Algorithmica (2002) 32: 641–650
DOI: 10.1007/s00453-001-0097-4

Algorithmica

Delayed Binary Search, or Playing Twenty Questions with a Procrastinator[1]

A. Ambainis,[2] S. A. Bloch,[3] and D. L. Schweizer[4]

Abstract. We study the classic binary search problem, with a delay between query and answer. For all constant delays, we give matching upper and lower bounds on the number of queries.

Key Words. Binary search, Delay, Fibonacci series, Golden Ratio, Monotone search.

1. Introduction and Definitions. Consider the problem, familiar to teachers, of matching the difficulty of homework assignments to the ability of a group of students. Assuming one has some reasonable measure of homework difficulty, an obvious algorithm is binary search: take a range of possible difficulty levels, give an assignment that lies in the middle, and then narrow the range to its upper or lower half depending on whether the students' performance indicates the assignment was too easy or too hard.

The realities of scheduling, however, often require that the next assignment be given before the previous one is graded (more generally, before the previous d assignments are graded). Trivially, this slows down the task of finding the correct level of difficulty by at most a factor of $d + 1$: simply give the same assignment $d + 1$ times in a row, ignoring all but the answers to the first assignment. (This is roughly equivalent to giving homework only every $d + 1$ weeks, which some of the students would no doubt prefer.) The question remains: is this slowdown of $d + 1$ optimal? It is not, and in this paper we show that the optimal slowdown is in fact $\log_{\varphi_d} 2$, where φ_d is the positive real root of $x^{d+1} = x^d + 1$.

We now define the problem precisely.

DEFINITION 1. The search problem with delay d and size r consists of the following:
Instance: A real number $\Delta > 0$ and an unknown real number $x \in [0, r)$.
Output: A pair of real numbers (a', b') such that

- $0 \le a' < b' \le r$,
- $b' - a' \le \Delta$, and
- $x \in [a', b')$.

[1] With apologies to [1].

[2] Computer Science Division, University of California, Berkeley, CA 94720, USA. ambainis@cs.berkeley.edu.
[3] Department of Mathematics and Computer Science, Adelphi University, Garden City, NY 11530, USA. sbloch@adelphi.edu.
[4] Barclays Global Investors, 45 Fremont Street, San Francisco, CA 94105, USA. dls@aya.yale.edu.

Received May 28, 1999; revised July 10, 2000. Communicated by C. H. Papadimitriou.
Online publication January 7, 2002.

Allowed operations: A sequence of queries $x_1, x_2, \ldots$, each a real number signifying the question "Is $x \geq x_i$?" For all i, the choice of x_i may depend only on the answers to queries $1, 2, \ldots, i-d-1$, and not the answer to query $i-d$ or later (in particular, the first $d+1$ queries must be made before *any* answers are available). Unlimited computation is allowed between queries.

Alternatively, the problem can be formulated as finding (within precision Δ) the zero of an unknown continuous function f, under the assumptions that $f(0) \leq 0 < f(n)$ and that f is monotone on $[0, n]$, by querying an oracle that tells, for a given x_i, whether $f(x_i) \geq 0$. This formulation provides a convenient analogy to the unimodal maximization problem discussed below.

As the title suggests, our problem is related to Ulam's famous question [2] about finding a number between one and a million by querying an opponent who may lie once or twice. Various specializations and generalizations of Ulam's problem have been studied (e.g., [3]–[5], [1], and [6]), including exact and asymptotic analyses, relations to error-correcting codes, and variations on the type of query allowed. None of this work has examined delay.

Our problem is also related to finding the maximum of a unimodal function. The first solutions [7]–[9], in which Fibonacci searching was developed, were extended to parallel searching [9], [10] and to searching with delay [11], [12]. More recently, this work has been extended to different searching rules [13] and generalized to k-modal function optimization (in which the function's kth derivative is assumed to have a unique zero; see [14]). Although the two problems are related, we will show that they are not equivalent.

DEFINITION 2. The unimodal maximization problem with delay d and size n consists of the following:

Instance: A real number $\Delta > 0$ and an unknown function $f\colon \mathbf{R} \to \mathbf{R}$ that is unimodal on $[0, n]$ (i.e., for some $c \in [0, n]$, f is increasing on $[0, c)$ and decreasing on $(c, n]$).

Output: A pair of real numbers a', b' such that

- $0 \leq a' < b' \leq n$,
- $b' - a' \leq \Delta$, and
- $a' \leq c \leq b'$, where c is the extremum described above.

Allowed operations: A sequence of queries $x_1, x_2, \ldots$, each a real number signifying the question "What is $f(x_i)$?" For all i, the choice of x_i may depend only on the answers to queries $1, 2, \ldots, i-d-1$, and not the answer to query $i-d$ or later (in particular, the first $d+1$ queries must be made before *any* answers are available). Unlimited computation is allowed between queries.

The unimodal optimization problem with delays $d = 1, 2$ (as well as a "lattice" version of the problem, in which all queries must be integers and the output is an integer that maximizes f) was studied by Beamer and Wilde [11], and extended to $d = 3$ by Li [12]. They showed that $\log_{\psi_d} n + o(\log_{\psi_d} n)$ queries are necessary and sufficient,

where

- $\psi_0 = (\sqrt{5}+1)/2 \approx 1.6180$,
- $\psi_1 = \sqrt{2} \approx 1.4145$,
- $\psi_2 \approx 1.325$, and
- $\psi_3 \approx 1.2786$.

These investigations used complicated case-by-case analysis; there is no obvious way of extending them to general d (or even to $d = 4$).

The search problem (reformulated in terms of a monotone function) reduces easily to the unimodal maximization problem: simply maximize the function $-f(x)^2$. However, the converse does not hold, and it is obvious that unimodal optimization requires more queries than search in the delay-0 case ($\log_{1.618\ldots}(n)$ versus $\log_2(n)$). Our results together with [11] and [12] imply that the same holds for delays 1, 2, and 3.

Our delay-d optimal algorithm searches the interval $[0, n)$ with $\log_{\varphi_d} n + O(1)$ queries. Recall that φ_d is the positive real root of $x^{d+1} = x^d + 1$; for example,

- $\varphi_0 = 2$ (indeed, binary search is the $d = 0$ case of our algorithm),
- $\varphi_1 = (\sqrt{5}+1)/2 \approx 1.6180$,
- $\varphi_2 \approx 1.4656$, and
- $\varphi_3 \approx 1.3803$.

2. Main Results. Our first approach to the problem was to use dynamic programming to construct a table of the minimum number of queries necessary to solve a discrete version of the problem on search spaces of various integral sizes with various pending queries. We then looked for patterns in the table, proved them, and converted them into recurrence relations. This program had been carried out for the cases of delay 1 and 2, and we were working on delay 3 (the tables are $(d+1)$-dimensional, so it becomes progressively harder to find, state, and prove the patterns), before we managed to generalize the results. While this approach generated a lot of pretty patterns and lemmas, we omit them in favor of the more concise general solution.

For simplicity, we assume henceforth that $\Delta = 1$ (equivalently, we scale the search space $[0, r)$ by a factor of $1/\Delta$).

DEFINITION 3. For any nonnegative integers d and t, let $A_d(t)$ denote the largest natural number such that the search problem with delay d on $[0, A_d(t))$ can be solved (to within $\Delta = 1$) with t queries.

DEFINITION 4. For any fixed nonnegative integer d and any integer t, let

$$B_d(t) = \begin{cases} 1, & \text{if} \quad t \le 0; \\ B_d(t-1) + B_d(t-d-1), & \text{if} \quad t > 0. \end{cases}$$

(For example, for any t, $B_0(t) = 2^t$, and $B_1(t)$ is the $(t+2)$nd Fibonacci number.)

Our main result is

THEOREM 5. *For all $d \geq 0$ and $t \geq 0$,*

$$A_d(t) = B_d(t).$$

Once this is established, standard techniques for solving recurrences yield the following corollaries:

COROLLARY 6.

$$A_d(t) \in \Theta(\varphi_d^t).$$

COROLLARY 7. *For all nonnegative integers d,*

$$\log_{\varphi_d}(n) + O(1)$$

queries are necessary and sufficient for the search problem of size n with delay d.

3. Proof of the Main Theorem. We begin with an easy observation:

PROPOSITION 8. *If $d \geq t$, then*

$$A_d(t) = B_d(t) = t + 1.$$

PROOF. If $t \leq d$, then all queries must be made before any answers are received. Thus, in this situation, all queries are nonadaptive and so we can do no better with our t queries than to query $1, 2, \ldots, t$. These suffice to search the interval $[0, t + 1)$.

On the other hand, $B_d(t) = B_d(t-1) + B_d(t-d-1)$. The latter term is 1 because $t - d - 1 < 0$, so by a straightforward induction, $B_d(t) = t + 1$. □

3.1. *The Upper Bound*

THEOREM 9. *For all $d \geq 0$, $t \geq 0$,*

$$A_d(t) \geq B_d(t).$$

PROOF. We establish the theorem by exhibiting an algorithm that solves the search problem with delay d on the interval $[0, B_d(t))$ with t queries. For ease of exposition, we write this algorithm with a queue of queries. Each query is placed onto the end of the queue; the dequeue operation returns both the value of the query it removed from the head of the queue and the answer to that query. The delay is enforced simply by initially seeding the queue with d uninformative queries.

Algorithm

```
1   read(d, t)
2   for i := 1 to d do
3   enqueue(0)
4   od
5   lower := 0
6   upper := B_d(t)
7   pending := 0
8   enqueue(B_d(t − 1))
9   while (upper−lower> 1) do
10        (query, result) := dequeue
11        if (query ≤ lower)
12          then pending := pending + 1
13           else if (result = "yes")
14                     then lower := query
15                          pending := 0
16                     else upper := query
17                fi
18        fi
19        enqueue(lower +(B_d(B_d^{-1}(upper−lower)−pending−1)))
20  od
```

Lines 1–4 simply initialize the algorithm. Since the search interval is $[0, B_d(t))$, the d initial repetitions of the query "Is $x \geq 0$?" must all return "yes," and hence convey no information. Lines 5–7 initialize the search interval and the counter of pending potentially informative queries. Line 8 places the first useful query onto the queue. Intuitively, this query divides the search interval unevenly (except in the degenerate case $d = 0$) in preparation for planting queries in the longer subinterval in a fashion that guarantees that, when they are answered, either the planted queries will be useful or the shorter interval will be compensatingly easy to search. As the algorithm retrieves answers to queries, there are three possibilities:

- Line 11: the query lies below the lower end of the current search interval and is thus useless. All the algorithm can do is plant another pending query in the current search interval.
- Line 13: the query is in the current search interval and lies below the value sought. Since all further pending queries lie below this one (a consequence, not necessarily obvious, of line 19), they immediately become useless and the algorithm starts over on the (significantly) reduced interval.
- Line 16: the query is in the current search interval and lies above the value sought. The algorithm will be able to use its previously planted queries and simply continues on the (somewhat) reduced interval.

Line 19 computes the next query. Two observations are worthwhile: first, that since $B_d(t)$ is monotonically increasing in t (for fixed d), pending queries are planted in decreasing order. Second, that the length of the current search interval is invariantly maintained as a value for which B_d^{-1} is well defined.

Verification: If $t \leq d$, then $B_d(t) = t + 1$ and the algorithm just plants queries at all integer points $1, 2, \ldots, t$.

If $t > d$, the algorithm plants $d+1$ queries at points $B_d(t-1)$, $B_d(t-2)$, $\ldots$, $B_d(t-d-1)$. After planting these queries, the algorithm receives the first useful answer, the answer to "Is $x \geq B_d(t-1)$?" There are two cases:

- If the answer is "yes," then x is in the interval $[B_d(t-1), B_d(t))$. The size of this interval is $B_d(t) - B_d(t-1) = B_d(t-d-1)$; by the induction hypothesis, it can be searched with the remaining $t-d-1$ queries.
- If the answer is "no," then x is in $[0, B_d(t-1))$.
  - — If $t - 1 = d$, then, by Proposition 8, the algorithm is searching the interval $[0, d)$ with pending queries to $d-1, d-2, \ldots, 1$, which can be done with no further queries.
  - — On the other hand, if $t - 1 > d$, the algorithm plants the $(d+2)$nd query at the point $B_d(t-d-2)$. Now we are in a situation similar to the one we were in before: we have the interval $[0, B_d(t-1))$, $t-1$ queries, and we have asked $d+1$ of them at the points $B_d(t-2)$, $B_d(t-3)$, $\ldots$, $B_d(t-d-2)$. The result follows by induction. □

3.2. *The Lower Bound.* In order to prove that $A_d(t) \leq B_d(t)$, we introduce a generalization of each of these functions and then show that the result holds between the generalizations.

DEFINITION 10. For $1 \leq i_1 < i_2 < \cdots < i_j \leq \min(d, t)$, let $\bar{A}_d(t, \langle i_1, i_2, \ldots, i_j \rangle)$ denote the largest natural number such that the search problem with delay d on $[0, \bar{A}_d(t, \langle i_1, i_2, \ldots, i_j \rangle))$ can be solved (to within $\Delta = 1$) with t queries under the additional constraint that of the first d queries, only those numbered $i_1, i_2, \ldots, i_j$ may be used.

This definition captures the situation in which one or more pending queries have already been made useless by recently learned answers to previous queries.

We write $\vec{\imath}$ to denote $\langle i_1, i_2, \ldots, i_j \rangle$ and $\overrightarrow{\imath - 1}$ to denote $\langle i_1 - 1, i_2 - 1, \ldots, i_j - 1 \rangle$.

DEFINITION 11. For $\vec{\imath}$ as in Definition 10, let

$$\bar{B}_d(t, \vec{\imath}) = B_d(t-d) + \sum_{i \in \vec{\imath}} B_d(t - i - d).$$

PROPOSITION 12. *If $t \leq d$, then*

$$\bar{A}_d(t, \vec{\imath}) = \bar{B}_d(t, \vec{\imath}) = j + 1.$$

PROOF. As in Proposition 8, all queries are nonadaptive, and so, analogously, the algorithm can do no better than to query $1, 2, \ldots, j$, which suffice only to search the interval $[0, j+1)$.

Also similarly to Proposition 8,

$$\begin{aligned}\bar{B}_d(t, \vec{\imath}\,) &= B_d(t-d) + \sum_{i \in \vec{\imath}} B_d(t-i-d) \\ &= 1 + \sum_{i \in \vec{\imath}} 1 \\ &= 1 + j.\end{aligned}$$

□

PROPOSITION 13. *If $t \geq d$, then*

1. $A_d(t) = \bar{A}_d(t, \langle 1, 2, \ldots, d\rangle)$, *and*
2. $B_d(t) = \bar{B}_d(t, \langle 1, 2, \ldots, d\rangle)$.

PROOF. The first claim is an immediate consequence of the definitions of $A_d(t)$ and $\bar{A}_d(t)$. The second claim is established algebraically thus:

$$\begin{aligned}B_d(t) &= B_d(t-1) + B_d(t-d-1) \\ &= (B_d(t-2) + B_d(t-d-2)) + B_d(t-d-1) \\ &\vdots \\ &= (\cdots((B_d(t-d) + B_d(t-d-d)) + B_d(t-d-(d-1))) + \cdots \\ &\quad + B_d(t-d-2)) + B_d(t-d-1) \\ &= B_d(t-d) + B_d(t-2d) + B_d(t-2d+1) + \cdots + B_d(t-d-1) \\ &= B_d(t-d) + \sum_{i=1}^{d} B_d(t-i-d) \\ &= \bar{B}_d(t, \langle 1, 2, \ldots, d\rangle).\end{aligned}$$

□

THEOREM 14. *For all $d \geq 0$ and $t \geq 0$,*

$$A_d(t) \leq B_d(t).$$

PROOF. We prove that for all $\vec{\imath}$ as in Definition 10, $\bar{A}_d(t, \vec{\imath}\,) \leq \bar{B}_d(t, \vec{\imath}\,)$. The theorem follows as a special case by Proposition 13.

Fix d and induct on t. Observe that if $t \leq d$, the result follows immediately from Proposition 12.

If $t > d$, assume that $\bar{A}_d(t-1, \vec{\imath}\,) \leq \bar{B}_d(t-1, \vec{\imath}\,)$ for all $\vec{\imath}$. We wish to show, for an arbitrary $\vec{\imath}$, that $\bar{A}_d(t, \vec{\imath}\,) \leq \bar{B}_d(t, \vec{\imath}\,)$. Since $t > d$, the algorithm can make a potentially useful query whose result will become available at time $d+1$. Let $\vec{\imath}'$ denote $\langle i_1, i_2, \ldots, i_j, d+1\rangle$ and i_1' denote the first element of $\vec{\imath}'$. (That is, $i_1' = i_1$ unless $j = 0$, in which case $i_1' = d+1$.)

Two cases arise in the analysis of $\bar{A}_d(t, \vec{\imath}')$:

1. $i'_1 > 1$. In this case the answer to the first query may not be used (i.e., conveys no useful information). The algorithm must mark time until the answer to the next query becomes available. That means that $\bar{A}_d(t, \vec{\imath}) = \bar{A}_d(t-1, \overrightarrow{\imath - 1}')$, and so, by the induction hypothesis, that $\bar{A}_d(t, \vec{\imath}) = \bar{B}_d(t-1, \overrightarrow{\imath - 1}')$. However,

$$\begin{aligned}
\bar{B}_d(t-1, \overrightarrow{\imath - 1}') &= \bar{B}_d(t-1, \langle i_1 - 1, i_2 - 1, \ldots, d\rangle) \\
&= B_d((t-1)-d) + \sum_{i \in \overrightarrow{\imath - 1}'} B_d((t-1) - i - d) \\
&= B_d((t-1)-d) + \sum_{i \in \vec{\imath}} B_d((t-1) - (i-1) - d) \\
&\quad + B_d((t-1) - d - d) \\
&= B_d(t-d-1) + B_d(t-2d-1) + \sum_{i \in \vec{\imath}} B_d(t-i-d) \\
&= B_d(t-d) + \sum_{i \in \vec{\imath}} B_d(t-i-d) \\
&= \bar{B}_d(t, \vec{\imath}).
\end{aligned}$$

   Note, incidentally, that this case cannot occur if $d = 0$.
2. $i'_1 = 1$. In this case the answer to the first query may be used (i.e., conveys useful information). The list of indices of remaining queries that may be used, $\langle i_2, i_3, \ldots, i_j, d+1\rangle$, can be partitioned into indices of queries that still may convey useful information if the first query returns "no" (*i.e.*, $x < x_1$—see Definition 1) and those that still may be useful if the first query returns "yes" (i.e., $x \geq x_1$). We denote these lists $\vec{l}$ and $\vec{r}$, respectively, because the queries lie to the left or the right of x_1. Note that

$$\vec{l} \cup \vec{r} \cup \{1\} = \vec{\imath}' = \vec{\imath} \cup \{d+1\}.$$

   The original interval of length $\bar{A}_d(t, \vec{\imath})$ is thus divided into two subintervals which, in order to guarantee completion in the time remaining, must have lengths at most $\bar{A}_d(t-1, \overrightarrow{l-1})$ and $\bar{A}_d(t-1, \overrightarrow{r-1})$, respectively. Thus

$$\bar{A}_d(t, \vec{\imath}) \leq \bar{A}_d(t-1, \overrightarrow{l-1}) + \bar{A}_d(t-1, \overrightarrow{r-1}).$$

   Applying the induction hypothesis to both terms of the right-hand side,

$$\begin{aligned}
\bar{A}_d(t, \vec{\imath}) &\leq \bar{B}_d(t-1, \overrightarrow{l-1}) + \bar{B}_d(t-1, \overrightarrow{r-1}) \\
&= B_d((t-1)-d) + \sum_{l \in \overrightarrow{l-1}} B_d((t-1) - l - d) \\
&\quad + B_d((t-1)-d) + \sum_{r \in \overrightarrow{r-1}} B_d((t-1) - r - d)
\end{aligned}$$

$$
\begin{aligned}
&= 2B_d(t-1-d) + \sum_{l\in\vec{l}} B_d((t-1)-(l-1)-d) \\
&\quad + \sum_{r\in\vec{r}} B_d((t-1)-(r-1)-d) \\
&= 2B_d(t-1-d) + \sum_{s\in\vec{l}\cup\vec{r}} B_d(t-s-d) \\
&= 2B_d(t-1-d) + \sum_{i\in\vec{i}'} B_d(t-i-d) - B_d(t-1-d) \\
&= B_d(t-1-d) + \sum_{i\in\vec{i}} B_d(t-i-d) + B_d(t-(d+1)-d) \\
&= B_d(t-1-d) + B_d(t-2d-1) + \sum_{i\in\vec{i}} B_d(t-i-d) \\
&= B_d(t-d) + \sum_{i\in\vec{i}} B_d(t-i-d) \\
&= \bar{B}_d(t,\vec{i}\,),
\end{aligned}
$$

which concludes the proof of Theorem 14 and hence of Theorem 5. □

4. Conclusions and Future Directions. We have given exactly matching upper and lower bounds on the number of queries needed to solve the search problem with any fixed delay $d \geq 0$.

For future research, it may be natural to consider variants of the problem that involve *nonconstant* delay: delay as a function of the clock, delay as a function of the queried value x_i, or stochastic delay. These problems, however, are almost certainly more involved.

The unimodal search problem studied in [11] and [12] remains unanalyzed for $d > 3$, but one might hope that it too has a clean, uniform solution, and that the dynamic programming approach described in Section 2 might again provide a bridge to the necessary insight. Indeed, one might study delayed versions of *any* computational problem whose difficulty is measured by counting queries, e.g., order statistics or even the Boolean complexity hierarchy.

A variant on delayed search is "block" or "parallel" search, in which some number k_1 of queries are made nonadaptively, all k_1 answers are received before making the next block of k_2 queries, and so on. Beamer and Wilde [15] gave an optimal algorithm for the case when $k_1, k_2, \ldots$ are fixed. Li [12] showed how to choose the best $k_1, k_2, \ldots$ if the total number of queries and the number of blocks are fixed. It remains to be seen how to unify this investigation with the study of delay: the ultimate goal would be a simple formula (and a matching uniform algorithm), parameterized by both delay and block sizes, giving the maximum size of an interval searchable within a given number of time steps.

Reingold points out that although the problem of "matching homework difficulty" described in the Introduction is in principle unbounded, our analysis is actually of a bounded version of the problem in which upper and lower bounds are known. Reingold

and coworkers [16], [17], [13] have analyzed unbounded unimodal search (in which query complexity is treated as a function not of the size of the search space, but of the eventual answer and the initial guess), and Beigel [18] surveys unbounded search algorithms in general. In private communication, Reingold suggests a link between k-modal search [14] and delay k, but we have not yet tried to analyze the unbounded search problem in the presence of delay.

Acknowledgments. The authors thank Bill Gasarch for his comments on drafts of the paper, Tanya Berger-Wolf for suggesting the unbounded angle, and the audience of the rump session at the 1998 Computational Complexity Conference for their comments, questions, and enthusiasm.

An abbreviated version of this paper appeared in the *Proceedings of SODA* 1999 [19].

References

[1] A. Dhagat, P. Gacs, and P. Winkler. On playing "twenty questions" with a liar. In G. Frederickson, editor, *Proceedings of the Third Annual ACM–SIAM Symposium on Discrete Algorithms*. ACM Press, New York, 1992.

[2] S. M. Ulam. *Adventures of a Mathematician*. Scribner's, New York, 1976.

[3] A. Pelc. Solution of Ulam's problem on searching with a lie. *Journal of Combinatorial Theory, Series A*, 44(1):129–140, January 1987.

[4] A. Pelc. Detecting errors in searching games. *Journal of Combinatorial Theory, Series A*, 51(1):43–54, May 1989.

[5] W. Guzicki. Ulam's searching game with two lies. *Journal of Combinatorial Theory, Series A*, 54(1):1–19, May 1990.

[6] M. Aigner. Searching with lies. *Journal of Combinatorial Theory, Series A*, 74(1):43–56, April 1996.

[7] J. Kiefer. Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, 4:502–505, 1953.

[8] L. T. Oliver and D. J. Wilde. Symmetrical sequential minmax search for a maximum. *Fibonacci Quarterly*, 2:169–175, 1964.

[9] M. Avriel and D. J. Wilde. Optimal search for a maximum with sequences of simultaneous function evaluations. *Management Science*, 12:722–731, 1966.

[10] R. M. Karp and W. L. Miranker. Parallel minimax search for a maximum. *Journal of Combinatorial Theory*, 4(1):19–35, January 1968.

[11] J. H. Beamer and D. J. Wilde. Time delay in minimax optimization of unimodal functions of one variable. *Management Science*, 15:528–538, 1969.

[12] Weixuan Li. *Optimal Sequential Block Search*. Volume 5 of R&E: Research and Exposition in Mathematics. Heldermann Verlag, Berlin, 1984.

[13] A. S. Goldstein and E. M. Reingold. A Fibonacci version of Kraft's inequality applied to discrete unimodal search. *SIAM Journal on Computing*, 22(4):751–777, August 1993.

[14] A. Mathur and E. M. Reingold. Generalized Kraft's inequality and discrete k-modal search. *SIAM Journal on Computing*, 25(2):420–447, April 1996.

[15] J. H. Beamer and D. J. Wilde. Minimax optimization of unimodal functions of one variable. *Management Science*, 16:529–541, 1969.

[16] E. M. Reingold and Xiaojun Shen. More nearly optimal algorithms for unbounded searching, part I: the finite case. *SIAM Journal on Computing*, 20(1):156–183, February 1991.

[17] E. M. Reingold and Xiaojun Shen. More nearly optimal algorithms for unbounded searching, part II: the infinite case. *SIAM Journal on Computing*, 20(1):184–208, February 1991.

[18] R. Beigel. Unbounded searching algorithms. *SIAM Journal on Computing*, 19(3):522–537, June 1990.

[19] A. Ambainis, S. Bloch, and D. Schweizer. Playing twenty questions with a procrastinator. In *Proceedings of the Tenth Annual ACM–SIAM Symposium on Discrete Algorithms*, pages S844–S845, January 1999.