Batched Interpolation Search

Y. P. MANOLOPOULOS†, J. (Y.) G. KOLLIAS*‡ AND F. WARREN BURTON§

† *University of Thessaloniki, Department of Electrical Engineering, Division of Electronics and Computers, 54006 Thessaloniki, Greece*

‡ *National Technical University of Athens, Department of Electrical Engineering, Division of Computer Science, 15773 Athens, Greece*

§ *University of Colorado at Denver, Department of Electrical Engineering and Computer Science, Denver, Colorado 80202, U.S.A.*

In a previous study an ordered array of* N *keys was considered and the problem of locating a batch of* M *requested keys was investigated by assuming both batched sequential and batched binary searching. This paper introduces the idea of batched interpolation search, and two variations of the method are presented. Comparisons with the two previously defined methods are also made.

Received December 1985

1. INTRODUCTION

The most common techniques for searching an ordered array residing in main memory are the ones described in detail in Ref. 8, pp. 393–422: sequential, binary and Fibonacci search. Some other techniques are also proposed for searching, for example jump search,[17] polynomial search,[16] as well as interpolation search.[2, 4–6, 8, 10, 12–15]

For uniformly distributed values interpolation search seems the most promising, based on the average number of probes for the search of one element in an array. On the other hand, if we compare the running times, the performance of interpolation search decreases because the calculation of the array index is time-consuming.

In this paper we consider the problem posed in Ref. 11, that is, to find out efficiently which element of a set of M sorted distinct keys exists in an array of length N, where $N > M$. Let us describe the method. Suppose we have a dense array in main memory consisting of N sorted keys (say ascendingly), uniformly distributed and we want to detect which of the keys of a set of size M, which may or may not be uniformly distributed, exist in the array. If the keys are sorted in the same order as the keys of the array, the successful search of the first key of the set enables us to search for the second key in the remaining sub-array, i.e. the interval between the already-found key and the beginning of the array is exempted. Thus the original array shrinks continuously and the search becomes more efficient. In Ref. 11 an algorithm based on binary search was proposed. This technique named 'Batched Binary Search' ('BBS') was compared to 'Batched Sequential Search' ('BSS') (for the implementation of BSS in the environment of secondary storage see also Refs. 1 and 9), and was shown to outperform it up to a certain threshold value of M, which is a function of the variable N, beyond which BSS is preferable to BBS. Consider also that in Ref. 7 a similar problem was investigated, that of detecting whether a k-vector belongs to a set of n distinct k-vectors. An algorithm compromising binary and sequential search was proposed to speed the search process.

* To whom correspondence should be addressed.

† This material is based upon work supported by the National Science Foundation under Grant no. ECS-8312748. Now at: Department of Computer Science, University of Utah, Salt Lake City, Utah 84112, U.S.A.

Intuitively, it seems that binary search is not optimal, since it does not benefit from the knowledge that the keys may be distributed all over the array. In this paper we investigate the performance of two techniques (forming a family called 'Batched Interpolation Search' or 'BIS') under the following assumptions: (*a*) the cost metric is the number of probes and not the running time; (*b*) the keys of the array are uniformly distributed; (*c*) the M keys being searched for are uniformly distributed; and (*d*) the M keys are distinct and all of them exist in the array. The first (second) technique is based on interpolation and sequential (binary) search.

Interpolation search and BIS specifically may be used in a wide spectrum of applications. As reported in Ref. 12, in computers with fast floating-point arithmetic handling, interpolation search is superior to binary search in all cases and should be preferred for both internal and external searching operations. In the case of external searching the cost metric should be the number of block accesses. In distributed systems communication cost dominates computation cost. Thus interpolation and BIS would be useful in this environment if applied by a processor with fast local memory which can also access information in a central memory and/or another processor more slowly. Interpolation and BIS could also be used for searching in optical discs, where the span capability provides access to adjacent tracks without moving the access mechanism. Since interpolation tends very fast to the required record, it would be fair to say that in the environment of optical discs the last access would be performed within one span and thus the search cost is minimised.

The paper is organised as follows. In Section 2 the two variations are described and estimates are derived for the expected number of probes performed when the two techniques are applied. In Section 3 comparison of BBS and BSS is made and a breakpoint between BIS and BSS is defined, beyond which BSS is preferable to BIS. In Section 4 the effects of other assumptions are discussed.

2. ANALYSIS OF BATCHED INTERPOLATION SEARCH

First, we will describe briefly the simple interpolation search algorithm. For more information see Ref. 6.

Suppose we have an ascendingly ordered array of length N with keys k_i, where $1 \leqslant i \leqslant N$. These keys are drawn from a uniform distribution over the range (k_0, k_{N+1}). Suppose that we are given a key value k and want to know whether k exists in the array or not. The array index to be probed is given by the equation:

$$AI = \left\lceil N \frac{k-k_0}{k_{N+1}-k_0} \right\rceil \tag{1}$$

Where $\lceil x \rceil$ is the least integer greater than or equal to the real number x. If the key value k' of this array index is equal to k, then the search terminates successfully. If the key value k' is greater (less) than k, then the search continues in the left (right) sub-array by interpolating to a new array index which is:

$$AI = \left\lceil (AI-1) \frac{k-k_0}{k'-k_0} \right\rceil \tag{2a}$$

$$\left(AI = \left\lceil (N-AI) \frac{k-k'}{k_{N+1}-k'} \right\rceil \right) \tag{2b}$$

The procedure goes recursively until the key is found or the sub-array becomes zero. It has been proved that the mean value of probes for a successful interpolation search is:[14]

$$\log \log N(3) \tag{3}$$

where log is the base 2 logarithm.

We come now to our problem. In the two next sections the two variations of BIS are described and analysed. Note that we assume that the cost of sorting the M keys of the requested set is negligible.

2.1. First algorithm

The procedure begins as described above, but at the time when the first element of the set is found in the array we have additional information about the position of the successive elements: it is certain that they will lie to the right of the array index where the search stopped. Therefore the left sub-array may be exempted, and we continue searching for the rest of the elements in the right sub-array taken one by one in a sequential way.

In the following analysis we will use the fact that if a function f is concave downwards and S is a set of values, then

$$f(\underset{x \in S}{\text{average}}\,(x)) \leqslant \underset{x \in S}{\text{average}}\,(f(x)) \tag{4}$$

The relation (4) is proved in the appendix. To derive estimates for the expected value of the required probes to complete the search, we note that the expected width of the successively exempted array subintervals and the expected length of the shrinking array are given by equation:[11]

$$(N-M)/(M+1) \tag{5}$$

$$N_i = N-(i-1)\frac{N+1}{M+1} \tag{6}$$

where i is in the range $(1, M)$. By combining formulae (3) and (4) it is derived that the expected value of probes to perform a BIS of this kind is

$$\sum_{i=1}^{M} \log \log N_i \tag{7}$$

Since log is concave downwards the simplification is pessimistic, and the resulting formula is a close upper bound on the number of probes.

2.2. Second algorithm

Unlike the previous algorithm we do not start searching for the first element of the set, but we start choosing the middle element of the set. After the search of this element has terminated, the array has been divided into two sub-arrays (left and right). After the exclusion of the already-searched element the set has also been divided into two subsets (left and right), which contain equal numbers of elements or differ at most by one. Now the procedure continues recursively, for example the left (right) subset is searched in the left (right) sub-array, by choosing the middle element of each subset in a binary way.

Suppose that $M = 2{**}m-1+A$, where A and m are not negative integers and $0 \leqslant A < 2{**}m$. The first middle element has to be searched in the initial array of length N. Then the two new middle elements are drawn, and they have to be searched in the two sub-arrays of expected length $(N-1)/2$. The next four middle elements have to be searched in the four subarrays of expected length $((N-1)/2-1)/2 = (N-3)/4$. It is easy to derive that the expected number of probes required to perform the search is:

$$\sum_{i=1}^{m} 2^{i-1} \log \log \frac{N+1-2^{i-1}}{2^{i-1}} + A \log \log \frac{N+1-2^m}{2^m} \tag{8}$$

With a reasoning based on expression (4) it is evident that formula (8) is a close upper bound on the number of probes.

3. COMPARISON BETWEEN BIS AND BBS (BSS)

From Ref. 11 it is known that, when searching for a set of M elements in an array of length N, the expected value of probes, when BBS or BSS are applied, is:

$$\sum_{i=1}^{M} \lfloor \log N_i \rfloor + 1 + (\lfloor \log N_i \rfloor + 2 - 2^{\lfloor \log N_i \rfloor + 1})/N_i \tag{9}$$

$$(N+1)\frac{M}{M+1} \tag{10}$$

Therefore formulae (7) and (8) should be compared with formulae (9) and (10). Table 1 gives such a comparison by listing the cost per search for various values of N and M. It can easily be seen that: (*a*) for small values of M BIS improves interpolation search a little, (*b*) for all cases BISs outperform BBS, and (*c*) after a certain breakpoint BSS is preferable to BISs. These breakpoints may be calculated by solving the equations:

$$\sum_{i=1}^{M} \log \log N_i - (N+1)\frac{M}{M+1} = 0 \tag{11}$$

$$\sum_{i=1}^{m} 2^{i-1} \log \log \frac{N+1-2^{i-1}}{2^{i-1}} + A \log \log \frac{N+1-2^m}{2^m} = (N+1)\frac{M}{M+1} \tag{12}$$

These equations cannot be transformed to a closed-form formula for a given M. By applying the regression method to a sample of 20 equally distanced points in the

Table 1. Comparison between BSS, BBS and BISs

		M				
N	Search method	2	5	10	20	50
100	BSS	33.66	16.83	9.18	4.81	1.98
	BBS	5.49	5.13	4.91	4.75	4.61
	BISa	2.66	2.57	2.49	2.43	2.36
	BISb	2.61	2.43	2.21	1.92	1.46
1000	BSS	336.67	166.83	91.0	47.67	19.63
	BBS	8.73	8.35	8.11	7.93	7.78
	BISa	3.27	3.21	3.17	3.14	3.1
	BISb	3.24	3.13	3.0	2.84	2.58
10,000	BSS	3333.3	1666.83	909.18	476.24	196.1
	BBS	12.07	11.66	11.41	11.23	11.07
	BISa	3.7	3.65	3.62	3.6	3.58
	BISb	3.68	3.59	3.5	3.4	3.23

range (500, 10,000) the curves may be approximated by lines with equations:

$$M = 64.07 + 0.28N \quad (13)$$

$$M = 5.62 + 0.75N \quad (14)$$

The fact that the correlation coefficient – for both cases – is 0.999 demonstrates that the approximation is very accurate in this range. Fig. 1 presents the two lines described by equations (13) and (14).

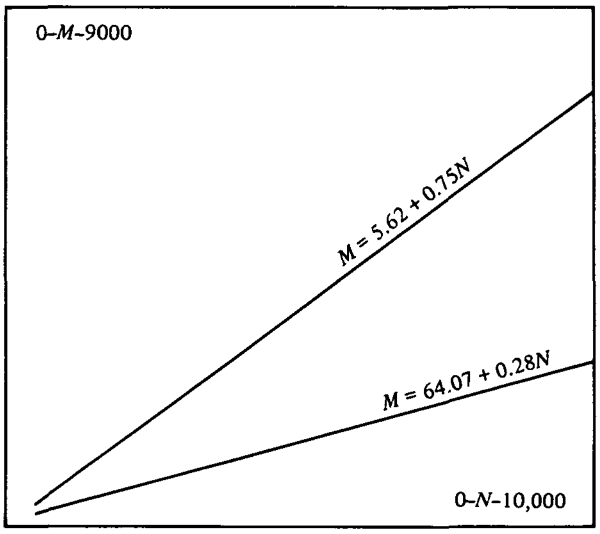

Fig. 1. Breakpoints between BISs and BSS.

4. DISCUSSION

Nowadays there is a trend towards bigger and cheaper main memory. Performance evaluation when searching in main memory becomes more interesting. We proposed and analysed two techniques – called Batched Interpolation Search or BIS – based on interpolation search, which answer the problem of searching efficiently for a set of *M* sorted elements in an array of length *N*, which is also sorted in the same order. It is assumed that the elements of the array and the set are drawn randomly from a uniform distribution. It is shown that interpolation search and BISs are efficient techniques for searching in main memory provided the cost metric is the number of probes to the array. BISs outperform Batched Binary Search (BBS) for all cases and Batched Sequential Search (BSS) up to two certain breakpoints, which are approximated by equations (13) and (14).

However, running time was not considered in the evaluation of the technique. It can be easily understood that the computation of the array index in the second algorithm of BIS is more expensive than that of the first algorithm of BIS, which in turn is more expensive than BBS and BSS, from the point of view of running time. Therefore, when we discuss main memory searching the computation required for BIS decreases the efficiency of the algorithm and therefore BBS becomes competitive to BIS.

The assumption of uniformity of attribute values in a file, as well as the assumption of uniformity of queries, have been proved to result in predicting only upper bounds of the expected cost.[3] Therefore, provided the keys of the array are uniformly distributed, if the *M* keys being searched for are not uniformly distributed, the binary form of BIS becomes more efficient. For example, suppose the middle key divides the array very unevenly. Then we search for half the keys in a short sub-array and half in a long sub-array. As an extreme case, with simple algebra, it can be proved that it takes much less time to search an array of length 1 and an array of length *N* than to search two arrays of length $N/2$. The worst case of this form of BIS is $O(N \log M)$, while without batching the worst case is $O(MN)$. In the case of not quite uniform distribution an approach like the ones in Refs 2 and 10, or simply combined binary and interpolation steps as in Ref. 15, could extend BIS.

Another possible extension could investigate the case that some of the *M* keys do not exist in the array. Then algorithm and analysis should be modified to cover this case. Also, when secondary storage is used the cost metric is the number of block accesses, while the required computation is negligible compared with the time needed to perform an access. An interesting exercise is to investigate the effect of having bigger or smaller blocks in the performance of interpolation search generally, and of BIS specifically. Intuitively, it seems that the probability that the next access will demand the same block is higher by applying BIS than BBS. In this way block accesses are saved and additional gain is achieved.

APPENDIX

Definition

A function f is concave downwards in the interval $J = [a, b]$ when:

$$f(\lambda x + (1-\lambda) y) \leqslant \lambda f(x) + (1-\lambda) f(y) \forall x, y \in J, x \neq y, \forall \lambda \in (0,1) \quad (15)$$

Lemma

If a function f is concave downwards in the interval $J = [a, b]$ then:

$$f\left(\frac{x_1 + \ldots + x_n}{n}\right) \leqslant \frac{f(x_1) + \ldots + f(x_n)}{n} \quad x_1, \ldots, x_n \in J \quad (16)$$

Proof

The proof follows by induction.

The equality holds for $n = i$. We accept that relation (16) holds for $n = k$:

$$f\left(\frac{1}{k}(x_1 + \ldots + x_k)\right) \leqslant \frac{1}{k}\left(f(x_1) + \ldots + f(x_k)\right) \quad x_1, \ldots, x_k \in J$$

We will prove that for $n = k+1$ the following relation holds:

$$f\left(\frac{1}{k+1}(x_1 + \ldots + x_{k+1})\right) \leqslant \frac{1}{k+1}\left(f(x_1) + \ldots + f(x_{k+1})\right)$$

Let

$$x = \frac{x_1 + \ldots + x_k}{k} = \frac{((x_1 + \ldots + x_k)/(k+1))}{(k/(k+1))} = \frac{((x_1 + \ldots + x_k)/(k+1))}{1 - (1/(k+1))}$$

Therefore

$$\begin{aligned} f\left(\frac{x_1 + \ldots + x_k + x_{k+1}}{k+1}\right) &= f\left(\frac{x_1 + \ldots + x_k}{k+1} + \frac{x_{k+1}}{k+1}\right) \\ &= f\left(x\left(1 - \frac{1}{k+1}\right) + x_{k+1}\frac{1}{k+1}\right) \\ &\leqslant \left(1 - \frac{1}{k+1}\right) f(x) + \frac{1}{k+1} f(x_{k+1}) \\ &\leqslant \left(1 - \frac{1}{k+1}\right)\frac{1}{k}(f(x_1) + \ldots + f(x_k)) + \frac{1}{k+1} f(x_{k+1}) \\ &= \frac{1}{k+1}(f(x_1) + \ldots + f(x_k)) + \frac{1}{k+1} f(x_{k+1}) \\ &= \frac{f(x_1) + \ldots + f(x_{k+1})}{k+1} \end{aligned}$$

Q.E.D.

REFERENCES

1. F. W. Burton and J. G. Kollias, Optimising disc head movements in secondary key retrievals. *The Computer Journal* **22** (3), 206–208 (1979).
2. F. W. Burton and G. N. Lewis, A robust variation of interpolation search. *Information Processing Letters* **10** (4), 198–201 (1980).
3. S. Christodoulakis, Implications of certain assumptions in database performance evaluation. *ACM Transactions on Database Systems* **9** (2), 163–186 (1984).
4. S. P. Ghosh and M. E. Senko, File organization: on the selection of random access index points for sequential files. *Journal of ACM* **16** (4), 569–579 (1969).
5. G. H. Gonnet and L. D. Rogers, The interpolation-sequential search algorithm. *Information Processing Letters* **6** (4), 136–139 (1977).
6. G. H. Gonnet and L. D. Rogers and J. A. George, An algorithmic and complexity analysis of interpolation search. *Acta Informatica* **13** (1), 39–52 (1980).
7. D. S. Hirschberg, On the complexity of search a set of vectors. *SIAM Journal on Computing* **9**(1), 126–129 (1980).
8. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, *Sorting and Searching*, pp. 406–419. Addison-Wesley, Reading, Massachusetts (1973).
9. J. G. Kollias, An estimate of seek time for batched searching of random or index sequential structured files. *The Computer Journal* **21** (2), 132–133 (1978).
10. G. N. Lewis, N. J. Boynton and F. W. Burton, Expected complexity of fast search with uniformly distributed data. *Information Processing Letters* **13** (1), 4–8 (1981).
11. Y. Manolopoulos, J. G. Kollias and M. Hatzopoulos, Sequential vs. Binary Batched Searching. *The Computer Journal*, **29** (4), 368–372 (1986).
12. M. V. D. Nat, On interpolation search. *Communications of ACM* **22** (12), 681 (1979).
13. Y. Perl and E. Reingold, Understanding the complexity of interpolation search. *Information Processing Letters* **6** (7), 219–222 (1977).
14. Y. Pearl, A. Itai and H. Avni, Interpolation search – a loglog N search. *Communications of ACM* **21** (7), 550–553 (1978).
15. N. Santoro and J. Sidney, Interpolation – binary search. *Information Processing Letters* **20** (5), 179–181 (1985).
16. B. Shneiderman, Polynomial search. *Software – Practice and Experience* **3** (1), 5–8 (1973).
17. B. Shneiderman, Jump searching: a fast sequential search technique. *Communications of ACM* **21** (10), 831–835 (1978).