LBL-22848 c.2

Lawrence Berkeley Laboratory

UNIVERSITY OF CALIFORNIA, BERKELEY

RECEIVED
MAR 17 1987
DOCUMENTS SECTION

Information and Computing Sciences Division

Presented at The Third International Conference on Data Engineering, Los Angeles, CA, February 2-6, 1987

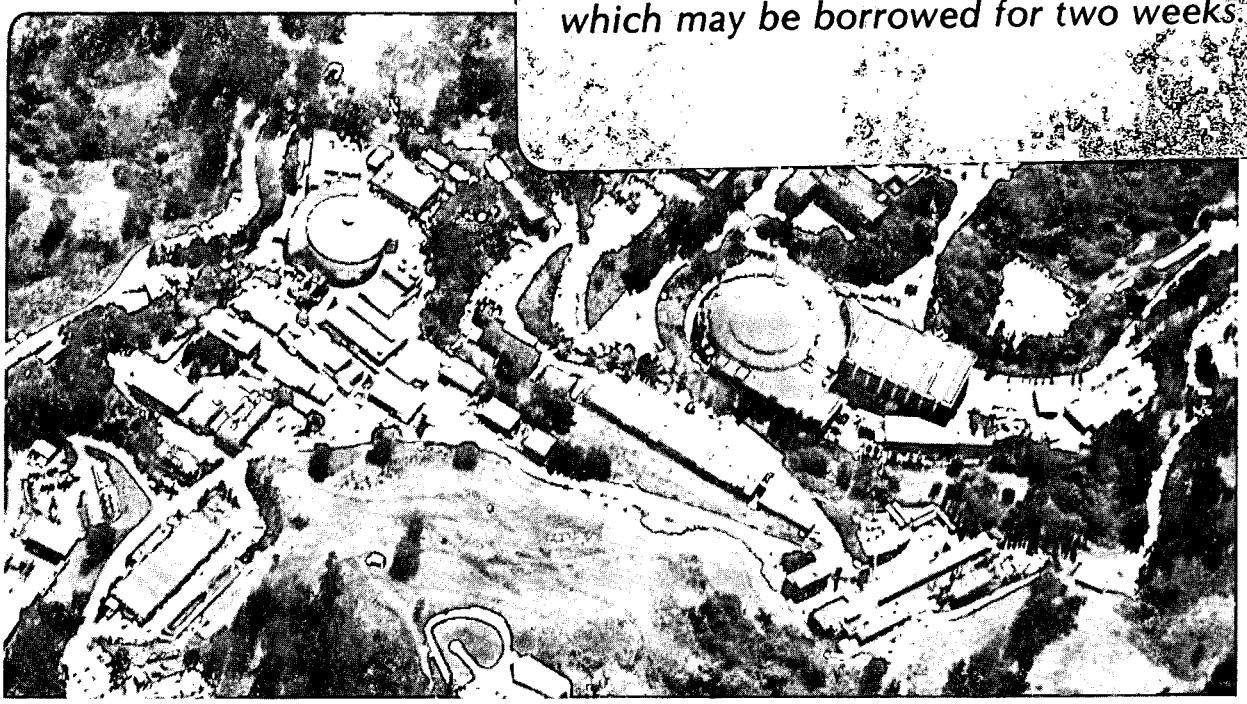
BATCHED INTERPOLATION SEARCHING ON DATABASES

J. Li and H.K.T. Wong

February 1987

TWO-WEEK LOAN COPY

This is a Library Circulating Copy which may be borrowed for two weeks.



LBL-22848 c.2

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

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-22848

Batched Interpolation Searching on Databases

Jian-zhong Li and Harry K.T. Wong

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

February, 1987

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.

Batched Interpolation Searching on Databases

Jian-zhong Li* and Harry K.T. Wong
Lawrence Berkeley Laboratory,
University of California

Abstract

This paper examines the effect of batching search requests in the Interpolation Search Algorithm on ordered tables in main-memory as well as in a more typical database environment, i.e. a blocked secondary memory. Experiments are performed on several hybrid interpolation search algorithms over non-uniformly distributed data. The effect of batching on these algorithms is examined in terms of algorithms, analysis and experiments.

Algorithms, analytic expressions and experimental results of these extensions are given and described. Analytic expressions of these algorithms are validated by 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 [11]. The second problem is related to the searching of hierarchical relationship implemented in a file structure called hierarchical transposed file [10]. 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 $x_1 < x_2 < \cdots < x_{1000}$ 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 between 600th and $1{,}000^{th}$ records. We therefore take a second guess that x_i 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.

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.

First published by Peterson [5], the Interpolation Search problem 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 [6-8,14]. 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 if the keys are uniformly distributed. In [8,14], 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. Several hybrid interpolation search algorithms have been proposed to remedy the worst case behavior $(O(N))$ of the Interpolation Search Algorithm [9, 15, 16].

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. The analysis and the experiments of batched interpolation search algorithm in non-blocked environment are shown in section 2. In section 3, the analysis and the experiments are described for the batched interpolation search algorithm in blocked environment. Experiments on several hybrid interpolation search algorithms and the discussion of hybrid algorithms for batched interpolation search are given in section 4. Section 5 summaries and concludes the paper.

2. Batched Interpolation Search

2.1. 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,

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

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 rth place of X where r is such that $x[r] = \alpha_{i-1}$ or $x[r] < \alpha_{i-1} < x[r+1]$. The savings of batched searching are achieved because the size of file X is monotonically decreasing.

ALGORITHM BIS

(1) $L := 0; H := n+1; i = 1;$
(2) **FOR** i=1 **TO** k **DO**
(3) Search α_i in $(x[L], \cdots, x[H])$ using Interpolation Search Algorithm;
(4) L=r where r satisfies $x[r] = \alpha_i$ or $x[r] < \alpha_i < x[r+1]$; H:=n+1;
(5) **END** .

The variables L and H represent lower and upper indices of the file searched respectively. For each i $(1 \leq i \leq k)$ α_i is searched by step(3) in the algorithm BIS in subfile $F_i = (x[L], \ldots\ldots, x[n+1])$ using the Interpolation Search Algorithm [5]. 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) and step(4).

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 (4) 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 determined 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 and B be the same as mentioned above. 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.

2.2. 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 and sorted 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 of the batched interpolation search algorithm on a file of 400,000 uniformly distributed integers over its unbatched counterpart. The savings are roughly 40% when $k \geq 20$.

3. 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.

3.1. Algorithm

The algorithm BBIS below is similar to BIS except the addition of step(7) where block access is taken into consideration. In the discussion below, we assume that $X = (x[1], \cdots, x[n])$ and $B = (\alpha_1, \ldots, \alpha_k)$ are uniformly distributed between a and b, $x[0] = \alpha_0 = a$, $x[n+1] = \alpha_{k+1} = b$ and m is the blocking factor.

ALGORITHM BBIS

```
(1)  L :=0;H :=n +1;
(2)  for i =1 to k do
(3)    begin
(4)      while H-L -1>0 do
(5)        begin
(6)          J=L +(H-L -1)· ⌈ (α_i -x [L ]) / (x [H]-x [L ]) ⌉;
(7)          if the ⌈J /m ⌉^th
             block is not in memory than read it;
(8)          if x [J]<α_i then L:=J;
(9)          if x [J]>α_i then H:=J;
(10)         if x [J]=α_i then
(11)         L:=J; H:=n+1; goto 14(Key found);
(12)       end ;
(13)     L:=J; H:=n+1; (Key not found)
(14)   end ;
```

As in 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 (5). The 0th search step refers to the beginning of an *iteration*.

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, $F_{1_0}=X$, $N_{1_0}=H-L-1$, $L_{1_0}=0$, and $H_{1_0}=n+1$.

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 L in step(11) or step(13) in the $(i-1)^{st}$ iteration in BBIS, which is the required index for α_{i-1}. And, $K_{1_0}=0$. For $i\geq 1$ and $j\geq 1$, K_{i_j} is the J in step (6).

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.

Next we will derive $D_{i_0}=|K_{i_1}-K_{i_0}|$. By the distribution 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_0} being less than or equal to α_i. The number of random keys in F_{i_0} being less than or equal to α_i is $\dfrac{n}{k+1}$ by lemma 1 and its proof. The size of F_{i_0} is $n-\dfrac{n\cdot(i-1)}{k+1}$ by lemma 1. Hence,

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

And thus, from step (6),

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

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

$$D_{i_0} = \frac{n}{k+1}. \tag{1}$$

By Perl et.al.'s corollary of lemma 1 [8] and our lemma 1, D_{i_1} is less than

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

The following lemma extends the result of Perl et.al.[8] to approximate the distance between 2 search steps of searching 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_0} 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 determined by (1), (2) and lemma 2.

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 not in the same block, since the keys in X are uniformly distributed. The outer sum is for total block accesses for searching all the elements in B. Q.E.D.

3.2. 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 batched and sorted records were also generated with size k for $k=1$ to 20. Table 2 contains the theoretical/experimental results for the combination of 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 $k > 30$.

4. Hybrid Algorithms for Non-uniformly Distributed Data

To remedy the worst case behavior $O(N)$ of interpolation search in the event of non-uniformly distributed data, several hybrid algorithms have been proposed for single key search.

The proposal of combining binary search with interpolation search first appeared in [9]. This algorithm (denoted by IBS) gives the analytic prediction of $O(loglog\,(N)+2)$ record accesses on the average on uniformly distributed files and $O(log\,(N))$ worst case behavior.

In [16] an algorithm (denoted by ABI) is shown which simply alternates between the methods of binary and interpolation search to obtain a retrieval time $O(\sqrt{log\,(N)})$ on non-uniformly distributed ordered files.

A recent paper on interpolation search [15] gives an algorithm (denoted by IR) with the expected time complexity of $O(c_1 \cdot loglog\,(N)+c_2)$ for non-uniformly distributed keys.

4.1. Experiments

To examine algorithms IBS, ABI, and IR in practice, we have implemented these algorithms according to the original papers. Six files of different sizes containing non-uniformly distributed keys were generated. The sizes are respectively 1,000, 5,000, 10,000, 50,000, 100,000 and 500,000 keys. For each algorithm, 1,000

searches are performed on each file. Fig. 3 contains the results of running IBS, ABI, IR and binary search (denoted by BI). As can be seen from the figure, binary search performs the best among the algorithms when the size is 50,000 or less. With file size exceeding 50,000, IBS requires the least number of record searches. The surprise is that IR and ABI, despite their attractive asymptotic behavior, behave consistently worse than the other two in practice.

4.2. Batched Hybrid Algorithms

From algorithm BIS and lemma 1, it is obvious that the worst case behavior of our batched interpolation search algorithm is $O(\sum_{i=1}^{k}(N-\frac{N\cdot(i-1)}{k+1}))$. We will provide algorithms for batched searching based on the hybrid interpolation search algorithms mentioned in this section.

Algorithm H

1. $L:=0$; $H:=n+1$;
2. **FOR** $i=1$ **TO** k **DO**
3. call hybrid interpolation search algorithm;
4. $L:=r$ where r satisfies $x[r]=\alpha_i$ or $x[r]<\alpha_i<x[r+1]$; $H:=n+1$;
5. **END** .

In step 3, the "hybrid interpolation search algorithm" can be any one of the hybrid interpolation algorithms mentioned above.

Let $X=(x[1],\cdots,x[n])$ be a database, $B=(\alpha_1,\ldots,\alpha_k)$ be batched keys, $D_0=0$, and D_i =the number of keys in X between $x[1]$ and α_i for $i=1$ to k, where $\alpha_0=x[1]$. Similar to theorem 1, we can give the average behavior of H when different hybrid interpolation search algorithms are used in step 3 in H.

When IBS is used, the average number of record accesses required by H on non-uniformly distributed ordered files is less than

$$O(\sum_{i=1}^{k}(log(n-D_{i-1})+2)).$$

When ABS is useed, the average retrieval time required by H on non-uniformly distributed ordered files is less than

$$O(\sum_{i=1}^{k}\sqrt{log(n-D_{i-1})}).$$

When IR is used, the expected time complexity required by H on non-uniformly distributed ordered files is less than

$$O(\cdot c_1\cdot\sum_{i=1}^{k}loglog(n-D_{i-1})+\cdot k\cdot c_2).$$

Next we will derive the expected values of D_i's. Let I_i denote the index of α_i's in X. The expected values of I_i's are

$$E(I_1) = \frac{1}{\sum_{l=1}^{n-(k-1)} \binom{n-l}{k-1}} \cdot \sum_{j=1}^{n-(k-1)} j \cdot \binom{n-j}{k-1},$$

$$E(I_i) = \frac{1}{\sum_{l=E(I_{i-1})+1}^{n-(k-i)} \binom{n-l}{k-i}} \cdot \sum_{j=E(I_{i-1})+1}^{n-(k-i)} j \cdot \binom{n-j}{k-i}$$

for i=2,...,k .

Thus, the expected values of D_i's are

$$E(D_1) = E(I_1),$$

$$E(D_i) = \ |E(I_i) - 1| \qquad \text{for } i = 2,\ldots,k.$$

In order to examine the behavior of H, we generated a file of 500,000 non-uniformly distributed sorted random numbers. 1,000 sets of sorted random numbers were also generated with the set size varies from 1 to 40 keys. We run H on the file using IR, IBS and ABI in step (3) in H. Fig. 4 shows the average number of record accesses of searching the batched random numbers for different batch sizes. As can be seen, batched IBS is the most efficient one followed by batched ABI and batched IR.

Fig. 5, Fig. 6 and Fig. 7 show the savings of executing batched IR, batched IBS and batched ABI on the file of 500,000 non-uniformly distributed random numbers over their non-batched counterparts. The savings of batched IR, batched ABI, and batched IBS are roughly 20%, 25%, and 70% respectively when the batch size exceeds 40.

5. Summary and Conclusion

In this paper, the basic Interpolation Search algorithm is extended to provide batched searching over blocked and non-blocked database environments. An examination on some hybrid interpolation search algorithms over non-uniformly distributed data is performed. Also, the effect of batching on these hybrid algorithms is given.

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

Our experiment has revealed some interesting surprises on hybrid algorithms for non-uniformly distributed data. The first surprise is that the simple binary search consistently out-performs other more elaborate algorithms in a non-batched environment. The second is that IBS, which combines binary and

interpolation search in a straightforward manner, behaves much better than the asymptotically more attractive algorithms such as IR and ABI in batched or non-batched environments.

These algorithms are an integral part of a scientific and statistical database management prototype system [10].

Acknowledgments

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

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. *Addressing for Random-access storage,* IBM J. Res. and Develop. 1.(1957), 131-132.

[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,* LBL Technical Report LBL-21284, 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.

[16] Willard, D., *Surprising efficient Search Algorithm for Nonuniformly Generated Files,* 21st Allerton Conference on Communication Control and Computing, 1983, pp. 656-662.

	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	3.53	3.59	3.61	3.57	3.73	3.66	4.21	4.46
2	6.83	7.25	6.95	7.33	7.03	7.14	7.16	7.15	7.33	7.29	8.38	8.65
3	10.17	10.63	10.40	11.07	10.55	10.80	10.63	10.74	11.03	11.31	12.54	12.94
4	13.49	14.60	13.80	14.53	14.02	14.24	14.18	14.43	14.85	14.93	16.68	16.93
5	16.78	18.02	17.16	18.15	17.45	19.75	17.67	17.89	18.27	18.79	20.81	21.36
6	20.09	21.03	20.57	21.62	20.63	21.13	21.15	21.44	21.87	22.35	24.94	25.24
7	23.35	24.99	23.93	24.67	24.33	24.55	24.22	25.03	25.47	25.35	29.06	29.39
8	25.65	29.01	27.31	28.55	27.75	27.69	28.07	28.41	29.06	29.54	33.18	33.43
9	28.94	31.57	33.69	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	35.25	34.69	34.63	33.01	35.58	35.24	35.74	41.42	41.85
11	35.49	37.67	37.49	38.62	38.01	38.17	38.47	38.64	38.82	39.28	45.53	45.77
12	38.78	40.81	40.75	41.67	41.42	41.27	41.93	42.09	43.40	43.93	49.65	49.86
13	43.03	44.07	44.10	45.03	44.83	44.59	45.38	45.24	46.63	47.85	53.76	54.01
14	46.28	47.14	47.45	48.23	49.24	47.85	48.83	48.54	50.55	51.32	57.87	58.00
15	49.55	50.20	50.80	51.28	51.85	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	54.55	55.73	55.33	57.71	58.36	66.07	65.87
17	56.08	58.22	57.50	57.64	58.45	57.76	59.18	58.50	61.22	61.40	70.20	70.13
18	59.33	59.23	62.64	61.04	61.85	60.59	62.82	61.77	64.85	65.09	74.31	74.36
19	62.55	62.11	64.19	64.38	65.25	63.85	66.07	64.72	68.43	68.72	78.42	78.19
20	65.84	65.15	67.52	67.41	68.65	67.07	69.52	68.05	72.01	72.09	82.52	82.59

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	2.59	2.58	2.43	2.53	2.37	2.50	2.33	2.46	2.30	2.42
2	4.95	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	7.39	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	12.25	12.31	11.94	12.03	11.69	11.79	11.50	11.61	11.35	11.39
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	15.83	15.85	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	21.39	20.98	20.98	20.62	20.63	20.16	20.38	19.85
10	24.35	23.66	23.74	22.87	23.27	22.48	22.91	22.11	22.62	21.81
11	25.78	25.73	26.10	25.17	25.58	24.64	25.18	24.14	24.67	23.65
12	28.13	27.93	28.49	27.21	27.85	26.55	27.42	26.10	27.03	25.67
13	31.55	28.57	30.75	29.28	30.16	28.62	29.70	27.95	29.33	27.56
14	33.93	32.13	33.11	31.36	32.47	30.84	31.57	29.95	31.57	29.48
15	35.38	34.25	35.47	33.43	34.78	32.81	34.25	31.92	33.82	31.45
16	38.79	35.55	37.82	35.63	37.09	35.11	36.53	34.12	35.07	33.65
17	41.21	38.45	40.18	37.81	39.40	35.71	38.80	35.25	38.32	35.30
18	43.55	40.45	42.47	39.39	41.66	38.52	41.03	37.68	40.53	37.03
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	45.28	42.41	45.59	41.45	45.02	40.75

Table 2

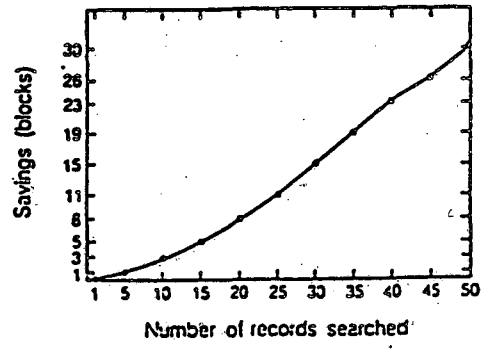


Fig. 2

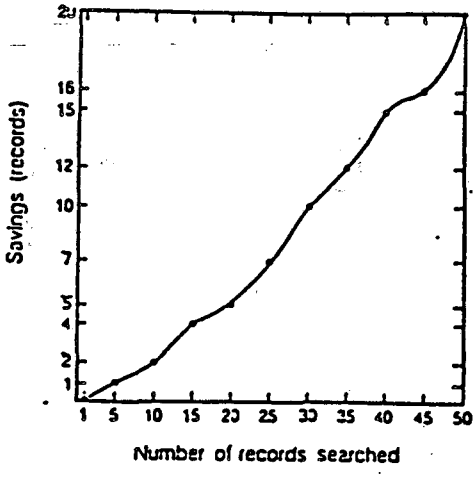


Fig. 1

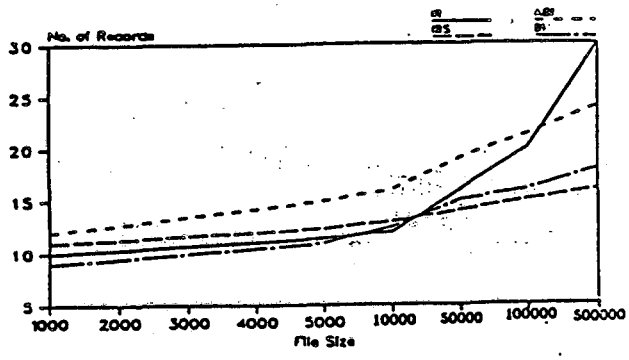


Fig. 3

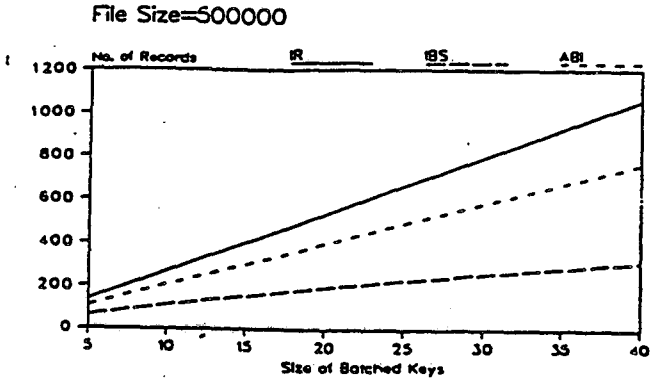


Fig. 4

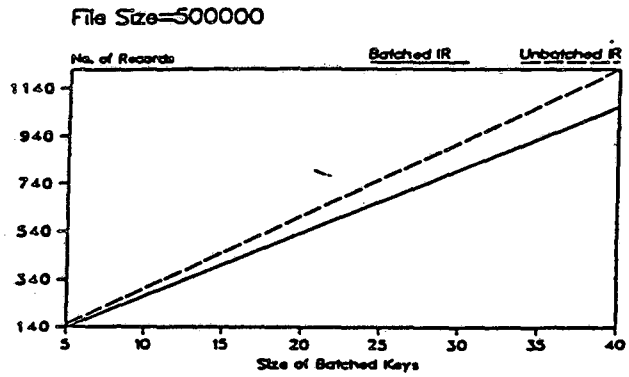


Fig. 5

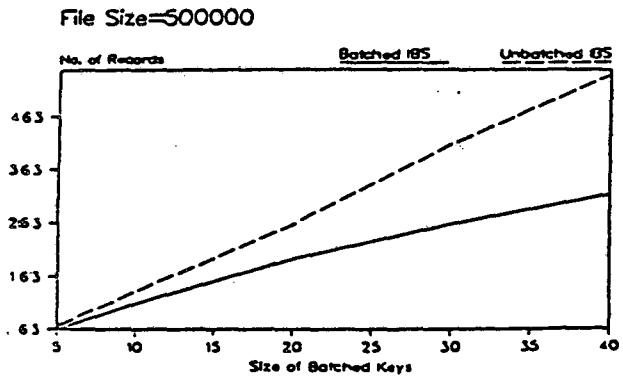


Fig. 6

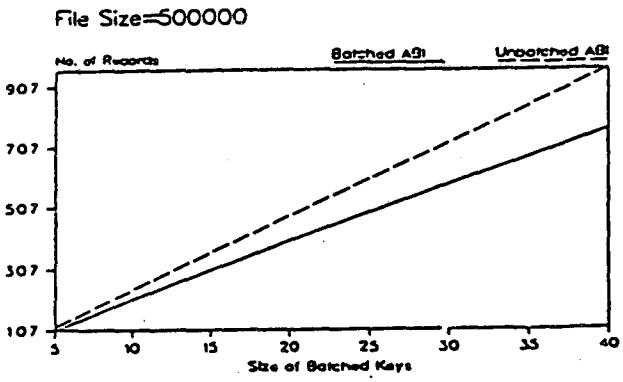


Fig. 7

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.

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