SLAC-PUB-1651
September 1975

HEURISTICS FOR PARTIAL-MATCH RETRIEVAL DATA BASE DESIGN

Jon Louis Bentley*
Stanford Linear Accelerator Center
Stanford University
Stanford, California 94305

Walter A. Burkhard**
Computer Science Division
Department of Applied Physics
and Information Science
University of California, San Diego
La Jolla, California 92093

ABSTRACT

This paper considers several practical heuristics for the design of good, if not optimal, partial-match data bases in situations which are typical of real life information retrieval systems.

(Submitted to Information Processing Letters)

---

* Work supported in part by U.S. Energy Research and Development Administration under Contract E(04-3)-515.

** Work supported in part by National Science Foundation Grant GP-8557.

KEY WORDS AND KEY PHRASES

associative retrieval

algorithms

heuristic algorithms

partial-match retrieval

search tries

information retrieval

data base design

1. Introduction

This paper is concerned with information retrieval based on secondary keys, that is, keys which cannot in general uniquely identify a record but can indicate certain attributes of the associated records. Partial-match retrieval is concerned with accessing those records of a file which match the user's query, although his query may be only partially specified. In this paper we will consider specifically "real life" files which are stored as binary search tries (binary search tries are described in section 2.)

Much work has appeared recently on the problem of partial match searching. Rivest has studied hashing techniques for partial-match searching in [Ri74a-b,75]. He gives algorithms with desirable worst case performance and algorithms with optimal average case performance. Burkhard describes a class of hash functions with very good worst case performance in [Bu75a]. Rivest was the first to use tries for partial-match searching [Ri74b,75]. He studies only their average case behavior. Burkhard develops a class of tries with very good worst case performance in [Bu75b-c]. Independently, Dubost and Trousse discovered a similar scheme which they describe in [DuTr75].

Thus we see that much work has been done on the questions of worst case and average case performance (average is usually defined as all queries and all records are equally probable to occur.) In "real-life" applications, however, one is often interested in different measurements. Specifically, one is interested in fast average retrieval time with good worst case bounds given non-uniform data and queries. In this paper we develop heuristics for building search tries that will produce data bases

with such properties. This problem was designated by Rivest [Ri74b,75] as one of the main open problems in this area.

In section 2 we give the definitions and basic concepts necessary for reading this paper. Section 3 contains five heuristics we propose for binary search tries in "real-life" applications. We state our conclusions and directions open for further work in section 4.

2. Definitions

A record R is defined to be an ordered k-tuple $(r_1,r_2,\ldots,r_k)$ of values. Each coordinate of the k-tuple is referred to as a key and we assume that each key takes as value either 0 or 1 (the consequences of restriction to binary keys are discussed later in this section.) Let R_k denote the set of all valid records; the cardinality of R_k is 2^k. A file F is a subset of R_k; there are 2^{2^k} possible files. Every record in a file is required to have the same number of attributes; thus the discussion here is not amenable to files comprised of records with differing numbers of attributes as described by Hsiao and Harary [HsHa70].

Let Q denote the set of queries the information system is to handle. For a given file F and query $q \in Q$, the set $q(F)$ denotes the desired subset of records in F. Our interest centers on partial-match queries Q_t in which t keys are specified and $k-t$ keys are unspecified. The unspecified keys are replaced in the query by the special place-holding symbol "*". For partial-match query $q = (q_i,\ldots,q_k)$, $q(F)$ denotes the subset of records $R = (r_1,\ldots,r_k)$

in F such that $r_i = q_i$ if q_i is either a 0 or a 1 for $1 \leq i \leq k$. For example, the query $q = (1,*,0)$ requests all records that have $r_1 = 1$ and $r_3 = 0$, regardless of their r_2 values.

We will now give a formal definition of a partial-match binary search trie: the reader interested in more detail is referred to Rivest [Ri74b,75]. A trie is a binary tree such that

1. each leaf node corresponds to exactly one record and conversely,
2. each internal node specifies an attribute position j such that no other nodes on the path from the root to that node specify j, and
3. if a node specifies attribute j, then all nodes in that node's left subtrie have a 0 in attribute j and all nodes in its right subtrie have a 1 in attribute j.

A partial-match trie search algorithm is easy to define recursively. As the algorithm visits an internal node with attribute position j, it visits both sons recursively if q_j is an asterisk; otherwise it visits only the appropriate son. As the algorithm visits a leaf node it checks to see if the corresponding record answers the query description and if so reports it.

If the size of the file is relatively small compared to the main memory of the computer, then the trie could be stored entirely in the main memory. If the size of the file prohibits this then the internal nodes of the trie could be stored in the primary memory and the external nodes of the trie could be stored on some secondary memory, such as disk. If this is the case, then the leaf nodes usually correspond to more than one record: they are referred to as buckets

and are typically assigned enough records to occupy a substantial part of a physical record of the secondary memory.

We have assumed in this discussion that the keys of the records will be binary. This is a general assumption, as any other keys can be encoded in binary. Sometimes, however, that is not the proper approach to take. Bentley discusses a trie-like structure for non-binary partial match queries in [Be75]. Many of the ideas we discuss in this paper can be easily extended to non-binary keys.

The storage costs of tries is fairly small; this is especially true when buckets are used. For tries stored entirely in main memory, the cost of a search is usually the sum of internal and external nodes visited during the search. When the records are stored in a secondary memory the cost of transfer from secondary to main memory usually dominates the search time and the cost of the search is therefore taken to be the number of external nodes visited. We will not always mention which of the particular cost functions we are assuming in the following discussion since minimizing one usually minimizes the other.

3. Heuristics

In this section we present five heuristics for use in files stored as partial-match search tries.

3.1 Subtrie record descriptors

The contents of a subtrie are necessarily characterized by the

path from the root of the tree to the root of the subtrie--this is the defining property of binary search tries. This characterization is not necessarily the most complete possible due to the interdependencies of the keys of the records in the file. For example, in a file containing records of people, if the "over six feet tall" bit were on and the "husky build" bit were on, then the "under hundred pounds weight" bit would not be on. One approach to using this information is to describe each bit among the records in the subtrie as being all zeros, all ones, or mixed. This approach could be implemented easily using 2k bits at each internal node.

3.2 Unbalanced trie schemes

Because there are k positions in a query, any one of which could be either a 0,1 or *, there are 3^k possible partial-match queries. Given a collection of records under the assumption that any one of these queries is equally likely to be posed, the optimal trie is defined to be that which minimizes the average number of records examined per query. Since the optimal trie is merely that element of a finite set which minimizes an easily computable function, it is clearly well defined. However, it appears not to be easy to compute the optimal trie; Comer and Sethi [CoSe75] have recently shown that a related problem is NP-Complete. In this light we offer the following heuristic which will build good, if not optimal, search tries.

Given that all partial-match queries are equally likely to occur the probability that a given node on level ℓ will be examined is $\left(\frac{2}{3}\right)^{\ell}$ since ther are $2^{\ell}3^{k-\ell}$ queries of the possible 3^k that will

visit that node (assuming that heuristic 3.1 is not used.) For this reason the deeper a node is in the trie, the less often it is examined. Therefore, very unbalanced tries (which have many deep nodes) should have good average search times. (Rivest pointed this out in [Ri74b,75].)

Every bit position not yet tested in a search trie is a candidate for the next tested position. In building a trie for a particular file, we could maximize the trie unbalance at every node in hopes of attaining a globally unbalanced trie. For example, if we are given a collection of records to make into a trie we choose a bit position which is the most unbalanced value (highest ratio of zeros to ones or ones to zeros without being all of one value) to be the root attribute position. We recur for both subtries using the partitioned file of the root. This heuristic could be extended to be more global. In particular, we could attempt to maximize the unbalance for the selected root node by considering the unbalance for several levels of the trie. This "look ahead" helps avoid local optimization that is globally sub-optimal. This heuristic constructs a trie which is totally sensitive to the records in the file.

3.3 Query set schemes

In section 3.2 we assumed that all queries were equally likely to be posed. In most "real life" situations this is not the case--the queries are usually taken from a relatively small subset of all possible queries. It is desirable to take the actual queries into account as the trie is being constructed. We therefore propose the following heuristic algorithm to transform a set of queries into a skeletal trie

into which the records of a file could be inserted to form a binary trie as defined in section 2.

As a search procedure is traversing the trie each node visited will cause either one or two of its sons to be visited (with possible exception if the record descriptors of section 3.1 are employed.) Since two sons will be visited if and only if the query position is an asterisk it is desirable to use as attributes positions that have the smallest number of asterisks among the queries. The extension of this strategy to a recursive algorithm to build a trie given a set of sample queries is obvious. As in section 3.2, look ahead could be used to take a more global picture into account.

Proceeding formally, we can define the cost of a query in a given skeletal trie as the number of internal and external nodes visited by the search corresponding to the query. The cost of a skeletal trie is the sum of the costs of the queries in the trie. A recursive expression for the cost of a skeletal trie is the sum of the costs of its left and right subtries plus the sum of the number of queries of each external node in the two subtries. This expression suggests the heuristic noted above since using positions with many asterisks as attributes will increase the number of queries in external nodes, thereby increasing the cost of the trie.

3.4 Hybrid schemes

Here we propose that heuristics 3.2 and 3.3 be combined to produce a search trie which is sensitive to both the record structure and the query structure of the data base. A recursive approach to this problem

would choose as the attribute for a given internal node that untested bit position maximizing some function of the records and the queries in that node's subcollection. One possibility would be to choose as the root attribute that bit maximizing the product of the percentage of one bits in the records (or zero bits if more were specified) and the percentage of non don't care queries. As in sections 3.2 and 3.3 once a local optimization prescription is found it could be applied with look ahead to avoid global suboptimizations.

3.5 Self organizing files

Heuristic 3.4 uses both the initial file structure and the initial query structure to build a trie well suited to those conditions. It could happen that during the course of the user interaction the query structure changes or some records are retrieved with great frequency. Thus, there is a need for reorganization criteria which would be used to ensure that the data base will continue to achieve low average retrieval times. A counter scheme associating counters with each record and each internal node of the trie could be used to record query requests and record accesses. The most commonly accessed records and queries should be given greater weight during a file reorganization.

4. Conclusion

The main thrust in this paper has been to present certain heuristics computationally suitable for determining good (if not optimal) binary search tries. The approach has been descriptive rather than

analytic. The analyses for these heuristics are not currently available and offer very reasonable research projects. The heuristics evidently offer practical approaches to partial-match file design. The authors would certainly appreciate learning of other experiences with the heuristics mentioned here.

REFERENCES

[Be75] J. L. Bentley. Multidimensional binary search trees used for associative searching. To appear in Communications of the ACM.

[Bu75a] W. A. Burkhard. Partial-match queries and file design. To appear in Proceedings of the Conference on Very Large Data Bases, 1975.

[Bu75b] W. A. Burkhard. Hashing and trie algorithms for partial-match retrieval. Submitted to Communications of the ACM.

[Bu75c] W. A. Burkhard. Partial-match retrieval: performance bounds. UCSD Computer Science Division TR #3, 1975.

[CoSe75] D. Comer and R. Sethi. NP-completeness of tree structured index minimization. Technical report 167, Computer Science Department, The Pennsylvania State University, May 1975, 43 pp.

[DuTr75] P. Dubost and J.-M. Trousse. Software implementation of a new method of combinatorial hashing. TR# STAN-CS-75-511, Computer Science Department, Stanford University, 35 pp.

[HsHa70] D. Hsiao and F. Harary. A formal system for information retrieval from files. Communications of the ACM. Volume 13:2, 1970, 67-73.

[Ri74a] R. L. Rivest. On hash-coding algorithms for partial-match retrieval. Proceedings of the 15th Annual Symposium on Switching and Automata Theory, October 1974, 95-103.

[Ri74b] R. L. Rivest. Analysis of associative retrieval algorithms. Technical report STAN-CS-74-415, Computer Science Department, Stanford University, 1974, 102 pp.

[Ri75] R. L. Rivest. Partial-match retrieval algorithms. To appear in SIAM Computing Journal.