18. Peterson, M., and Bulgren, W. Studies in Markov models of computer systems. Proc. 1975 ACM Annual Conf., Minneapolis, Minn., pp. 102-107.
19. Price, T.G. Models of multiprogrammed computer systems with I/O buffering. Proc. Fourth Texas Conf. on Comptng. Syst., U. of Texas at Austin, 1975.
20. Rose, C.A. Validation of a queueing model with classes of customers. Proc. Int. Symp. on Comptr. Performance Modeling, Measurement and Evaluation, Harvard U., Cambridge, Mass., March 1976, 318-325.
21. Sauer, C.H., and Chandy, K.M. Approximate analysis of central server models. *IBM J. Res. and Develop. 19*, 1, (Jan. 1975), pp. 301-313.
22. Sauer, C.H., and Chandy, K.M. Parametric modeling of multi-miniprocessor systems. IBM Res. Rep. RC5978, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., 1976.
23. Shedler, G.S. A cyclic queue model of a paging machine. IBM Res. Rep. RC2814, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., 1970.
24. Smith, W.L. Renewal theory and its ramifications. *J. Royal Statist. Soc. B20* (1958), 243-302.
25. Towsley, D. Local Balance Models of Computer Systems. Ph.D. Th., Dept. Comptr. Sci., U. of Texas at Austin, 1975.

Programming Techniques

S.L. Graham, R.L. Rivest Editors

Jump Searching: A Fast Sequential Search Technique

Ben Shneiderman
University of Maryland

When sequential file structures must be used and binary searching is not feasible, jump searching becomes an appealing alternative. This paper explores variants of the classic jump searching scheme where the optimum jump size is the square root of the number of records. Multiple level and variable size jump strategies are explored, appropriate applications are discussed and performance is evaluated.

Key Words and Phrases: jump searching, sequential files, file management, search strategies, database structures, index searching

CR Categories: 3.74, 4.34

1. Introduction

Locating a record with a given target key, or determining its absence from a file, is a central problem in file management. For sequentially organized files which are sorted by a single key field, Knuth [4] describes a variety of useful algorithms. If storage has been allocated contiguously, binary searching can be used to produce very good performance. Interpolation or curve fitting techniques based on linear or higher degree polynomials [10] have seen limited use because of the high cost of computation and uneven performance. Sequential searching is simple to program but is costly, when compared to binary searching. If records are placed in a random access memory and linked together with explicit pointers, the binary tree search technique or its variants [7] are preferred.

Author's address: Dept. of Information Systems Management, University of Maryland, College Park, MD 20742.

Some situations which require efficient sequential searching are unsuited to the binary search techniques. In such cases a jump search, or a "block search" as it has been called [5], may be appropriate. Jump search algorithms jump over portions of the sorted file until the search is localized to a small block or section of the file. Then smaller jumps may be taken or a sequential search is performed until the target key is located or its absence demonstrated. The traditional simple jump search algorithm requires jumps the size of the square root of the number of records in the file. We assume equal probability of request for all records. For example, in a file of one hundred records arranged in ascending order, the 10th, 20th, 30th, 40th . . . record keys are examined until the target key value is reached or exceeded. If the target value is bypassed, a sequential search of the remaining records is performed.

Jump searching is applied in a number of diverse situations. It is used to search the sequence sets in the lowest level of the indexes constructed by VSAM [8]. Jump searching is used because the keys have been compressed, producing variable length keys and preventing the use of binary search schemes. Pointers are used to jump over approximately $\sqrt{N}$ keys, where N is the number of keys in the sequence set.

For tape oriented searches where records may be blocked, jump searching is the natural form of processing a random request. The blocks are brought in and the key of the last record in the block is examined. Blocks are examined until the right one is found, then each record is examined. For this environment, where the cost of a jump differs from the cost of the sequential scan, the optimum jump size is $\sqrt{rN}$ where r is the ratio of the jump cost to the sequential scan cost.

In index searches, Maruyama and Smith [6] demonstrate that jump searching can be more efficient than binary searching when each node of the tree structured index has certain properties. Wagner [14] describes an "index record in which two or more levels are incorporated in a single record" when referring to a jumping strategy for index searching. Strong, Markowsky, and Chandra [13] describe a "family of *root search* strategies" in a comparison with B^*-trees and binary tree search for index page searching. Their square and cube root strategies correspond to the simple and two level simple jump searches in this paper.

In data communications, a variation of the jump searching scheme may be employed to decide how often to examine an ordered message stream when trying to locate a particular item. If the message stream contains 100 ordered records, then every tenth record should be examined to see if the expected message has arrived. If it has, a sequential scan of the last packet of messages is performed.

In summary, when binary searching can be done it is hard to do better, but some situations exist when jump searching is preferred. The most common situation is in sorted files which are linked to maintain a sequential ordering. This may be the case for highly volatile files or for variable length records. The next section compares performance of several variations on the jump searching theme.

2. Jump Searching Variations

2.1 Simple Jump Searching

Demonstrating the optimum jump size to be the square root of the number of keys can be done by writing down the average expected cost of the search, $C(N)$, over N records:

$$C(N) = \tfrac{1}{2}N/n + \tfrac{1}{2}(n - 1) \tag{1}$$

where n is the size of the jump. Assuming continuity of this function, taking the derivative with respect to n and setting the result to zero yields $n = \sqrt{N}$. Placing this result back into (1) gives the cost in terms of the number of keys examined as $\sqrt{N}$.

If the cost of a jump is a, and the cost of a sequential search is b, then the optimum jump size is $\sqrt{(a/b)N}$ and the search cost is $\sqrt{abN}$.

2.2 Two Level Simple Jump Searching

Recognizing that jump searching may be performed within a block leads to the easily programmable two level simple jump search, where the square root jump size is reapplied to the $(n - 1)$ records in a block. The expected number of keys that must be examined is $\frac{1}{2}\sqrt{N} + N^{1/4}$, which is about twice as fast as simple jump searching. Repeating this idea for more than two levels produces a decreasing benefit.

2.3 Two Level Fixed Jump Searching

If two levels of jumping are anticipated then it is no longer optimal to have the first level jumps as small as $\sqrt{N}$. If the first level jumps are of size n_1 and the second level jumps n_2 then the approximate search cost for a two level fixed jump search is given by:

$$C_2(N) = \tfrac{1}{2}N/n_1 + \tfrac{1}{2}n_1/n_2 + \tfrac{1}{2}n_2. \tag{2}$$

Taking partial derivatives with respect to n_1 and n_2, setting the results to zero and solving the pair of equations yields optimum fixed jump sizes of $n_1 = N^{2/3}$ and $n_2 = N^{1/3}$. Placing these results back in (2) gives the search cost as $C_2(N) = (3/2)N^{1/3}$.

Assuming the first level jumps cost a, the second level jumps b, and a sequential search c, then the optimum jump sizes are

$$n_1 = (N^2a^2/(bc))^{2/3} \quad \text{and} \quad n_2 = (Nab/c^2)^{1/3}$$

and the search cost is

$$C_{2abc}(N) = \tfrac{1}{2}N^{1/3}((bc/a^2)^{1/3} + (ac/b^2)^{1/3} + (ab/c^2)^{1/3}).$$

Of course this process can be repeated for an arbitrary number of levels, in which case the solution of the system of equations becomes increasingly complex [11]. If the number of levels becomes as large as $\log_2 N$, optimum jump searching is the same as binary searching.

2.4 Variable Jump Searching

In some applications of jump searching, it may not be more difficult to vary the size of successive jumps and to obtain a faster algorithm. Initially, this approach would have larger jumps than the simple jump search, but as the jumps neared the end of the file, the jumps would become smaller. As the size of the file left to search decreases so should the size of the jumps. If we specify the jump size as a function, $0 < f(N) < N$, of the *decreasing file size* N, then the average cost of the search can be written as a recurrence:

$$\begin{aligned} C_v(N) &= 1*(1/N) + (1 + \tfrac{1}{2}(f(N) - 1))*(f(N) - 1)/N \\ &\quad + (N - f(N))/N*(1 + C_v(N - f(N))) \\ &= 1 + (f^2(N) - f(N))/(2*N) \\ &\quad + (N - f(N))/N*C_v(N - f(N)). \end{aligned}$$

A numerical solution showed that the optimum jump size was approximately $\sqrt{2N}$. The precise determination of the function $f(N)$ which minimizes $C_v(N)$ was accomplished by James Spriggs of the University of Maryland Mathematics Department, who proved by induction that

$$C_v(N) = \frac{1}{N}\left(\frac{f(N)*(f(N) + 1)*(2f(N) + 1)}{6} + (f(N) + 1)*\left(N - \frac{f(N)*(f(N) + 1)}{2}\right)\right)$$

and that

$$f(N) = \lfloor \tfrac{1}{2}(\sqrt{8N + 1} - 1)\rfloor.$$

Since $f(N)*(f(N) + 1)$ is approximately $2N$, a good estimate for the cost is:

$$C_v(N) \approx \tfrac{1}{3}(2f(N) + 1) \approx \tfrac{1}{3}\sqrt{8N} + 1. \quad (2)$$

This solution has the pleasing property that the jump sizes are the inverse of the "triangle number" function. This integer function, derived from successive sums of the integers, follows the progression 1, 3, 6, 10, 15, 21, 28 ... and implies that if $N = 28$ the jump sizes are 7, 6, 5, 4, 3, 2, 1. Furthermore, the number of jumps, if we were to follow the jumps to the last key in the file, is given by the same function, $f(N)$.

Comparing formulas (1) and (2) reveals that variable jump searching yields a performance improvement of approximately six percent over simple jump searching; a modest but helpful margin if the implementation cost is reasonable. With contiguous array allocation there is some overhead in computing the jump sizes and with pointer-linked implementations there is a slight increase in the number of pointers required.

2.5 Two Level Variable Jump Searching

By repeating the variable jump searching process within the block which contains the target key, performance can be improved substantially. The cost function for a two level variable jump search can be written as:

$$\begin{aligned} C_{2v}(N) &= 1*(1/N) + (1 + C_v(g(N) - 1))*(g(N) \\ &\quad - 1)/N + (N - g(N))/N*(1 + C_{2v}(N - g(N))) \end{aligned}$$

where $0 < g(N) < N$ is the function which determines the jump size at the first level.

Although an analytic solution was not obtained, the conjectured solution was validated by implementation. For a two level variable jump search, the first level of jumps is determined by the inverse of the "tetragonal numbers" function and the second level jumps are carried out by the method of the previous section. The tetragonal numbers are the successive sums of the triangle numbers. Table I shows the values of the triangle and tetragonal numbers. Let $r(N)$ be the triangle number function, $s(N)$ be the tetragonal number function, and t: $r(N) \rightarrow s(N)$, that is, $t^{-1}(s(N)) = r(N)$. The first level of jumps are made according to $t^{-1}(N)$, which is what we called $g(N)$, and the second level of jumps are made according to $r^{-1}(N)$, which is what we called $f(N)$ earlier.

For example in an array of 120 values, if the target value were in position 77, it would be located after examination of positions 36, 64, 85, 70, 75, 79, 76, and 77 (jumps of 36, 28, 21, 6, 5, 4, 1, and 1).

The approximate cost of this algorithm can be expressed succinctly as

$$C_{2v}(N) \approx C_v(g(N)) + 1.$$

Two level variable jump searching produces performance superior to any of the other jump searches and can be competitive with binary searching. Of course, multiple level variable jump search strategies are possible.

Table II gives sample values for the five jump search algorithms presented in this paper.

3. Implementation Considerations and Applications

Jump searching is a useful technique because of its simplicity and efficiency in sequential searching when binary searching is unfeasible. A particularly appealing application would be for owner-member coupled sets of the data-structure-set approach described in the Data Base Task Group Report [1]. Owner records are linked to member records which are typically kept in ascending or descending order. Sequential order is often maintained by pointer links which may traverse pages of disk-based storage. Locating a specific member may require a costly search through multiple pages. With jump searching the number of pages brought into main memory may be reduced. With careful programming, insertions and deletions can be made without disrupting the jump pointers.

In some sequential files the records closest to the beginning of the file may be requested more often than others. This is typical in transaction oriented on-line

Table I. The triangle and tetragonal numbers.

N	$r(N)$ (triangle numbers)	$s(N)$ (tetragonal numbers)
1	1	1
2	3	4
3	6	10
4	10	20
5	15	35
6	21	56
7	28	84
8	36	120
9	45	165
10	55	220
11	66	286
12	78	364
13	91	455
14	105	560
15	120	680

Table II. Approximate average number of keys examined for five jump searching strategies.

No. keys in file	Simple	Two level simple	Two level fixed	Variable	Two level variable
50	7.1	6.2	5.5	6.7	5.2
100	10.0	8.2	7.0	9.4	6.3
500	22.4	15.9	11.9	21.1	10.3

systems where the most recently entered transactions are queried frequently. We assume records are ordered by date, descending. Variants of jump searching based on unequal probabilities of record requests will do well in such systems since the jumps can be chosen to match the distribution of record requests. If records at the end of the file are frequently requested, then the jump search can start from the end of the file and jump towards the beginning.

List merging algorithms can be improved by applying a jump search strategy. The first element of the shortest list is taken as a target key for a jump search through the longer list or lists. Since there is a high probability of locating the target key near the front of the longer list, jump search variants may have an advantage over classical sequential merge or binary merge techniques [3], when the number of comparisons is the criterion of performance.

Key compression algorithms [2] often require searches to begin at the start of the file and scan sequentially. By applying a jump search strategy, it is possible to provide improved search times with only a minor deterioration in the compression ratio.

4. Conclusions

Jump searching and its variants are useful algorithms for searching sequential files when binary searching is unfeasible. Variable jump searching is marginally better than simple and fixed jump searching. When the cost of a jump differs from the cost of a sequential scan, substantial performance improvements can be made by adjusting the jump size. Significant gains can be made by using a two level jumping process before a sequential scan is used.

Potential applications include scanning contiguous arrays, traversing pointer linked lists, searching through index blocks, list merging, and compressed key searches. Relationship to group testing algorithms should be studied [9] and performance when unequal probability of request is assumed or when requests are batched [12] should be examined.

Received May 1977; revised May 1978

References

1. CODASYL Data Base Task Group Report, April 1971. Available from ACM, New York.
2. Date, C.J. *An Introduction to Database Systems.* Addison-Wesley, Reading, Mass., 2nd Ed., 1977.
3. Hwang, F.K., and Lin, S. A simple algorithm for merging two disjointed linearly ordered sets. *SIAM J. Comptng. 1,* 1 (March 1972), 31–39.
4. Knuth, D. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
5. Martin, J. *Computer Data-Base Organization.* Prentice-Hall, Englewood Cliffs, N.J., 2nd Ed., 1977.
6. Maruyama, K., and Smith, S.E. Analysis of design alternatives for virtual memory indexes. *Comm. ACM 20,* 4 (April 1977), 245–254.
7. Nievergelt, J. Binary search trees and file organization. *ACM Computing Surveys 6,* 3 (Sept. 1974), 195–207.
8. OS/VS Virtual Storage Method (VSAM) Planning Guide. Order No. GC26-3799, IBM Corp., White Plains, N.Y.
9. Pippenger, N. Group testing. IBM Res. Rep. RC 6218, IBM T. J. Watson Res. Ctr., Yorktown Heights, N.Y., Sept. 1976.
10. Shneiderman, B. Polynomial search. *Software—Practice and Experience 3* (1973), 5–8.
11. Shneiderman, B. A model for optimizing indexed file structures. *Int. J. Comptr. and Inform. Sci. 3,* 1 (1974).
12. Shneiderman, B., and Goodman, V. Batched searching of sequential and tree structured files. *ACM Trans. Database Systems 1,* 3 (Sept. 1976), 268–275.
13. Strong, H.R., Markowsky, G., and Chandra, A.K. Searching within a page. IBM Res. Rep. RJ 2080, IBM Res. Lab., San Jose, Calif., Sept. 1977.
14. Wagner, R.E. Indexing design considerations. *IBM Syst. J. 10,* 4 (1973), 351–367.