LBL-20708
UC-32

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

Computing Division

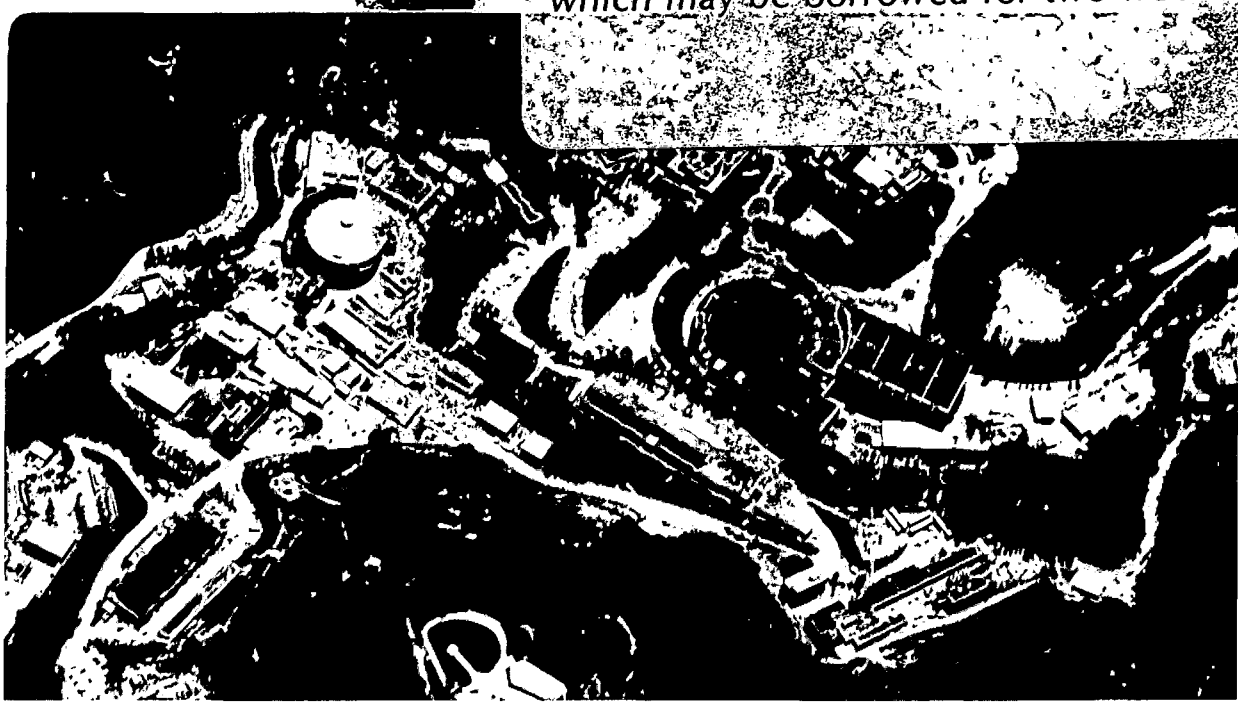
RECEIVED
LAWRENCE
BERKELEY LABORATORY
APR 7 1986
LIBRARY AND
DOCUMENTS SECTION

FURTHER RESULTS ON INTERPOLATION SEARCHING OF DATABASES

Z.J. Li and H.K.T. Wong

February 1986

TWO-WEEK LOAN COPY
This is a Library Circulating Copy
which may be borrowed for two weeks.



Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

LBL-20708 c.2

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

LBL-20708

Further Results on Interpolation Searching of Databases

Z.J. Li and H.K.T. Wong

Computer Science Research Department
University of California
Lawrence Berkeley Laboratory
Berkeley, California 94720

February, 1986

This research was supported by the Applied Mathematics Sciences Research Program of the Office of Energy Research, U.S. Department of Energy under contract DE-AC03-76SF00098.

LBL-20708

Further Results on Interpolation Searching of Databases

Z. J. LI* and Harry K.T. Wong
Lawrence Berkeley Laboratory,
University of California

Abstract

This paper extends known results of the Interpolation Search Algorithm on ordered tables in three ways. First, it examines the effect of batching the search queries. Second, it applies the basically main-memory algorithm to a more typical database environment, i.e. a blocked secondary memory. Third, it examines a hybrid algorithm to remedy the worst case behavior of the pure Interpolation Search in the event of non-uniform distribution of the ordered file while retaining the average complexity.

Algorithms, analytic expressions and experiment results of these extensions are given and described. Analytic expressions of these algorithms are validated by the experiments.

1. Introduction

Our interest in batched interpolation search comes from three separate search problems in statistical and scientific databases. The first problem involves the searching of data items in a compressed file. In particular, the compression is performed using a technique called header compression scheme [10]. The second problem is related to the searching of hierarchical relationship implemented in a file structure called hierarchical transposed file [11]. The third problem is the searching of data items in a sparse multi-dimensional data structure [12]. All three of these search problems can be reduced to batched interpolation search over ordered files.

The idea behind the Interpolation Search algorithm on ordered tables is simple and natural. An example will be used to illustrate the algorithm in action.

Given a table of 1,000 records with key $x_1 < x_2 < \ldots < x_{1000}$ uniformly distributed between 0 and 10,000. Our task is to find index i such that $x_i = 6{,}000$. It is reasonable to guess that about $\frac{6{,}000}{10{,}000} \times 1{,}000$ keys are less than or equal to x_i, and the required record should be near the 600th record. However, let us assume that x_{600} contains a key with value 5,800. The desired record should lie

Supported by the Office of Energy Research, U.S. DOE under Contract No. DE-AC03-76SF00098.

* On leave from Dept. of Computer Science, Heilongjiang Univ., China.

between 600th and $1{,}000^{th}$ records. We therefore take a second guess that x_{600} should be the $\frac{6{,}000-5{,}800}{10{,}000-5{,}800}\times(1{,}000-600)=19^{th}$ record of the new file. This process continues until the record is found.

First published by Peterson [5], the Interpolation Search Algorithm has received extensive attention. The major result is the $loglog\,(N)$[1] (where N is the number of keys in the table) complexity behavior of a single search [6-8,12]. These works, however, did not take the effect of batching search queries into consideration.

The advantages of batched searching on databases have been advocated by a number of researchers [1,2,3,4]. The major argument is that by batching searches or updates, the throughput of the system is increased and the potential reduction on processor demand may in fact reduce the response time.

The research on interpolation search cited above concentrates mainly on main-memory data structure and ignores the database secondary memory consideration. We are interested in adding block accesses as well as providing block accesses approximation expressions to the basic Interpolation Search algorithm, similar to [13].

The $loglog\,(N)$ behavior is guaranteed only in uniformly distributed keys in databases [2]. In this paper, a hybrid algorithm is given which combines binary search and interpolation search to remedy the worst case behavior ($O\,(N)$) of the Interpolation Search Algorithm [3].

The benefits of batched searches using Interpolation Search are analyzed in this paper. We will provide performance expressions of average behavior for both cases in terms of record accesses as well as block accesses, similar to [13].

The paper is organized as follows. In section 2, the analysis and the experiments of batched interpolation search algorithms in non-blocked and blocked environments are given. In section 3, two algorithms are described which are combination of binary search and interpolation search. Again, analysis and experiments are described for the algorithms in non-blocked and blocked environments. Section 4 summaries the paper.

---

[1] Throughout this paper, "log" designates base 2 logarithm.

[2] In [8,12], remarks were made to the effect that the same result is achievable on non-uniform distributions if the distribution function on the keys is known and used to map an initial non-uniform distribution onto an uniform distribution. This mapping, however, is typically expensive or impossible to attain for very large databases.

[3] A recent result on interpolation search [13] claims $loglog\,(N)$ complexity for non-uniform distributions. See the Appendix for comment on this result.

2. Batched Interpolation Search

2.1. Batched Interpolation Search Algorithm

Let $X=(x[1],x[2],\ldots\ldots,x[n])$ be an ordered file of uniformly distributed keys between a and b, where $x[j]<x[j+1](1\leq j\leq n-1)$. For expository reasons, we add the keys $x[0]=a$ and $x[n+1]=b$ as the first and last keys of the file.

Let $B=(\alpha_1,\alpha_2,\ldots\ldots,\alpha_k)$ be a ordered collection of search keys to be applied to file X, where $\alpha_i\,(1\leq i\leq k)$ is uniformly distributed keys between a and b, and $\alpha_i<\alpha_{i+1}(1\leq i\leq k-1)$, the algorithm BIS below will find an index j for each $\alpha_i\,(1\leq i\leq k)$ such that $x[j]=\alpha_i$ if such an index exists, otherwise, $x[j]<\alpha_i<x[j+1]$.

The idea behind algorithm BIS given below is that in searching file X for each element α_i in B, one can take advantage of the previous search for element α_i-1. Since both B and X are ordered, BIS can start the search for α_i at the place of X where α_{i-1} was found. The savings of batched searching are achieved because the size of file X is monotonically decreasing.

2.1.1. Algorithm

ALGORITHM BIS

input: $X=(x[1],x[2],\ldots\ldots,x[n])$, $x[0]=a$, $x[n+1]=b$,
$B=(\alpha_1,\alpha_2,\ldots\ldots,\alpha_k)$.
output: $seach=(search[1],search[2],\ldots\ldots,search[k])$ (the required indices).

```
(1) L :=0;H :=n +1;i =1;
(2) X_L :=x [0];X_H :=x [n +1];
(3) while i ≤k do
(4)       begin
              N :=H -L -1;
              J :=L +N · ⌈ (α_i -X_L)/(X_H -X_L) ⌉;
(5)           while N >0 and x [J ]≠α_i do
(6)               begin
(7)                 if x [J ]<α_i then
(8)                   begin L :=J ;X_L :=x [J ] end
(9)                 else
(10)                  begin H :=J ;X_H :=x [J ]; end
(11)              N :=H -L -1;
                  J :=L +N · ⌈ (α_i -X_L)/(X_H -X_L) ⌉;
(12)              end
```

```
(13)        if N ≤ 0 then
(14)           begin
                  search [i]:=L ;H :=n +1;

                 X_H :=x [n +1];
               end
(15)        else
(16)           begin
                search [i]:=J ;L :=J ;H :=n +1;
                 X_L :=x [J];X_H :=x [n +1];
               end

(17)       i :=i +1;
(18)     end
```

2.1.2. Analysis

The variables L and H represent lower and upper indices of the file searched respectively. L is continuously set to the index of the last found key. Step(3) of BIS begins the searching of each element in B. For each i $(1 \leq i \leq k)$ α_i is searched by step(4) to step(16) in the algorithm BIS in subfile $F_i = (x[L], \ldots\ldots, x[n+1])$, where $L = 0$ at the beginning. In order to find the desired index, BIS chooses an index $J = L + (H - L - 1) \cdot \frac{\alpha_i - x[L]}{x[H] - x[L]}$, with $H = n + 1$ at the beginning. The number of keys remaining in file X at the beginning of the search for α_i is given by the following lemma. First, we define an *iteration* of BIS to be the execution of step(3) to step(18).

LEMMA 1. The ith iteration in the algorithm BIS is to search α_i in the subfile $(x[r], \ldots\ldots, x[n+1])$, where $r = \frac{n \cdot (i-1)}{k+1}$.

Proof. When $i = 1$, $r = 0$ and the lemma is true. Let $i \geq 2$. From step (14) and step (16) in BIS, the ith iteration must be to search α_i in subfile $(x[r], \ldots\ldots, x[n+1])$, where r is such that $x[r] = \alpha_{i-1}$ or $x[r] < \alpha_{i-1} < x[r+1]$. Since the keys in X and B are uniformly distributed, the k elements of B divide X into $k+1$ subsets. Hence, α_{i-1} should be found on the average at location $r = \frac{n \cdot (i-1)}{k+1}$ of X. Thus, the lemma is proved. Q.E.D.

The behavior of BIS is summarized by the following theorem. It shows that the behavior is still $O(loglog(N))$, but n is reduced by a term proportional to N. The savings gained in practice are discussed in the next section.

THEOREM 1. Let $X=(x[1],x[2],\ldots\ldots,x[n])$ be an ordered file of uniformly distributed keys between a and b, where $x[j]<x[j+1](1\leq j\leq n-1)$, and $B=(\alpha_1,\alpha_2,\ldots\ldots,\alpha_k)$ be uniformly distributed keys between a and b, where $\alpha_i<\alpha_{i+1}$ $(1\leq i\leq k-1)$. The average number of record accesses required by algorithm BIS for searching B is less than

$$\sum_{i=1}^{k} loglog\,(n-\frac{n\cdot(i-1)}{k+1}).$$

Proof. Let us consider the ith iteration. From lemma 1, the ith iteration is to search α_i $(1\leq i\leq k)$ in subfile $F_i=(x[r],\ldots\ldots,x[n+1])$, and $r=\frac{n\cdot(i-1)}{k+1}$, that is, the number of unchecked keys in file F_i is $n-\frac{n\cdot(i-1)}{k+1}$. The ith iteration is interpolation search for one record. By Perl et.al.'s theorem 2 [8], the average number of record accesses in i^{th} iteration is less than $loglog\,(n-\frac{n\cdot(i-1)}{k+1})$. Thus, it follows that the average number of record accesses for searching B is less than

$$\sum_{i=1}^{k} loglog\,(n-\frac{n\cdot(i-1)}{k+1}).$$

Q.E.D.

2.1.3. Experimental Results

To validate theorem 1 experimently, we generated 6 sorted files of uniformly distributed random integers between 0 and 2^{31}. 1,000 sets of batched records are also generated with integers uniformly distributed between 0 and 2^{31} with size k for $k=1$ to 20. Table 1 contains the results of comparing the theoretical result of theorem 1 (the T.R. column) and the experimental result of executing algorithm BIS (the E.R. column). As can be seen, theorem 1 provides a good approximation to the behavior of algorithm BIS.

Figure 1 shows the savings by executing batched interpolation search algorithms on a file of 400,000 uniformly distributed integers other than by executing unbatched interpolation search algorithm. The savings due to the batching of queries over the unbatched interpolation search are roughly 40% when $k>20$.

2.2. Blocked Batched Interpolation Search

In this section, algorithm BIS is modified to take blocking into consideration. The analysis following the algorithm provides block access approximation.

2.2.1. Algorithm

The algorithm BBIS below is similar to BIS except the addition of step(3') and step(11) to step(16) where block access is taken into consideration. Let m be

the blocking factor.

ALGORITHM BBIS

input: $X=(x[1],x[2],\ldots\ldots,x[n]), x[0]=a$, $x[n+1]=b$, $B=(\alpha_1,\alpha_2,\ldots\ldots,\alpha_k)$, m .
output: $search=(search[1],\ldots\ldots,search[k])$ (required indices).

```
(1) L :=0;H :=n +1;i :=1;
    preblock :=-1;curblock :=0;
    X_L :=x [0]; X_H :=x [n +1];
(2) while i ≤k do
(3)   begin
(3')
        N :=H -L -1;
        J :=L +N · ⌈(α_i -X_L)/(X_H -X_L)⌉;
        curblock := ⌈J/m⌉;
        if curblock ≠preblock then
          begin
            read the curblock^th block;
            preblock :=curblock ;
          end
(4)     while N >0 and x [J]≠α_i do
(5)       begin
(6)         if x [J]<α_i then
(7)           begin L :=J ;X_L :=x [J]; end
(8)         else
(9)           begin H :=J ;X_H :=x [J]; end
(10)        N :=H -L -1;
            J :=L +N · ⌈(α_i -X_L)/(X_H -X_L)⌉;
(11)        curblock := ⌈J/m⌉;
(12)        if curblock ≠preblock then
(13)          begin
(14)            read the curblock^th block;
(15)            preblock :=curblock ;
(16)          end
(17)      end
(18)    if N ≤0 then
(19)      begin
            search [i ]:=L ;H :=n +1;
            X_H :=x [n +1];X_L :=x [L ];
```

```
        end
(20)    else
(21)      begin
            search [i]:=J;L:=J;H:=n+1;
            X_H:=x[n+1];X_L:=x[J];
          end
(22)    i:=i+1;
(23)  end
```

2.2.2. Analysis

In the discussion below, we assume both $X=(x[1],x[2],\ldots,x[n])$ and $B=(\alpha_1,\alpha_2,\ldots,\alpha_k)$ are uniformly distributed between a and b, and $x[0]=\alpha_0=a$ and $x[n+1]=\alpha_{k+1}=b$.

Like section 2.1, an *iteration* in BBIS is said to begin with execution of step (3), and a *search step* is said to begin with the execution of step (4).

Let F_{i_j} denote the searched subfile of the jth search step in the ith iteration and L_{i_j} and H_{i_j} be the lower and upper indices of F_{i_j}, i.e. $F_{i_j}=(x[L_{i_j}],\ldots,x[H_{i_j}])$. F_{i_j} consists of $N_{i_j}=H_{i_j}-L_{i_j}-1$ unchecked keys, which are uniformly distributed between $x[L_{i_j}]$ and $x[H_{i_j}]$. Obviously, $F_{1_1}=X$, $N_{1_1}=H-L-1$, $L_{1_1}=0$, and $H_{1_1}=n+1$.

Let K_{i_j} denote the index of the key accessed in the jth search step in the ith iteration. For $i\geq 2$, K_{i_0} is J in step(21) or L in step(19) in the $(i-1)^{st}$ iteration in BBIS, which is the required index for α_{i-1}. And, $K_{1_0}=0$.

We define the distance between two consecutive search steps in the ith iteration, $D_{i_j}=|K_{i_{j+1}}-K_{i_j}|$. Since there is at least one block retrieved for processing B, the minimum value of D_{1_0} is assumed to be m, the blocking factor.

$D_{i_0}=|K_{i_1}-K_{i_0}|$. By the distributions of X's and B's, $\dfrac{\alpha_i-x[K_{i_0}]}{x[n+1]-x[K_{i_0}]}$ is the probability of a random key in F_{i_1} being less than or equal to α_i. The number of random keys in F_{i_1} being less than or equal to α_i is $\dfrac{n}{k+1}$ by lemma 1 and its proof. The size of F_{i_1} is $n-\dfrac{n\cdot(i-1)}{k+1}$ by lemma 1. Hence,

$$\frac{\alpha_i-x[K_{i_0}]}{x[n+1]-x[K_{i_0}]}=\frac{\dfrac{n}{k+1}}{n-\dfrac{n\cdot(i-1)}{k+1}}.$$

And thus,

$$K_{i_1}=K_{i_0}+\left(n-\frac{n\cdot(i-1)}{k+1}\right)\cdot\frac{\alpha_i-x[K_{i_0}]}{x[n+1]-x[K_{i_0}]}=K_{i_0}+\left(n-\frac{n\cdot(i-1)}{k+1}\right)\cdot\frac{\dfrac{n}{k+1}}{n-\dfrac{n\cdot(i-1)}{k+1}}$$

$=K_{i_0}+\frac{n}{k+1}$. Thus, $D_{i_0}=\frac{n}{k+1}$.

By Perl et.al.'s corollary of lemma 1 [8] and our lemma 1, D_{i_1} is less than $\frac{1}{2}\cdot(n-\frac{n\cdot(i-1)}{k+1})^{2^{-1}}$.

The following lemma extends the result of Perl et.al.[8] to approximate the distance between 2 search steps of a single elment in B.

LEMMA 2. The average value of D_{i_j} is less than

$$(n-\frac{n\cdot(i-1)}{k+1})^{2^{-j}}$$

, where $j\geq 1$.

Proof. From lemma 1 and its proof, the ith iteration is to search α_i in subfile F_{i_1} with size $n-\frac{n\cdot(i-1)}{k+1}$. By the proof of Perl et.al's theorem 1 [8], the average value of the D_{i_j} is less than

$$(n-\frac{n\cdot(i-1)}{k+1})^{2^{-j}}$$

for $j\geq 1$. Q.E.D.

The approximated block accesses by BBIS is given by the following theorem.

THEOREM 2. The expected number of block accesses required by BBIS for searching B in X is less than

$$\sum_{i=1}^{k}\sum_{j=0}^{r-1} block\,(D_{i_j},m\,),$$

where,

$$r=loglog\,(n-\frac{n\cdot(i-1)}{k+1}),$$

$$block\,(D\,,m\,)=\begin{cases}1 & \text{if} D\geq m\\ D\,/m & \text{if} D<m\end{cases}$$

and D_{i_j} is the estimated distance between two consecutive search steps as defined above.

Proof. The inner sum represents the expected block accesses required to search for α_i. By theorem 1 above, we need at most $r=loglog\,(n-\frac{n\cdot(i-1)}{k+1})$ search steps to search for α_i. The definition of the function *block* reflects the fact that the number of block accesses required from the jth search step to $(j+1)^{st}$ search step is 1 if the distance between them is great than or equal to m. Otherwise, it is D_{i_j}/m, where the D_{i_j}/m is the probability that $x\,[K_{i_j}]$ and

$x[K_{i_j+1}]$ are in the same block, and K_{i_j} is the index of the key accessed in the jth search step in the ith iteration (since the keys in X are uniformly distributed). The outer sum is for total block accesses for searching every element in B. Q.E.D.

2.2.3. Experimental Results

In this experiment, five sorted files of 400,000 integers uniformly distributed between 0 and 2^{31} were generated, each with a different blocking factor. A 1,000 sets of sorted records were also generated with size k for $k=1$ to 20. Table 2 contains the theoretical/experimental results for the combination of n (size of file), m (blocking factor) and k (size of batch). Again, theorem 2 provides good approximation to the experimental results.

Figure 2 shows the savings of batched block accesses over unbatched block accesses in a file of 400,000 records with blocking factor of 100. Again, there is roughly more than 50% savings when $k > 30$.

From the analysis above, it is shown that the average number of record accesses required by batched interpolation search is less than $O(k \cdot loglog(n))$ where, k and n are the sizes of batch and file respectively. But in the event of non-uniform distribution, the record accesses could be $O(k \cdot n)$ in the worst case because the search may degenerate into a linear scan. In the section to follow, we will give a single algorithm, called *independent batched binary interpolation search,* which combines the algorithm BIS and a batched binary search algorithm to give $O(k \cdot loglog(n))$ average behavior and $O(k \cdot log(N))$ worst case behavior.

3. Independent Batched Binary Interpolation Search

The algorithm presented below consists of the union of two algorithms. One is the interpolation search which is from step(10) to step(15). The other is the binary search which is from step(16) to step(20). For each α_i in the batch file, the algorithm invokes the interpolation search and the binary search. When one of them finds the required index, both of them will stop looking for α_i and begin the searching for α_{i+1}.

As before, the algorithm is described in both non-blocked and blocked environments.

3.1. Independent Batched Binary Interpolation Search

3.1.1. Algorithm

ALGORITHM IBBIS

input: $X=(x[1],x[2],\ldots\ldots,x[n])$, $x[0]=a$, $x[n+1]=b$, $B=(\alpha_1,\alpha_2,\ldots\ldots,\alpha_k)$.
output: $search=(search[1],search[2],\ldots\ldots,seach[k])$ (required indices).

```
(1) IL :=0;IH :=n +1;BL :=1;BH :=n ;i :=1;
(2) X_L :=x [0];X_H :=x [n +1];
(3)  while  i ≤k  do
(4)     begin
(5)       IN :=IH -IL -1;
(6)       IJ :=IL +IN · ⌈ (α_i -X_L)/(X_H -X_L) ⌉;
(7)       BJ := ⌈ (BL +BH)/2 ⌉
(8)       while  IN >0 and BL ≠BH  and x [IJ ]≠α_i  and x [BJ ]≠α_i  do
(9)         begin
(10)          if  x [IJ ]<α_i  then
(11)            begin  IL :=IJ ;X_L :=x [IJ ];  end
(12)          else
(13)            begin  IH :=IJ ;X_H :=x [IJ ];  end
(14)          IN :=IH -IL -1;
(15)          IJ :=IL +IN · ⌈ (α_{i+1}-X_L)/(X_H -X_L) ⌉
(16)          if  x [BJ ]<α_i  then
(17)            BL :=BJ ;
(18)          else
(19)            BH :=BJ ;
(20)          BJ := ⌈ (BL +BH)/2 ⌉
(21)        end
(22)      if  IN ≤0  then
(23)        begin
            search [i ]:=IL ;IH :=n +1;
            BL :=IL +1;BH :=n ;
            X_L :=x [IL ];X_H :=x [n +1];
(24)        end
(25)      if  BL =BH  then
          begin
            search [i ]:=BL ;IL :=BL ;IH :=n +1;
            X_L :=x [IL ];X_H :=x [n +1];
            BL :=BL +1;BH :=n ;
(26)      end
(27)      else
(28)        begin
(29)          if  x [IJ ]=α_i  then
(30)            begin
(31)              search [i ]:=IJ ;IL :=IJ ;IH :=n +1;
                  BL :=IJ +1;BH :=n ;
(32)              X_L :=x [IJ ];X_H :=x [n +1];
```

```
(33)            end
(34)          else
(35)            begin
(36)              search [i]:=BJ;IL:=BJ;IH:=n+1; BL:=BJ+1;BH:=n;
                 BL:=BJ+1;BH:=n;
(37)              X_L:=x[BJ];X_H:=x[n+1];
(38)            end
(39)        end
(40)    i:=i+1;
(41)  end
```

3.1.2. Analysis

In the discussion below, an *iteration* is said to begin with the execution of step (3), and a *search step* is said to begin with the execution of step (8). We assume again that the file, $X=(x[1],x[2],\ldots,x[n])$, and the batched queries, $B=(\alpha_1,\alpha_2,\ldots,\alpha_k)$, are both uniformly distributed between a and b, and $x[j]<x[j+1](1\leq j\leq n-1)$ and $\alpha_i\leq\alpha_{i+1}$ $(1\leq i\leq k-1)$.

Since algorithm IBBIS consists of two independent search routines, two immediate results can be obtained by referring to the well known result of binary search and our result of interpolation search presented above.

THEOREM 3. The number of record accesses required by IBBIS for searching B is at most

$$2\sum_{i=1}^{k} log\,(n-\frac{n\cdot(i-1)}{k+1}).$$

Proof. The worst case for IBBIS to search for α_i is for the binary search portion to dominate the search. When α_i is found by the binary search, the interpolation search portion of IBBIS will also stop. As before, the size of file X at the time α_i is being searched is reduced by $\frac{n\cdot(i-1)}{k+1}$ records. Since every search step of IBBIS involves 2 record accesses, the number of record accesses for finding α_i in the worst case is

$$2log\,(n-\frac{n\cdot(i-1)}{k+1}).$$

And the theorem is proved. Q.E.D.

THEOREM 4. The average number of record accesses required by IBBIS for searching batched records with size k in a file with size n is less than

$$2\sum_{i=1}^{k} loglog\,(n-\frac{n\cdot(i-1)}{k+1}).$$

Proof. Similar to the proof of theorem 3, except that if the interpolation search portion dominates, the average behavior of IBBIS will just be 2 times that of the average behavior of algorithm BIS. Q.E.D.

3.1.3. Experimental Results

In this experiment, we generated six sorted files of random integers. Five of them are uniformly distributed and one is non-uniformly distributed. The algorithm IBBIS was run against these files with 1,000 sets of batched queries from size 1 to 20. Table 3 gives the comparison of the theoretical result and experimental results with IBBIS running on the five uniformly distributed files.

Figure 3 gives the experimental results of running algorithm IBBIS and algorithm BIS over uniformly distributed file. Not surprisingly, the latter provides better performance. Similarly, figure 4 gives the comparison of results of executing IBBIS and BIS over non-uniformly distributed file. The former gives better performance than the latter. This figure only gives the savings by IBBIS

The proposal of alternating binary search with interpolation search to remedy the possible worst case behavior of the latter is first proposed by [9] for non-batched searching. Their proposed algorithm denoted by IBS, which is more complex than ours, gives the analytic prediction of $4 loglog\ (n)$ record accesses in the average which is more than a factor of 2 larger than ours. Our experiment shows that algorithm IBBIS performs a little better (table 4) for non-batched environment, and alternating better (figure 5) in batched environment than the algorithm described in [9].

3.2. Independent Blocked Batched Binary Interpolation Search

3.2.1. Algorithm

We will add the block access consideration to IBBIS described above to construct the algorithm below.

ALGORITHM IBBBIS

input: $X=(x[1],x[2],\ldots\ldots,x[n])$, $x[0]=a$, $x[n+1]=b$, $B=(\alpha_1,\alpha_2,\ldots\ldots,\alpha_k)$, m.
output: $search=(search[1],search[2],\ldots\ldots,search[k])$ (required indices).

(1) $IL:=0; IH:=n+1;$
(2) $BL:=1; BH:=n; i:=1;$
(3) $X_L:=x[0]; X_H:=x[n+1];$
  $Ipreblock:=-1; Icurblock:=0;$
  $Bpreblock:=-1; Bcurblock:=0;$
(4) **while** $i \leq k$ **do**
(5) **begin**
(6) $IN:=IH-IL-1;$
  $IJ:=IL+IN\cdot\left\lceil\dfrac{\alpha_i-X_L}{X_H-X_L}\right\rceil;$

```
        Icurblock := ⌈IJ/m⌉;
        if Ipreblock ≠ Icurblock then
          begin
            read Icurblock^th block;
            Ipreblock := Icurblock;
          end
        BJ := ⌈(BL + BH)/2⌉;
        Bcurblock := ⌈BJ/m⌉;
        if Bpreblock ≠ Bcurblock then
          begin
            read Bcurblock^th block;
            Bpreblock := Bcurblock;
(14)      end
(15)    while IN > 0 and BL ≠ BH and x[IJ] ≠ α_i
            and x[BJ] ≠ α_i do
(16)      begin
(17)        if x[IJ] < α_i then
(18)          begin IL := IJ; X_L := x[IJ]; end
(19)        else
(20)          begin IH := IJ; X_H := x[IJ]; end
(21)        IN := IH - IL - 1;
(22)        IJ := IL + IN · ⌈(α_i - X_L)/(X_H - X_L)⌉;
(23)        Icurblock := ⌈IJ/m⌉;
(24)        if Ipreblock ≠ Icurblock then
(25)          begin
(26)            read Icurblock^th block;
(27)            Ipreblock := Icurblock;
(28)          end
(29)        if x[BJ] < α_i then
(30)          BL := BJ;
(31)        else
(32)          BH := BJ;
(33)        BJ := ⌈(BL + BH)/2⌉;
(34)        Bcurblock := ⌈BJ/m⌉;
(35)        if Bpreblock ≠ Bcurblock then
(36)          begin
```

```
(37)            read Bcurblock^th block;
(38)            Bpreblock :=Bcurblock ;
              end
(39)        end
(40)      if IN ≤0 then
(41)        begin
            search [i ]:=IL ;IH :=n +1;
            BL :=IL +1;BH :=n ;
            X_L :=x [IL ];X_H :=x [n +1];
(42)        end
(43)      if BL =BH then
(44)        begin
            search [i ]:=BL ;IL :=BL ;IH :=n +1;
            X_L :=x [IL ];X_H :=x [n +1];
            BL :=BL +1;BH :=n ;
(45)        end
(46)      else
(47)        begin
(48)          if x [IJ ]=α_i then
(49)            begin
(50)              search [i ]:=IJ ;IL :=IJ ;IH :=n +1;
                BL :=IJ +1;BH :=n ;
(51)              X_L :=x [IJ ];X_H :=x [n +1];
(52)            end
(53)          else
(54)            begin
(55)              search [i ]:=BJ ;IL :=BJ ;IH :=n +1;
                BL :=BJ +1;BH :=n ;
(56)              X_L :=x [BJ ];X_H :=x [n +1];
(57)            end
(58)        end
(59)      i :=i +1;
(60)    end
```

3.2.2. Analysis

Let us define an *iteration* in IBBBIS is to begin with the execution of step(4), and a *search step* begins with the execution of step(15).

Let IK_{i_j} be the index of the key accessed by the interpolation search in the jth search step in the ith iteration, BK_{i_j} be the index of the key accessed by the binary search in the jth search step in the ith iteration. Let BK_{i_0} be the required index for α_{i-1} for $i \geq 2$, and $BK_{1_0}=0$. The IK_{i_0} and IK_{1_0} are the same as K_{i_0} and K_{1_0} in section 2.2.2.

Define ID_{i_j} to be the distance between two consecutive search steps performed by interpolation search in the ith iteration, $ID_{i_j} = |IK_{i_{j+1}} - IK_{i_j}|$. Obviously, $ID_{i_j} = D_{i_j}$ as in theorem 2. Define BD_{i_j} to be the distance between two consecutive search steps performed by binary search in the ith iteration, $BD_{i_j} = |BK_{i_{j+1}} - BK_{i_j}|$. BD_{1_0} is assumed to be at least m, the blocking factor, since at least one block has to be retrieved by the binary search in IBBBIS.

LEMMA 3. The average value of BD_{i_j} is

$$\frac{n - \frac{n \cdot (i-1)}{k+1}}{2^{j+1}},$$

for $i \geq 1$ and $j \geq 0$, except that BD_{1_0} is greater than the blocking factor.

Proof. By the uniform distribution assumption, the average size of file X when α_i is being search for is $n - \frac{n \cdot (i-1)}{k+1}$ as before. The distance between the jth and the $(j+1)^{st}$ search steps is always half the distance between the $(j-1)^{st}$ and the jth search steps. The lemma is proved. Q.E.D.

The approximated block accesses for algorithm IBBBIS is similar to BBIS before except that the block accesses required for the binary search is also taken into consideration.

THEOREM 5. The average number of block accesses required by BBBIS for searching B in file X is less than

$$\sum_{i=1}^{k} \sum_{j=0}^{r-1} (block(ID_{i_j}, m) + block(BD_{i_j}, m)),$$

where,

$$block(D, m) = \begin{cases} 1 & \text{if } D \geq m \\ D/m & \text{if } D < m' \end{cases}$$

$$r = loglog\left(n - \frac{n \cdot (i-1)}{k+1}\right),$$

and ID_{i_j} and BD_{i_j} is as defined above.

Proof. Similar to the proof of theorem 2. Q.E.D.

3.2.3. Experiment Results

In this experiment, five sorted files of uniformly distributed random integers between 0 and 2^{31} were generated. Each of them has 400,000 records. Different

blocking factors denoted by m as well as different batched size denoted by k are used to experiment with algorithm IBBBIS. Table 5 gives the comparison of theorem 5 and the actual experimental results.

4. Summary

In this paper, the basic Interpolation Search algorithm is extended to provide batched searching over blocked and non-blocked database environments. Also, a combined algorithm of independent binary and interpolation search is given to remedy for the worst case behavior of the pure interpolation search while still retaining the average behavior of the latter.

Analytic expressions for the behavior of these extensions are given. All expressions are validated by extensive experiments.

Acknowledgments

We are very grateful to Doron Rotem and Arie Shoshani for many suggestions and useful comments on our work.

Appendix

A recent paper on interpolation search [15] claims the expected time complexity of $O(c_1 \cdot loglog(N)+c_2)$ for non-uniformly distributed keys. A detailed examination of the algorithm reveals that there is a need of 3 record accesses per "3-cycle" in [15], hence the complexity is really $3c_1 \cdot loglog(N)+3c_2$. The constents c_1 and c_2 are dependent on many factors, such as the distribution of the keys in a file and the parameters in [15], which can hardly determined. In practice, it could be very difficult to precisely determine the values of c_1 and c_2 other than the fact that $c_1 \geq 1$ and $c_2 > 0$. Also, as pointed out by the author, it is a open problem to minimize c_1 and c_2. Hence, $3c_1 \cdot loglog(N)+3c_2$ may be larger than $log(N)$ when c_1 is slightly larger than 1. Thus, the algorithm in [15] may not be more efficient than binary search in practice. Table 6 below compares the results of $3c_1 \cdot loglog(N)+c_2$ and $log(N)$ with $c_1=1$ and 1.5. Table 7 gives the comparison of the experimental results of running the *iterative reduction* algorithm in [15], binary search and our IBBIS algorithms on 9 files with different size N. The keys in these files are $x_i=(x_{i-1}+i)/5{,}000{,}000{,}000$ for $i=1$ to N.

As can be seen, the algorithm in [15] (which has not been validated experimentally by the author) does not provide much improvement over binary search and our algorithm in non-uniform distribution.

References

[1] Nijssen, G.M. *Efficient Batched Updating of a Random file.* Proc. 1971 ACM-SIGFIDET Workshop-Data Description, Access and Control. pp.173-186.

[2] Shneiderman, B. and Goodman, V. *Batched Searching of Sequential and Tree*

Structured Files. ACM Transactions on Database Systems, Vol.1, No.3, September 1976, pp.286-275.

[3] Palvia, P. *Expressions for Batched searching of Sequential and Hierarchical Files.* ACM Transactions on Database systems, Vol.10, No.1, March 1985, pp.97-106.

[4] Piwowarski, K. *Comments on Batched searching of Sequential and Tree-structured Files.* ACM Transactions on Database Systems, Vol.10, No.2, June 1985, pp.285-287.

[5] Peterson, W.W. *The Interpolation File Search Method.* Information 16, 612-615(1974).

[6] Price, C.E. *Table Lookup Techniques.* Computing Surveys, 3, 56-58(1971).

[7] Yao, A.C. and Yao, F.F. *The Complexity of Searching an Ordered Random Table.* Proc. Seventeenth Annual Sysmp. Foundations of Comptr. Sci., 1976, pp.173-177.

[8] Perl, Y., Itai, A. and Avni, H. *Interpolation Search--A loglog* (N) *Search.* Communication of the ACM, Vol.21, No.7, July 1978, pp.550-553.

[9] Santoro, N. and Sidney, J.B. *Interpolation-Binary search.* Information Processing Letters 20(1985), pp.179-181.

[10] Wong, H.K.T., LI, J.Z. *Hierarchical Bit Transposed Files,* To appear, 1986.

[11] Eggers, S., Olken, F., Shoshani, A., "A Compression Technique for Large Statistical Databases", Proc. 1981 International Conference on VLDB, Cannes, France, 1981.

[12] Nievergelt, J., Hinterberger, H., Sevcik, K.C., "The Grid File: An Adaptable, Symmetric Multikey File Structure", ACM TODS, 9, 1, pp38 - 71.

[13] Yao, S.B. *Approximating Block Accesses in Database Organization.* CACM 20, 4 (April, 1977), 260-261.

[14] Gounet, G.H., Rogers, L.D., and George, J.A. *An Algorithmic and Complexity Analysis of Interpolation Search,* Acta Inform., 13 (1980) pp. 39-52.

[15] Willard, Dan E., *Searching unindexed and Nonuniformly Generated Files in loglog* (N) *Time,* SIAMJ. Comput., Vol.14, No.4, November 1985.

	n=2000		n=3000		n=4000		n=5000		n=10000		n=400000	
K	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.
1	3.45	3.76	3.52	3.70	3.58	3.50	3.61	3.57	3.73	3.66	4.21	4.46
2	6.83	7.28	6.98	7.33	7.08	7.14	7.16	7.15	7.39	7.29	8.38	8.65
3	10.17	10.93	10.40	11.07	10.56	10.60	10.68	10.74	11.03	11.31	12.54	12.94
4	13.49	14.60	13.80	14.50	14.02	14.24	14.18	14.43	14.65	14.90	16.68	16.99
5	16.79	18.09	17.19	18.15	17.48	17.75	17.67	17.89	18.27	18.79	20.81	21.36
6	20.09	21.08	20.57	21.62	20.90	21.13	21.15	21.44	21.87	22.36	24.94	25.24
7	23.38	24.99	23.95	24.97	24.33	24.55	24.62	25.08	25.47	25.36	29.06	29.39
8	26.66	28.01	27.31	28.58	27.76	27.99	28.09	28.41	29.06	29.54	33.18	33.43
9	29.94	31.57	30.68	31.83	31.18	31.28	31.55	31.69	32.65	33.16	37.30	37.67
10	33.22	34.71	34.04	35.26	34.60	34.63	35.01	35.59	36.24	36.74	41.42	41.85
11	36.49	37.89	37.40	38.62	38.01	38.17	38.47	38.64	39.82	40.28	45.53	45.77
12	39.76	40.81	40.75	41.89	41.42	41.27	41.93	42.09	43.40	43.90	49.65	49.86
13	43.03	44.07	44.10	45.03	44.83	44.59	45.38	45.24	46.98	47.85	53.76	54.01
14	46.29	47.14	47.45	48.23	48.24	47.85	48.83	48.54	50.56	51.32	57.87	58.00
15	49.55	50.20	50.80	51.28	51.65	51.23	52.28	51.84	54.14	54.37	61.98	62.11
16	52.82	53.25	54.15	54.70	55.05	54.56	55.73	55.33	57.71	58.36	66.09	65.87
17	56.08	56.22	57.50	57.64	58.46	57.76	59.18	58.50	61.29	61.40	70.20	70.13
18	59.33	59.23	60.84	61.04	61.86	60.59	62.62	61.77	64.86	65.09	74.31	74.36
19	62.59	62.11	64.19	64.39	65.26	63.85	66.07	64.72	68.43	68.79	78.42	78.19
20	65.85	65.18	67.53	67.41	68.66	67.09	69.52	68.06	72.01	72.09	82.52	82.50

Table 1

	m=60		m=70		m=80		m=90		m=100	
K	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.
1	2.50	2.58	2.43	2.53	2.37	2.50	2.33	2.46	2.30	2.42
2	4.96	5.11	4.82	5.01	4.72	4.87	4.64	4.78	4.57	4.73
3	7.40	7.53	7.20	7.38	7.05	7.22	6.93	7.10	6.84	6.94
4	9.83	9.75	9.57	9.75	9.37	9.56	9.22	9.37	9.10	9.22
5	12.26	12.31	11.94	12.03	11.69	11.79	11.50	11.61	11.35	11.38
6	14.68	14.43	14.30	14.16	14.01	13.82	13.79	13.65	13.61	13.39
7	17.11	16.87	16.66	16.51	16.33	16.11	16.07	15.88	15.86	15.56
8	19.53	19.11	19.02	18.77	18.64	18.34	18.35	17.99	18.11	17.67
9	21.94	21.46	21.38	20.96	20.98	20.62	20.63	20.16	20.36	19.86
10	24.36	23.66	23.74	22.97	23.27	22.48	22.91	22.11	22.62	21.81
11	26.78	25.73	26.10	25.17	25.58	24.64	25.18	24.14	24.87	23.65
12	29.13	27.93	28.40	27.21	27.85	26.55	27.42	26.10	27.08	25.67
13	31.55	29.97	30.75	29.28	30.16	28.62	29.70	27.95	29.33	27.56
14	33.96	32.13	33.11	31.36	32.47	30.84	31.97	29.95	31.57	29.48
15	36.38	34.25	35.47	33.43	34.78	32.81	34.25	31.92	33.82	31.45
16	38.79	36.55	37.82	35.63	37.09	35.11	36.53	34.12	36.07	33.65
17	41.21	38.45	40.18	37.81	39.40	36.71	38.80	36.26	38.32	35.30
18	43.55	40.45	42.47	39.39	41.66	38.52	41.03	37.68	40.53	37.00
19	45.97	42.64	44.83	41.37	43.97	40.54	43.31	39.75	42.78	38.94
20	48.38	44.51	47.18	43.24	46.29	42.41	45.59	41.46	45.00	40.75

Table 2

	n=2000		n=3000		n=4000		n=10000		n=400000	
K	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.
1	6.90	6.96	7.05	7.02	7.16	7.10	7.46	7.53	8.43	8.21
2	13.66	13.14	13.96	14.08	14.17	14.15	14.79	14.94	16.77	16.43
3	20.34	20.52	20.81	21.06	21.13	20.74	22.07	22.03	25.08	24.68
4	26.98	27.37	27.61	28.06	28.04	27.84	29.31	29.49	33.36	33.13
5	33.59	34.04	34.39	35.36	35.93	34.94	38.54	36.61	41.63	40.97
6	40.18	40.15	41.15	41.54	41.81	41.45	43.74	43.82	49.88	49.21
7	46.76	47.27	47.90	48.23	48.67	48.50	50.94	50.87	58.13	57.40
8	53.33	54.05	54.63	54.94	55.52	55.36	58.12	58.18	66.37	65.60
9	59.89	60.18	61.36	61.21	62.36	62.08	65.30	64.85	74.61	73.81
10	66.44	66.72	68.08	67.75	69.20	69.18	72.48	71.73	82.84	81.97
11	72.98	72.51	74.80	73.96	76.02	76.02	79.64	78.84	91.07	90.11
12	79.56	78.89	81.51	80.78	82.85	82.12	86.81	85.99	99.30	98.47
13	86.06	85.75	88.21	87.82	89.67	88.72	93.97	93.17	107.52	106.34
14	92.59	91.65	94.91	93.83	96.49	95.54	101.13	100.22	115.75	114.14
15	99.11	98.12	101.61	100.75	103.30	101.87	108.28	107.62	123.97	122.36
16	105.64	104.74	108.31	107.07	110.11	109.83	115.43	114.51	132.19	130.29
17	112.16	111.33	115.00	112.93	116.92	115.06	122.58	121.64	140.40	137.83
18	118.67	117.35	121.69	119.22	123.75	121.18	129.73	128.63	148.62	145.43
19	125.19	123.98	128.38	124.88	130.53	127.67	136.87	135.31	156.84	153.82
20	131.70	130.30	135.06	131.12	137.33	134.18	144.02	142.54	165.05	161.27

Table 3

n	2loglog(n)	IBBIS	4(loglog(n)+2)	IBS
1000	6.62	6.58	21.24	7.21
2000	6.90	6.98	21.80	6.78
3000	7.06	7.02	22.12	7.03
4000	7.16	7.10	22.24	7.59
5000	7.22	6.96	22.44	7.52
10000	7.46	7.53	22.92	8.15
400000	8.42	8.21	24.84	9.36

Table 4

	m=60		m=70		m=80		m=90		m=100	
K	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.	T.R.	E.R.
1	6.50	6.49	6.43	6.43	6.37	6.40	6.33	6.33	6.30	6.30
2	12.96	12.83	12.82	12.79	12.72	12.77	12.64	12.67	12.57	12.59
3	19.40	19.21	19.20	19.15	19.05	19.06	18.93	19.01	18.84	18.83
4	25.83	25.56	25.57	25.45	25.37	25.27	25.22	25.21	25.10	25.09
5	32.26	31.61	31.94	31.36	31.69	31.16	31.50	30.95	31.35	30.84
6	38.68	37.94	38.30	37.65	38.01	37.38	37.79	37.07	37.61	37.11
7	45.11	44.70	44.66	43.91	44.33	43.52	44.07	43.08	43.86	43.32
8	51.53	50.66	51.02	50.38	50.64	49.77	50.35	49.66	50.11	49.31
9	57.94	56.55	57.38	56.41	56.95	55.02	56.63	55.35	56.36	55.39
10	64.36	62.82	63.74	63.44	63.27	61.09	62.91	61.48	62.62	61.05
11	70.78	69.00	70.10	68.56	69.58	67.57	69.18	67.36	68.87	67.08
12	76.13	74.92	75.40	74.41	74.85	73.67	74.42	73.36	74.08	72.78
13	82.55	80.76	81.75	80.28	81.16	79.39	80.70	78.93	80.33	78.41
14	88.96	86.28	88.11	86.24	87.47	84.96	86.97	84.43	86.57	83.86
15	95.38	92.63	94.47	92.22	93.78	91.21	93.25	90.57	92.82	90.07
16	101.79	98.27	100.82	97.51	100.09	96.77	99.53	96.13	99.07	95.41
17	108.21	104.16	107.18	103.48	106.40	102.52	105.80	101.70	105.32	101.21
18	113.55	109.66	112.47	108.94	111.66	108.18	111.03	107.35	110.53	108.71
19	119.97	115.44	118.83	114.37	117.97	113.50	117.31	112.79	116.78	112.18
20	126.38	121.95	125.18	120.31	124.29	119.19	123.59	118.34	123.00	117.60

Table 5

N	$log(N)$	$3c_1 \cdot loglog(N)+c_2$	
		$c_1=1$	$c_1=1.5$
1000	10.01	$10.01+c_2$	$15.02+c_2$
2000	11.01	$10.43+c_2$	$15.64+c_2$
3000	11.60	$10.65+c_2$	$15.98+c_2$
4000	12.02	$10.81+c_2$	$16.21+c_2$
5000	12.34	$10.92+c_2$	$16.39+c_2$
10000	13.34	$11.26+c_2$	$16.90+c_2$
50000	15.68	$11.96+c_2$	$17.95+c_2$
100000	16.68	$12.23+c_2$	$18.35+c_2$
500000	19.01	$12.80+c_2$	$19.20+c_2$

Table 6

N	iterative reduction	binary search	IBBIS
1000	10.39	8.79	12.30
2000	10.93	10.01	13.44
3000	11.32	10.61	14.36
4000	11.39	10.92	14.36
5000	11.67	11.26	14.82
10000	11.73	12.39	16.35
50000	16.26	14.66	18.96
100000	19.91	15.76	21.36
500000	30.10	17.86	24.03

Table 7

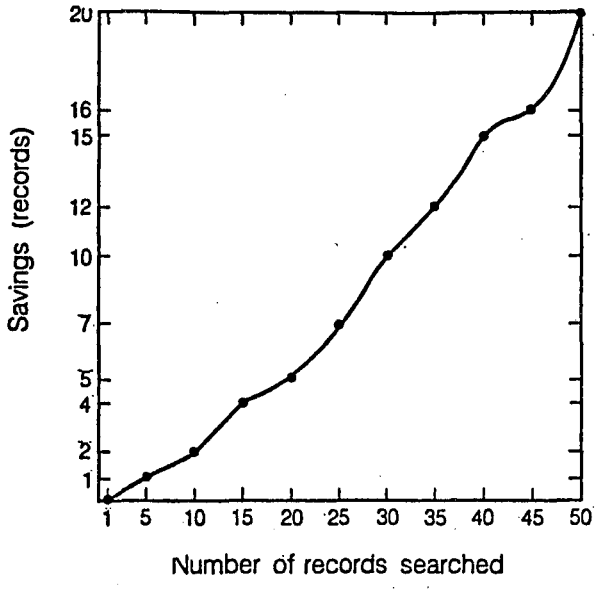

Savings (records)
20
16
15
12
10
7
5
4
2
1
1 5 10 15 20 25 30 35 40 45 50
Number of records searched

Fig. 1

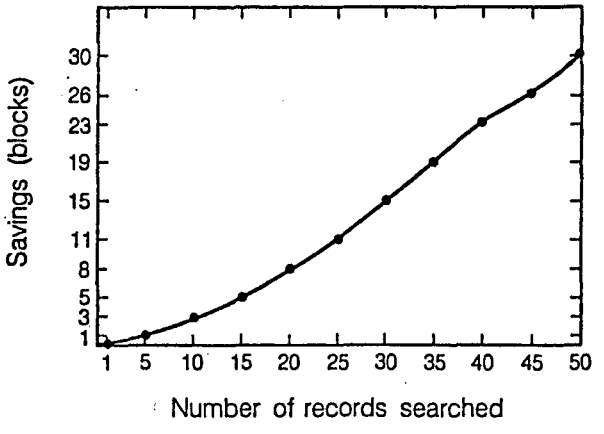

Savings (blocks)
30
26
23
19
15
11
8
5
3
1
1 5 10 15 20 25 30 35 40 45 50
Number of records searched

Fig. 2

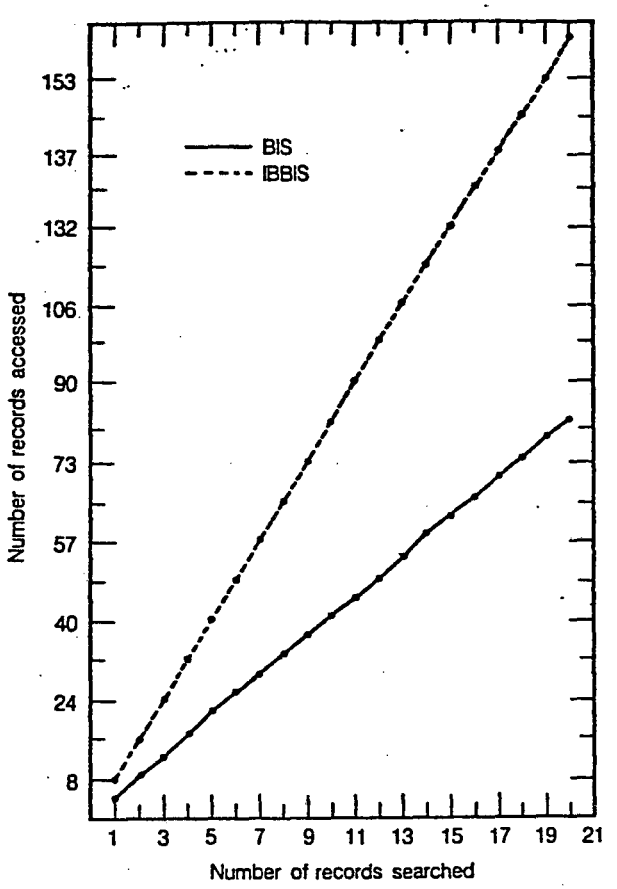


Fig. 3

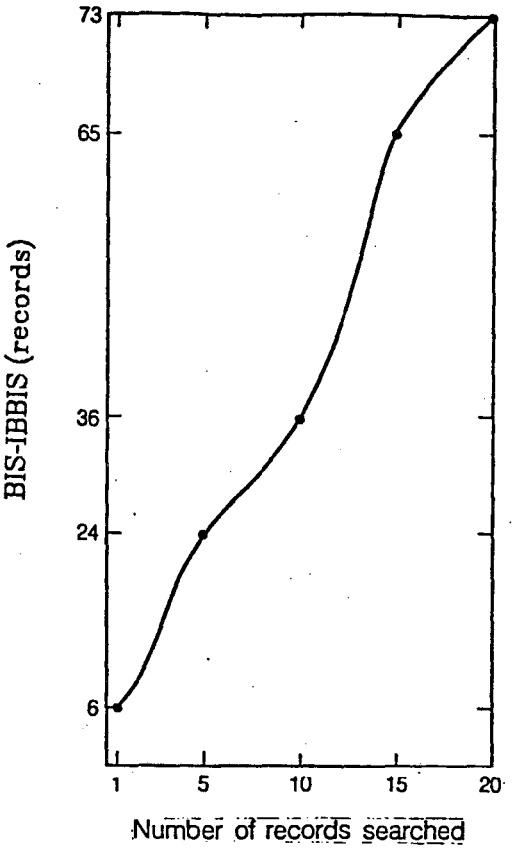


Fig. 4

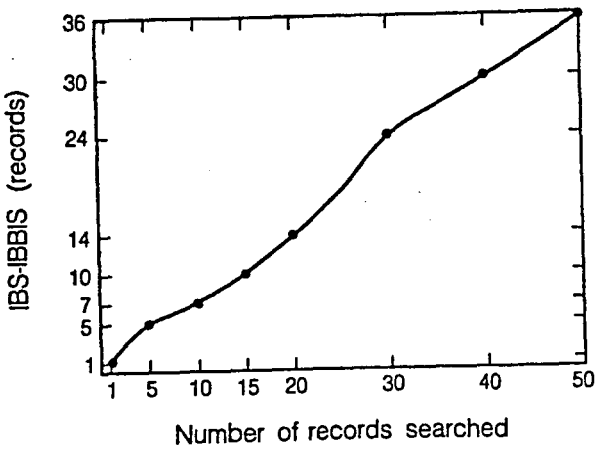


Fig. 5

This report was done with support from the Department of Energy. Any conclusions or opinions expressed in this report represent solely those of the author(s) and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory or the Department of Energy.

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720