS** first adopts the existing **MWU** algorithm as a blackbox and then refines the certain candidates in the subsequent stages. Thanks to this, the proof of **NoS** is relatively simpler. Moreover, **NoS** is also easily incorporated into the UNBOUNDEDNOISY setting which is the main focus of this work.

There is a variety of models, and accompanying algorithms, for solving binary search in noisy conditions; several of these models differ substantially from ours [19, 22, 18, 8]. For instance, Karp and Kleinberg [19] considered a setting where the search is conducted on a sequence of coins, each equipped with a fixed (but distinct) probability of showing head when tossed. The coins are sorted according to their head probabilities. The goal is to find the leftmost coin whose head probability is lower than some target probability, with the least number of coin tosses. This is a more general noisy binary search model than NOISY as the probability of making a mistake varies from query to query. Interestingly, **NoS** almost matches the information-theoretical lower bound obtained by Karp and Kleinberg [19].

3 Bounded Search with Noisy Information

We begin our technical presentation with an introduction to an existing algorithm in NOISY setting, but with a different oracle definition, culminating in our algorithm **NoS**, as part of the proof of Theorem 2.

3.1 Preliminary: A Graph-based Noisy Search Algorithm

We outline the **MWU** (**M**ultiplicative **W**eight **U**pdate) algorithm [11][1] in this section, as a preliminary to our **NoS** algorithm. **MWU** is designed for the problem of searching in a graph via queries to an oracle. Since a sorted sequence of integers can be viewed as a *path graph*, **MWU** is also applicable to the predecessor problem. However, **MWU** requires a *ternary oracle* in the probability model, which is a stronger oracle than the binary oracle as discussed earlier. Specifically, for the index domain $[n]$, a *ternary comparison oracle* is

[1] Here and onwards we cite the ArXiv version [11] instead of the conference version [12]: as the authors themselves later noted, the bounds established in the conference version turn out to be inaccurate.

defined as a function F' such that for all $i \in [n]$, $F'(i) = -1$ if $s_i < t$, $F'(i) = 0$ if $s_i = t$, while $F'(i) = 1$ if $s_i > t$. Nonetheless, as we show shortly, we can strengthen the **MWU** algorithm to be applicable with a binary oracle without affecting its query complexity.

To avoid distractions, we describe **MWU** in the context of predecessor problem on a sorted sequence of integers. Let us define some notation first. For an index $i \in [n]$, define its *left set* as $\mathcal{L}_i = [1, i-1]$ and its *right set* as $\mathcal{R}_i = [i+1, n]$, each an interval that includes both endpoints. For a query on i, the response $F'(i) = 1$ implies $t \in \mathcal{L}_i$, while $F'(i) = -1$ implies $t \in \mathcal{R}_i$, and $F'(i) = 0$ implies $t = s_i$. For a specified error probability $0 < p < 1/2$, each response is correct, independently, with probability at least $1 - p$. An index j is called *compatible* with the oracle's response for index i if j is in the implied interval[2], and is called *incompatible* otherwise. We learn the target t by eliminating indices that are incompatible with too many queries, and assess the likelihood of an index being the target via the assignment of weights at each round of the algorithm. For every index $i \in [n]$, let $\mu(i)$ be the weight of i, initialized to be 1. We overload function μ so that for a subset of indices $S \subseteq [n]$, define $\mu(S) = \sum_{i \in S} \mu(i)$, and let $\mu = \mu([n])$, the total weight in array $[n]$. The weighted distance sum of index $i \in [n]$ is

$$\Phi(i) = \sum_{j \in [n]} \mu(j) \cdot |i - j| \,.$$

Let $q = \operatorname{argmin}_{i \in [n]} \Phi(i)$ be the index that minimizes the weighted distance sum, Φ. Analogous to standard binary search, q is the *median* that we query, and the response suggests eliminating *half* of the potential candidates for the target. In Noisy, we choose q so that the nodes preceding q constitute *roughly* half the weight and those following q *roughly* possess half the total weight.

Algorithm **MWU** proceeds in iterations. Initially each index has weight 1. At each iteration, **MWU** queries the comparison oracle at q: it reduces the weight of each *incompatible* index multiplying by factor $1/\Gamma$, for some $\Gamma > 0$ defined later. **MWU** terminates when only one index in $[n]$ has weight at least $1/\Gamma^L$, for some pre-determined integer $L > 0$. This termination condition assumes that the oracle makes at most L mistakes throughout the search, a property of an oracle in the *fixed-lies* error model. However, by setting the parameters appropriately, the fixed-lies error model can be transformed into the probabilistic error model. In particular, Dereniowski et al. [11], show the following:

▶ **Fact 8.** *Over all iterations of the entire process of **MWU**, the* amortized rate of decrease *of the total weight is at least $\frac{\Gamma+1}{2\Gamma}$. Specially, if the total weight at the beginning of an iteration is μ, at the end of this iteration, the total weight decreases to $\frac{\Gamma+1}{2\Gamma}\mu$, amortized.*

The main result of **MWU** is as follows.

▶ **Theorem 9** (Section 3.3 [11]). *Algorithm **MWU** solves the predecessor problem under Noisy parameterized by $0 < p < 1/2$ and $0 < Q < 1/2$ with a ternary comparison oracle, with at most*

$$\frac{1}{1 - H(p)} \left(\log_2 n + O(\sqrt{\log n \log Q^{-1}} \cdot \log \frac{\log n}{\log Q^{-1}}) + O(\log Q^{-1}) \right) \tag{5}$$

queries, and the parameters are set as $L = (r \log_2 n)/(1 - H(r))$, $\Gamma = (1-r)/r$, $r = (1-\varepsilon_0)/2$, and $\varepsilon_0 = (1-2p)/\left(1 + \sqrt{8 \ln Q^{-1} / \ln n}\right)$.

[2] Treat $F'(i) = 0$ as $t \in [s_i, s_i]$

In fact, Dereninowski et al. [11] showed that as long as an iterative algorithm can reduce the total weight by an (amortized) rate of $(\Gamma+1)/(2\Gamma)$, the algorithm satisfies the bound in Expression (5) with the parameters set in Theorem 9. With high probability, it takes that many (Expression 5) iterations to reduce the total weight to an amount such that only one node can possibly have weight more than $1/\Gamma^L$. Unfortunately, Theorem 9 is not directly applicable to the setting with a binary comparison oracle. The binary oracle cannot provide answers to the queries as strong as those a ternary oracle provides. Specifically, for a query index i, a binary oracle can only separate $[n]$ into two parts, $\mathcal{L}_i \cup \{i\}$ and $\mathcal{R}_i$, rather than three parts: $\mathcal{L}_i$, $\{i\}$ and $\mathcal{R}_i$. Hence Fact 8 need not hold for the binary oracle.

3.2 Our Two-Stage Noisy Search Algorithm

In this section, we present our algorithm **NoS** (**No**isy **S**earch), to solve the predecessor problem under the NOISY setting with a binary oracle, achieving the same query complexity as **MWU**. Algorithm **NoS** first finds a smaller candidate set that includes the target index and then searches on the set to find the target index. Recall that for a queried index v, the possible compatible sets yielded from a query response are $\mathcal{L}_v \cup \{v\}$ when $F(v) = -1$ and $\mathcal{R}_v$ when $F(v) = 1$. We call the first response that includes the queried index an *inclusive answer* and the second an *exclusive answer*. The basic idea of our **NoS** is that it first collects all the queried indices into a set M as if a ternary oracle is adopted, and then, it further searches the target index in M in the second stage.

Algorithm 1 Algorithm **WeightedBinary** with Γ and L as parameters (defined in Theorem 9).

```
1:  function WeightedBinary(A, L, Γ)
2:      M ← ∅
3:      for v ∈ [1, n] do μ(v) ← 1 and l_v ← 0
4:      while more than one index x ∈ [n] has l_x ≤ L do
5:          Query the oracle at q ← arg min_{x∈[1,n]} Φ(x)
6:          if the query response is an inclusive answer then
7:              M ← M ∪ {q}, μ(q) ← 0, and l_q ← L + 1
8:          for all incompatible v do
9:              μ(v) ← μ(v)/Γ and l_v ← l_v + 1
10:     M ← M ∪ {x ∈ [n] : l_x ≤ L}
11:     return M
```

Algorithm 1 is the first stage of **NoS**, where a small set of potential targets is identified. We adapt the ternary-oracle algorithm **MWU** in the previous section to our setting. Compared to its ternary counterpart, **WeightedBinary** returns a set M that contains the single index that has sufficiently large weight, *plus* all queried indices for which an inclusive answer is returned. We have:

▶ **Lemma 10.** *At each iteration, if the total weight at the beginning of the iteration is μ, then **WeightedBinary** reduces the total weight to $\frac{\Gamma+1}{2\Gamma}\mu$.*

Proof. To see this, we compute the rate by which the total weight diminishes at an iteration. It is easy to prove that by the definition of q, $\mu(\mathcal{L}_q) \le \mu/2$ and $\mu(\mathcal{R}_q) \le \mu/2$.

At any iteration, consider the response of the binary oracle for a queried node q. Suppose the total weight is μ. Then by definition $\mu = \mu(\mathcal{L}_q) + \mu(\mathcal{R}_q) + \mu(q)$. If the answer is exclusive, then weights in $\mathcal{L}_q \cup \{q\}$ are lowered. Since $\mu(\mathcal{L}_q \cup \{q\})/\Gamma + \mu(\mathcal{R}_q) \le \frac{\mu}{\Gamma} + \frac{\Gamma-1}{\Gamma}\mu(\mathcal{R}) \le \frac{\Gamma+1}{2\Gamma}\mu$, the total weight is reduced sufficiently. If the answer is inclusive, then the total weight is lowered by a factor at least $\frac{\Gamma+1}{2\Gamma}$, as here q has its weight reduced to 0. ◀

Observe that the bound in Lemma 10 holds at every iteration, contrasting with Fact 8, which holds in amortization. From Theorem 9 we conclude the following.

▶ **Theorem 11.** *With suitable parameter settings (as in Theorem 9), for the predecessor problem under* NOISY, *with* $0 < p < 1/2$ *and* $0 < Q < 1/2$, *Algorithm* ***WeightedBinary*** *returns a set* M *containing the target, taking at most* β *queries, where* $|M| \le \beta + 1$ *and*

$$\beta = \frac{1}{1 - H(p)} \left(\log_2 n + O(\sqrt{\log n \log Q^{-1}} \cdot \log \frac{\log n}{\log Q^{-1}}) + O(\log Q^{-1}) \right) .$$

The second stage of **NoS** can be conducted with an existing algorithm, e.g., **Feige** [17] whose query complexity is $O(\log(n/Q))/(1 - H(p))$, as shown in Algorithm 2.

Algorithm 2 Algorithm **NoS**.

```
1: function NoS(A, Q)
2:     Q ← Q/2                                          ▷ For the union bound
3:     r ← (1 − (1−2p)/(1+√(8(ln Q^{-1})/(ln n)))) /2
4:     L ← (r log_2 n)/(1 − H(r)) and Γ ← (1 − r)/r
5:     M ← WeightedBinary(A, L, Γ)
6:     Run Feige on (M, Q/2)
7:     return the target found
```

It first runs **WeightedBinary** on the input to obtain M, whose size is bounded by $O(\log_2 n/(1 - H(p))$. Second, it runs **Feige** [17] on M and obtain a target within $\left(O(\log \frac{\log n}{1-H(p)}) + O(\log Q^{-1})\right)/(1 - H(p)) = O(\log \log n + \log Q^{-1})/(1 - H(p))$ queries. A union bound on the failure probability and summing up these two costs lead to Theorem 2.

3.3 Proof of Corollary 3

To obtain the promised expected query complexity of **NoS**, we apply a familiar trick [4]. Observe that when $\log Q^{-1} \in o(\log n)$, the worst-case query complexity of **NoS** can be written as $\frac{1}{1-H(p)} \left(\log_2 n + o(\log n)\right)$. As a result, the "gap" (i.e., the difference) between this worst-case bound and the expected bound, in Expression (2), is $\frac{Q}{1-H(p)} \left(\log_2 n + o(\log n)\right)$.

We strengthen **NoS** as follows. On the one hand, when $Q \le \frac{1}{\log_2 n}$, this gap becomes at most $\frac{1}{1-H(p)}(1 + o(1)) \in o(\log n)$. In this case, the worst-case query complexity of **NoS** suffices to meet the bound $\frac{1-Q}{1-H(p)} \left(\log_2 n + o(\log n)\right)$. But when $Q > \frac{1}{\log_2 n}$, we perform the following steps. With probability $Q - \frac{1}{\log_2 n}$, we return index 1, and are done. Otherwise, i.e., with probability $1 - Q + \frac{1}{\log_2 n}$, we run **NoS** with a failure probability $Q' = \frac{1}{\log_2 n}$.

Clearly, the probability of not returning the target index by the above strengthened version of **NoS** is at most $Q - \frac{1}{\log_2 n} + Q' = Q$. Furthermore, in expectation, the query cost is $(1 - Q + \frac{1}{\log_2 n}) \cdot \frac{1}{1-H(p)} \left(\log_2 n + o(\log n)\right)$ which is bounded by $\frac{1-Q}{1-H(p)} \left(\log_2 n + o(\log n)\right)$.

4 Unbounded Search Without Noise

In this section we introduce an algorithm for the UNBOUNDED setting by Bentley and Yao [5] (denoted by **BY**), a central plank of our approach to UNBOUNDED. Bentley and Yao express their results, i.e., Lemmas 12, 13 and 14 via some customized functions. Our exposition has more standard notation, via the logarithmic function.

We consider UNBOUNDED for the index domain $A = \{1, 2, \ldots\}$, an infinite sequence. We first define a series of subsequences of A. Specifically, put $A^0 \equiv A$, and for $i, j \geq 1$ let $A^i[j] = A^{i-1}[2^j - 1]$. We illustrate the definition of the $\{A^i\}$ in Figure 1's left panel.

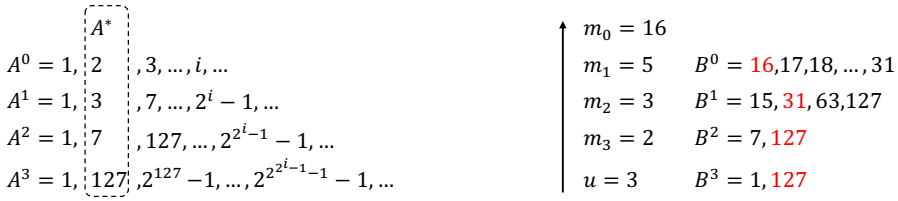

Figure 1 Left panel: An illustration of the first four subsequences, A_0, A_1, A_2, A_3, and A^*. Right panel: an example of running **BY** for target index $N = 16$, with u, m_i and B^i. The arrow indicates the execution order, and the red-colored item indicates the answer to the predecessor query. Quantity m_i is the index of the red-colored item in A^i.

For all $i \geq 0$, define m_i to be the answer to the predecessor query for t on A^i: the goal of UNBOUNDED is to find m_0. We can also express the value of m_i in terms of N, the target.

▶ **Lemma 12** (Derived from [5]). *For $1 \leq i \leq k$, where $k = \log^* N$, $m_i \leq \lfloor \log^{(i)} N \rfloor + 2$.*

Next, the unbounded sequence A^* consists of the *second* item in each sequence A^i. Formally, $A^*[j] = A^j[2]$, for $j \geq 1$. Let u denote the answer to the predecessor query on A^*.

▶ **Lemma 13** (Derived from [5]). *We have $u \leq k + 1$, where $k = \log^* N$.*

Algorithm 3 outlines the **BY** algorithm. It has two stages: first we find u via a unary search on A^*; then we repeatedly call **BinarySearch** on suitable subsequences, B^i. For some array R, **BinarySearch**(R) returns the lowest i satisfying $F(i) = 1$, in $\lceil \log |R| \rceil$ queries. Line 6 defines B^{i-1}, a subsequence of A^{i-1}, so that it contains all items that are between the immediate preceding item of the current search result, $A^i[m_i - 1]$, and the current search result, $A^i[m_i]$. Formally, for $1 \leq j < 2^{m_i - 1} + 1$, we have $B^{i-1}[j] = A^{i-1}[j + 2^{m_i - 1} - 1]$. The intuition of the algorithm is simple: at each iteration, we identify, via binary search, the answer to the predecessor problem for a subsequence of A^i, and the result helps us to "zoom in" to the items contained by an interval of a more fine-grained subsequence, A^{i-1}. The right panel of Figure 1 shows an example of running **BY**.

Algorithm 3 BY.

```
1: function BY(A)
2:     Find u by evaluating A* linearly
3:     B^u ← A^u
4:     for i ← u down to 1 do
5:         m_i ← BinarySearch(B^i)
6:         B^{i-1} ← the subsequence of A^{i-1} for indexes in [2^{m_i - 1}, 2^{m_i})
7:     m_0 ← BinarySearch(B^0)
8:     return m_0
```

▶ **Lemma 14** (Derived from [5]). *Let $k = \log^* N$. The query complexity of algorithm **BY** is $5 + 2k + \sum_{1 \leq i \leq k} \lfloor \log^{(i)}(N) \rfloor$.*

5 Unbounded Noisy Search

With the help of algorithms **BY** and **NoS**, we can tackle UnboundedNoisy. Algorithm **NoSU** builds on the idea of an *almost* result-sensitive algorithm [16, Section 3.1].

5.1 An Existing Algorithm

We first describe an existing UnboundedNoisy algorithm called **Unbounded**, derived from Noisy algorithm **Feige**.

▶ **Theorem 15** (Theorem C.3 [11]). *For the predecessor problem under UnboundedNoisy parameterized by $0 < p < 1/2$ and $0 < Q < 1/2$, Algorithm **Unbounded**, with probability at least $1 - Q$, correctly finds the target index. **Unbounded** is built upon an existing Noisy algorithm $\mathcal{A}$ and it uses at least 10 but at most $O(1)$ times as many queries as does $\mathcal{A}$. When $\mathcal{A}$ is **Feige** [17], **Unbounded** has query complexity $O\left(\frac{1}{1-H(p)} \cdot \log(N/Q)\right)$.*

The **Unbounded** algorithm invokes **Feige** as a blackbox and inherits the same asymptotic query complexity bound. Theorem 15 provides a satisfactory algorithm for solving UnboundedNoisy. In contrast, **NoSU** achieves a bound with leading term $\frac{1}{1-H(p)} \cdot \log_2(N/Q)$. The hidden constant leading coefficient in **Unbounded** is at least 10, i.e., the query complexity of algorithm **Unbounded** is not as tight as desired. The bound of Epa et al. [16] also includes a large constant, dervied from the Chernoff bound.

5.2 Algorithm NoSU

Algorithm 4 details our procedure, **NoSU**. Symbols m_i, u and B^i are the same as in **BY**, defined in Section 4. Similar to **BY**, **NoSU** has stages, one to figure out u, the other to simulate the iterations of B^i. We prove Theorem 5 here.

Algorithm 4 NoSU.

```
1: function NoSU(A, Q)
2:     run Unbounded(A*, Q/3) to find u
3:     B^u ← A^u
4:     for i from u to 1 do
5:         m_i ← NoS(B^i, Q/(3u))
6:         Let B^{i-1} be a subsequence of A^{i-1} for indexes in [2^{m_i - 1}, 2^{m_i})
7:     m_0 ← NoS(B^0, Q/3)
8:     return m_0
```

Proof of Theorem 5. Clearly, by the union bound, Algorithm 4 correctly finds the target index N with probability at least $1 - Q$.

We now focus on the query complexity of each of the two stages of the algorithm. Line 2 identifies u by calling **Unbounded**$(A^*, Q/3)$. Recall that $k = \log^* N$. Lemma 13 shows that the target index is u and $k + 1 \geq u$, so Theorem 15 implies that $O\left(\frac{1}{1-H(p)} \cdot \log(k/Q)\right)$ many queries are needed for this step.

For $1 \leq i \leq u$, let $g_i(N, p, Q)$ be the upper bound on the number of queries issued by algorithm **NoS** (Theorem 2) when applied to B^i. For $0 \leq i \leq u - 1$, the size of B^i at iteration i is $2^{m_{i+1}-1}$; and according to Lemma 12, for $1 \leq j \leq k$, $m_j \leq \lfloor \log^{(j)} N \rfloor + 2$.

Therefore, for $0 \le i \le u-1$, $|B^i| = 4\log^{(i)}(N) - 1$. Also, since $|B^u| = 2$, the cost of calling **NoS** on B^u is a constant. Summing over all $g_i(N,p,Q)$ terms yields

$$\sum_{i=0}^{u-1} g_i(N,p,Q) \le \sum_{i=0}^{k} g_i(N,p,Q) = \frac{1}{1-H(p)}\left(C_1 + C_2 + C_3 + C_4\right) ,$$

where $C_1 = \sum_{i=0}^{k} \log_2^{(i+1)} N$, $C_2 = \sum_{i=0}^{k} O(\log^{(i+2)} N)$, $C_3 =$

$$\sum_{i=1}^{k} O\left(\sqrt{\log^{(i+1)} N \log(3k/Q)} \cdot \log \frac{\log^{(i+1)} N}{\log(3k/Q)}\right) + O\left(\sqrt{\log N \log(3/Q)} \cdot \log \frac{\log N}{\log(3/Q)}\right) ,$$

and $C_4 = \sum_{i=1}^{k} O(\log(3k/Q)) + O(\log(3/Q))$.

We inspect each of the four terms individually. For the first term, C_1, we have simply $\sum_{i=1}^{k+1} \log_2^{(i)} N$, which is $\sum_{i=1}^{k} \log_2^{(i)} N$ plus a small constant. For the second term, C_2, the quantity folds into $O(\log\log N)$. For the third term, C_3, with large enough N, we have

$$\sum_{i=1}^{k} O(\sqrt{\log^{(i+1)} N \log(3k/Q)}) + O(\sqrt{\log N \log(3/Q)}) \le O(\sqrt{\log N \log Q^{-1}}) .$$

And for $1 \le i \le k$ the multiplicative factor of C_3 satisfies $\log \frac{\log^{(i+1)} N}{\log(3k/Q)} \le \log \frac{\log N}{\log Q^{-1}}$, and $\log \frac{\log N}{\log(3/Q)} \le \log \frac{\log N}{\log Q^{-1}}$. The fourth term, C_4, adds up to $O(k\log(k/Q))$, and it absorbs the cost of **Unbounded**. We thus obtain the bound of Theorem 5. ◀

Although tight, the upper bound in Theorem 5 does not immediately provide a clear perspective on the bound. Consider the worst-case query complexity of **NoSU**. In our setting, we have $\sum_{i=1}^{k} \log_2^{(i)} N + O(k\log k) < \log_2 N + O(\log\log N)$, so the Theorem 5 bound is at most

$$\frac{1}{1-H(p)}\left(\log_2 N + O(\log\log N) + O(\sqrt{\log N \log Q^{-1}} \log \frac{\log N}{\log Q^{-1}}) + O(k\log Q^{-1})\right) .$$

This is remarkably similar to the bound in Theorem 2. We now prove Corollary 6.

Proof of Corollary 6. Recall that **Unbounded** requires $O\left((1-H(p))^{-1}\log(k/Q)\right)$ many queries, which resolves to $o(\log N)$ under our setting. For $i \in [0,k]$, the call of **NoS** on B^i only results at most $\log_2^{(i+1)} N + o(\log^{(i+1)} N)$ queries, as $|B^i| = 4\log^{(i)} N - 1$. The cost of searching in B^u is again a small constant. Corollary 3 hence confirms **NoSU** has expected query complexity $(1-Q)/(1-H(p)) \sum_{i=1}^{k} \log_2^{(i)} N + o(\log N)$. ◀

6 The Lower Bound

This section is dedicated to prove Theorem 7, a lower bound on the expected query complexity for the predecessor problem under UnboundedNoisy. Let $\bar{\gamma}$ be the expected query complexity of an arbitrary algorithm $\mathcal{A}$ that solves the predecessor problem under UnboundedNoisy, parameterized by $0 < p < 1/2$ and $0 < Q < 1/2$. We lower bound $\bar{\gamma}$.

The idea of the lower bound proof is to reduce to the predecessor problem under UnboundedNoisy, from a well-studied task in information theory. The proof for the Noisy setting by Ben-Or and Hassidim [4] is not directly applicable.

First we introduce some basic concepts and terminologies in information theory. In a classic communication problem, we have a transmitter, A, who wants to send some information to a receiver, B. The information is from a *source* set S, and it is sent via a *communication*

channel where only special symbols are allowed to travel, one at a time. In our context, we focus on a *binary channel* that transmits only a binary string bit by bit. A *codeword* w from a binary alphabet is defined as a non-empty binary string, i.e., $w \in \{0,1\}^+$. A *code scheme* is an injective mapping $c : S \to \{0,1\}^+$ that maps an item in the source set S to its unique codeword. The communication channel we consider here is a *noisy binary symmetric memoryless channel with feedback* (BSM-F). The channel is *noisy* if for every bit sent from A to B, upon arrival, it could be flipped (from 1 to 0 or from 0 to 1) independently with certain probability. A channel is *symmetric* if the probability of flipping is the same for the bit 1 and 0, and *memoryless* if the probability also does not vary throughout the transmission, independent to the transmission history before that bit. A communication channel has a *feedback channel* if for every bit A sends, A knows what B receives from the feedback (sometimes called backward) channel. The feedback is useful here as A is adaptive to the noise and can make a decision on the next bit to transmit accordingly.
From the full version of the paper by Ben-Or and Hassidim [4] we have:

▶ **Fact 16** (Theorem B.1 of full version [4]). *For a fixed flipping probability $0 < p < 1/2$ and a failure probability $0 < Q < 1/2$, consider the communication problem where A sends a message to B over a BSM-F. A bit is received wrongly with probability p, and the message is of length η. Besides, the codeword is correctly received with probability at least $1 - Q$. Let $\tau(\eta)$ be the expected number of bits A has to send to achieve the goal. Then for a large enough η, we have $\tau(\eta) > (1 - Q)\frac{\eta}{1-H(p)} - 10$.*

Define a communication task *Comm* that requires A to send to B the information, an item of $S = \{1, 2, \dots\}$, over a BSM-F. This naturally induces an UnboundedNoisy instance with error probability p and failure probability Q. Denote by $\mathcal{A}$ an algorithm that solves the predecessor problem under UnboundedNoisy. We invoke $\mathcal{A}$ to solve *Comm*, and the lower bound on *Comm* implies a lower bound on the query complexity of $\mathcal{A}$.

The following construction of the communication protocol between A and B is inspired by the proof of Theorem 2.8 by Ben-Or and Hassidim [4]. B employs Algorithm $\mathcal{A}$ to identify the index of the item in S (which is equivalent to the item itself in this case) that A wants to send; and B also uses the feedback channel to inform A which index it wants to query. Specifically, A sends back a bit 0 if $F(i) = -1$, and a bit 1 if $F(i) = 1$ via the noisy channel to B. The communication is terminated once Algorithm $\mathcal{A}$ decides that the target index is found, i.e., B knows the target item in S. In the above definition of the feedback channel, A only knows the bit B received from the last transmission, i.e., the noisy answer for the latest query. However, we can achieve the same effect by simulating algorithm $\mathcal{A}$ on A as well as on B. As both A and B know the same queried answer, each copy of the algorithm yields the same steps. Conceptually, we can think of the feedback channel is capable of requesting a specific item index to inquire.

As alluded to earlier, the channel handles not directly the index N of the item but a binary code, which we refer to as the codeword of N. Algorithm $\mathcal{A}$ defines a code scheme $c : S \to \{0,1\}^+$, where $c(N)$ is the codeword for the index N transmitted in the channel when none of the bits of $c(N)$ is flipped by the noise. Intuitively, when B receives from A successfully the information of N, $c(N)$ is the message that B actually *discovers* from the transmission. B may not receive exactly the bits in $c(N)$ but after correcting errors caused by the channel noise, B should be able to conclude that the right codeword is $c(N)$.

We define a code scheme to connect *Comm* to Theorem 16. Let $\eta_N = \big|c(N)\big|$ be the number of bits of $c(N)$. Crucially, as observed by Bentley and Yao [5], the code scheme c determines a prefix code, i.e., $c(i)$ is not a prefix of $c(j)$ for all $i < j$. This is also easy to

verify, because if for some i, $c(i)$ is a prefix of another codeword, then algorithm $\mathcal{A}$ will not know whether to terminate or not when $c(i)$ arrives. They proved the following lower bound on the message length of a prefix-code codeword.

▶ **Fact 17** (Theorem A, section 3 of [5]). *Let $k = \log^* N$. We have $\eta_N > \sum_{i=1}^{k} \log_2^{(i)} N - 2k$.*

Our lower bound result (Theorem 7) follows from Fact 16 and Fact 17.

7 Conclusion

In this work, we provide improved algorithms for both NOISY and UNBOUNDEDNOISY settings of predecessor search. For the former, our algorithm **NoS** achieves, in the leading term, query complexity $(\log n)(1-H(p))^{-1}$ and expected complexity $((1-Q)\log n)(1-H(p))^{-1}$. For the latter, our algorithm **NoSU** achieves, in the leading term, query complexity $\sum_{i=1}^{k}(\log N)(1-H(p))^{-1}$ and expected complexity $(1-Q)(\sum_{i=1}^{k}\log N)(1-H(p))^{-1}$. Our expected upper bounds also closely match lower bounds. We construct **NoSU** by creatively combining results from the UNBOUNDED and NOISY settings. Our result emphasizes obtaining the best constant on the leading term, particularly in the UNBOUNDEDNOISY setting.

Since our **NoS** algorithm is derived from algorithms designed for graph searches, it would be interesting in future to explore the unbounded setting in graphs and test if our idea can be generalized to more general graphs as well. The error model assumes here that the probability of the oracle making a mistake is the same for all queries. In a more flexible setting, the error probability follows a different distribution, rather than the uniform distribution. As others found [16, 17], our algorithm could potentially be applied as a subroutine to many more problems that rely on the predecessor problem.

References

1 Javed A Aslam and Aditi Dhagat. Searching in the presence of linearly bounded errors. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 486–493, 1991.

2 Paul Beame and Faith E Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002.

3 Richard Beigel. Unbounded searching algorithms. *SIAM Journal on Computing*, 19(3):522–537, 1990.

4 Michael Ben-Or and Avinatan Hassidim. The Bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *Proc. 49th IEEE Symposium on Foundations of Computer Science*, pages 221–230, 2008.

5 Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(SLAC-PUB-1679), 1976.

6 Ryan S Borgstrom and S Rao Kosaraju. Comparison-based search in the presence of errors. In *Proc. 25th ACM Symposium on Theory of Computing*, pages 130–136, 1993.

7 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms: Reliable Computation with Unreliable Information*. Springer, 2013.

8 Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In *Advances in Neural Information Processing Systems*, volume 17, pages 337–344, 2005.

9 Christian Deppe. Coding with feedback and searching with lies. In Imre Csiszár, Gyula OH Katona, and Gábor Tardos, editors, *Entropy, Search, Complexity*, pages 27–70. Springer, 2007.

10 Dariusz Dereniowski, Aleksander Łukasiewicz, and Przemysław Uznański. Noisy searching: simple, fast and correct. *arXiv preprint*, 2021. `arXiv:2107.05753`.

11 Dariusz Dereniowski, Stefan Tiegel, Przemysław Uznański, and Daniel Wolleb-Graf. A framework for searching in graphs in the presence of errors. *arXiv preprint*, 2018. `arXiv:1804.02075`.

12 Dariusz Dereniowski, Stefan Tiegel, Przemyslaw Uznanski, and Daniel Wolleb-Graf. A framework for searching in graphs in the presence of errors. In *Proc. 2nd Symposium on Simplicity in Algorithms*, volume 69, pages 4:1–4:17, 2019.

13 Aditi Dhagat, Peter Gács, and Peter Winkler. On playing "twenty questions" with a liar. In *Proc. 3rd ACM/SIAM Symposium on Discrete Algorithms*, volume 92, pages 16–22, 1992.

14 Ehsan Emamjomeh-Zadeh and David Kempe. A general framework for robust interactive learning. In *Proc. 31st International Conference on Neural Information Processing Systems*, pages 7082–7091, 2017.

15 Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In *Proc. 48th ACM Symposium on Theory of Computing*, pages 519–532, 2016.

16 Narthana S Epa, Junhao Gan, and Anthony Wirth. Result-sensitive binary search with noisy information. In *Proc. 30th International Symposium on Algorithms and Computation*, pages 60:1–60:15, 2019.

17 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.

18 Daniel Golovin, Andreas Krause, and Debajyoti Ray. Near-optimal bayesian active learning with noisy observations. In *Advances in Neural Information Processing Systems*, volume 23, pages 766–774, 2010.

19 Richard M Karp and Robert Kleinberg. Noisy binary search and its applications. In *Proc. 18th ACM-SIAM Symposium on Discrete Algorithms*, pages 881–890, 2007.

20 Yong Sin Kim and Roland Winston. Unbounded binary search for a fast and accurate maximum power point tracking. In *AIP Conference Proceedings*, volume 1407, pages 289–292, 2011.

21 Amos Korman, Jean-Sébastien Sereni, and Laurent Viennot. Toward more localized local algorithms: removing assumptions concerning global knowledge. *Distributed Computing*, 26(5-6):289–308, 2013.

22 Robert Nowak. Generalized binary search. In *Proc. 46th IEEE Allerton Conference on Communication, Control, and Computing*, pages 568–574, 2008.

23 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM Symposium on Theory of Computing*, pages 232–240, 2006.

24 Andrzej Pelc. Searching with known error probability. *Theoretical Computer Science*, 63(2):185–202, 1989.

25 Andrzej Pelc. Searching games with errors – fifty years of coping with liars. *Theoretical Computer Science*, 270(1-2):71–109, 2002.

26 Alfréd Rényi. On a problem of information theory. *MTA Mat. Kut. Int. Kozl. B*, 6(MR143666):505–516, 1961.

27 Alfréd Rényi. On the foundations of information theory. *Revue de l'Institut International de Statistique*, pages 1–14, 1965.

28 Alfréd Rényi and Zsuzsanna Makkai-Bencsáth. *A Diary on Information Theory*. Akadémiai Kiadó Budapest, 1984.

29 Ronald L. Rivest, Albert R. Meyer, Daniel J. Kleitman, Karl Winklmann, and Joel Spencer. Coping with errors in binary search procedures. *Journal of Computer and System Sciences*, 20(3):396–404, 1980.

30 Michael Saks and Avi Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees. In *Proc. 27th IEEE Symposium on Foundations of Computer Science*, pages 29–38, 1986.

31 Pranab Sen. Lower bounds for predecessor searching in the cell probe model. In *Proc. 18th IEEE Conference on Computational Complexity*, pages 73–83, 2003.

32 Sanjay Kumar Singh and Amarjeet Singh. *Software testing*. Vandana Publications, 2012.

33 John A Stankovic. Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1):3–9, 2014.

34 Stanislaw M Ulam. *Adventures of a Mathematician*. University of California Press, 1991.