数理解析研究所講究録
第 495 巻 1983 年 221-247

COLORED TRIE SCHEMES FOR THE ADAPTIVE SEGMENTATION OF RELATIONAL FILES

Yuzuru TANAKA
Department of Electrical Engineering
Hokkaido University
Sapporo, 060 JAPAN

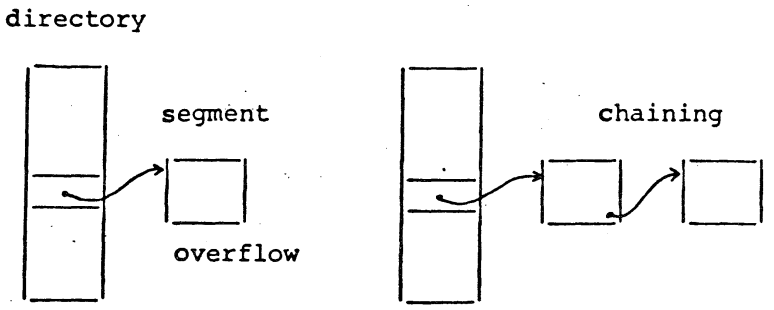
1. REVIEWS ON FILE SEGMENTATION SCHEMES

File segmentation is inevitable to cope with large files of information. If files are segmented arbitrarily, most queries require accesses to all the segments, which severely abates the system performance.

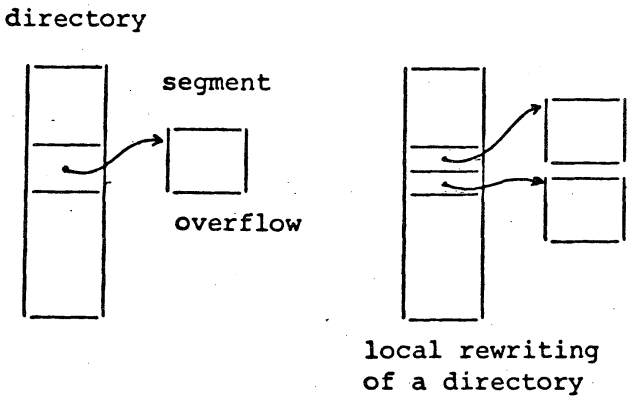
File segmentation schemes are clustering techniques that appropriately distribute records to segments so as to balance and minimize the number of segment accesses necessary to process various queries. Every segmentation scheme consists of two components, a directory and a set of segments. A directory is a set of rules that specifies how to distribute records to a set of segments. It may be represented by a hash function, a table, or a search tree. Every segment has the same finite size, and hence it may possibly overflow.

A segmentation scheme is static if its directory is apriori constructed based on an estimated probability distribution of record values. The overflow of segments does not change the directory. It is resolved by chaining a new

segment to the overflowing segment (Fig. 1 (a)). The increase of the file size may cause the excessive chaining of segments and severely increase the number of segment accesses necessary for the processing of each query. Excessive lowering of performance needs the reorganization of the whole file.



(a) A static segmentation scheme



(b) An adaptive segmentation scheme

Fig. 1 A static segmentation scheme and an adaptive segmentation scheme.

An adaptive segmentation scheme, however, does not presuppose the distribution of record values. When a segment overflows, the directory is locally rewritten to split this segment into two new segments so that the records in the old segment may be almost equally distributed to the two new segments (Fig. 1 (b)).

A file is said to be relational if it stores a relation defined by E.F.

Codd [1]. A primary key of a relational file is an attribute of the relation that uniquely specifies a record in the file. Some query may request a search based on the values of other attributes in the records. Such attributes are denoted by secondary keys. For the retrievals on a single key, whether it is prime or not, a lot of segmentation schemes have been proposed. Some of them have been practically used and approved. Among them are hashing as a static segmentation scheme, the B-trees of Bayer and McCreight [2], radix search trees (also known as tries) of Fredkin [3], expandable hashing of Knott [4], dynamic hashing of Larson [5], virtual hashing of Litwin [6], and extendible hashing of Fagin [7] as dynamic, or adaptive, segmentation schemes.

However, segmentation for the retrievals on secondary keys has not been much explored yet. Only several schemes are known as static schemes, and one as an adaptive scheme. Static schemes are essentially classified into two schemes, a combinatorial hashing scheme of Rivest [8] and a balanced filing scheme of Abraham [9]. These schemes can be applicable to restricted cases in which the number of segments and the number of secondary keys have some special relationship. Besides, their directories can not be adaptively rewritten.

An extended k-d tree of Chang [10] is an only known adaptive segmentation scheme for retrievals on secondary keys. It is an extension of a k-d tree of Bently [11] that was originally a search tree whose leaf has a single record. A k-d tree is a binary search tree except that one of the secondary key attributes is assigned to each of its levels. Each internal node splits a set of records into two subsets by the values of the attribute assigned to the level of this node, i.e., a set of records with the smaller attribute values, and a set of records with the larger attribute values. The splitting value of the attribute can be arbitrarily chosen at each node.

An extended k-d tree scheme has several disadvantages. The removal of the restriction that the secondary keys used for the segment splitting should be

fixed at each level of the tree may decrease the average or the maximum number of segment accesses necessary for query processing. Actually, a k-d tree does not ensure the minimization of them. Besides, the scheme does not specify how to analyze the value distribution of records in an overflowing segment. Generally, an overhead it causes is inevitable.

This paper proposes a new segmentation scheme classified into the same type as an extended k-d tree scheme, namely it falls into adaptive segmentation schemes for retrievals on secondary keys. Its directory is represented by a binary trie whose node is labeled with one of the secondary keys. Different from an extended k-d tree, its node is labeled with an arbitrary secondary key. The splitting of segments is based upon the values of a certain bit of this attribute values, and hence, it can be arbitrarily chosen either to minimize the average number of segment accesses or to improve the worst case performance. Besides, if the values of secondary keys are independently and uniformly distributed, a search of the directory for N segments and its local rewriting need only O(log N) time for large N.

2. COLORED TRIE SCHEMES FOR RELATIONAL FILES

2.1. Abstract modeling of a relational file segmentation problem

Suppose first that we have a relational file of records each containing n secondary keys, where each secondary key has a fairly large number of possible values. We can map the records whose secondary keys are $(k_1, k_2, \ldots, k_n)$ to the (n*m)-bit number

$$h_1(k_1)h_2(k_2)\ldots h_n(k_n),$$

where each h_i is a hash function taking each secondary key into an m-bit value, and the above expression stands for the juxtaposition of n m-bit values.

Now the above segmentation of a relational file is defined in an abstract manner as follows. Suppose that we have a lot of beeds each colored with one of the different colors, $c_0, c_1, \ldots, c_{n-1}$. This set of colors is denoted by C. A bead with c_i color is referred to as a c_i-bead. Each bead is labeled with an m-bit value. There may be beads with a same color and a same label. A rosary is a string of n beads each having a different color. The c-label of a rosary is defined as the label on its c-bead.

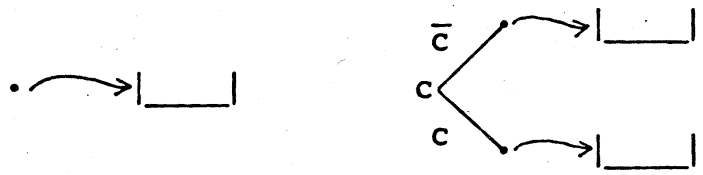
Rosaries are made one by one, choosing an arbitrary label for each color. They are stored in a set of drawers each having a constant capacity. While the number of produced rosaries is less than the capacity of a drawer, they are all stored in a single drawer. If it overflows, the rosaries stored in it are appropriately distributed into two new empty drawers. The number of drawers used to store rosaries is increased by one.

Suppose that each customer requests a search for rosaries with a specified label v on a bead with a specified color c. This request is expressed by c=v. In order to decrease the wait time of customers, rosaries should have been appropriately distributed into a set of drawers. The wait time is proportional to the number of drawers to be searched. This number varies for various colors and labels. If we desire to minimize the maximum responce time, the maximum number of drawers to be searched should be minimized. If the throughput of services is desired to be maximized, the average number of drawers to be searched should be minimized.

2.2. A colored binary trie

Initially, only a single drawer is used to store rosaries, and hence its

directory has only one entry (Fig. 2 (a)). If an overflow occurs, the rosaries in the drawer should be divided into two classes. They can be divided based upon the values of a certain bit of a certain color label. For this division, we use the most significant bit of some color label. The directory will come to have two entries corresponding to two new drawers that store the two classes. It can be represented as a binary trie with two leaves and a root that is painted with a color whose labels are used as a basis of the division (Fig. 2 (b)). If one of the drawers overflows again, its contents are further divided into two classes. In general, the division of a cluster can be based upon any bit of any color label that is not already used as a basis of another division in the process of having produced this cluster. We use, in every division, the leftmost unused bit of some color label. The directory of drawers that describes the rules of cluster division can be represented as a colored binary trie. It is a binary trie whose each internal node is painted with one of the n colors.



a directory a drawer — a directory drawers

(a) initial state — (b) after the split by a color C

Fig. 2 Division of the contents of a drawer based on a bit of the C-labels of rosaries.

A left branch from a node colored with c is represented by $\bar{c}$, while the

right by c. The concatenation of the representation of branches along the path from the root to a node uniquely identifies that node in the trie. This identifier is referred to as a node code. For a node with a node code α and each color c, we define the c-code of that node as a bit sequence that is obtained by first deleting all the appearances of c' and $\bar{c}$' from α for each c' different from c, and then replacing c and $\bar{c}$ respectively with '1' and '0'. The c-code of a node with a node code α is denoted by $c(\alpha)$, while the length of α and that of $c(\alpha)$ are respectively represented by $\rho(\alpha)$ and $\rho(c(\alpha))$. A node α of a colored binary trie stands for a cluster of rosaries whose c-labels begin with $c(\alpha)$ for each color c.

2.3. Access cost

Each customer requests a search based on the values of some specified c-label. It requires a search of a directory for drawers necessary to search, and searches of these drawers for requested rosaries. The wait time of a customer is approximately proportional to the number of drawers to be searched. Fig. 3

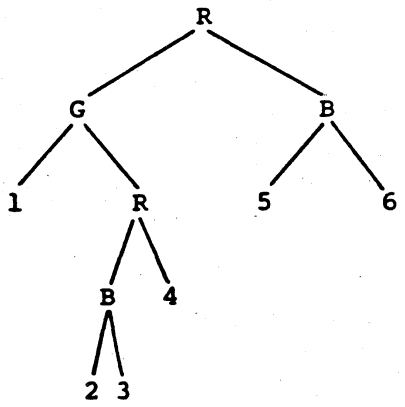


Fig. 3 An example directory represented by a colored binary trie with 3 colors.

shows an example directory represented by a colored binary trie with three colors, R, G, and B. They are labeled with the numbering from 1 to 6. Let the search for rosaries with a c-label v be referred to as a 'c=v' search. For the search R=00...00, it is necessary to pull out three drawers 1, 2, and 3. For R=00...01, a set of necessary drawers is same. Generally, these drawers are necessary and sufficient to search for the R-labels beginning with 00. These search requests are represented by R=00**...*, where '*' stands for an arbitrary binary value. A search request B=0**...* requires to pull out four drawers, 1, 2, 4, and 5. The number of necessary drawers varies depending on the color c and its label v. This number is denoted by naccess(T, c, v), where T denotes a directory trie.

Let Cavg(T, c) and Cworst(T, c) respectively denote the average and the maximum number of segment accesses necessary for searches based on the values of the c-label, i.e.,

$$Cavg(T, c) = \underset{v\in\{0,1\}^m}{\text{average}} (naccess(T, c, v))$$

$$Cworst(T, c) = \max_{v\in\{0,1\}^m} (naccess(T, c, v)).$$

Two kinds of access costs can be defined:

1. average cost

$$cost^1(T) = \underset{c\in C}{\text{average}} (Cavg(T, c)),$$

2. worst cost

$$cost^2(T) = \max_{c\in C} (Cworst(T, c)).$$

Suppose that we have a directory T and that one of its drawers overflows. We want to choose the most desirable color to split the overflowing leaf of T so that the result trie may have the least cost. Suppose that the overflow occurs at a leaf with a node code α. Let a trie obtained by splitting this leaf based on a bit of c-label be denoted by new(T, α, c). The most desirable color is

formally defined as the one that minimizes

$$Ccost^{i}_{T,\alpha}(c) = cost^{i}(new(T, \alpha, c)).$$

There can be three different schemes corresponding to the two cost functions. The best average scheme minimizes $Ccost^{1}_{T,\alpha}(c)$, while the best worst scheme minimizes $Ccost^{2}_{T,\alpha}(c)$. The best average scheme results in a good performance throughput, while the best worst scheme improves responce time.

3. ADAPTIVE SEGMENTATION SCHEMES

3.1. Best average scheme

For a colored trie T and an overflowing leaf α, $Ccost^{1}_{T,\alpha}(c)$ is calculated as follows (Appendix A.1).

Theorem 3.1

$$Ccost^{1}_{T,\alpha}(c) = cost^{1}(T)+(1/n)\ sum\ (1/2)^{\rho(c'(\alpha))}-(1/n)(1/2)^{\rho(c(\alpha))}. \quad (1)$$

The first two terms in the right hand side of the equation (1) do not depend on c. Besides, the equality (1) implies that the minimization of $Ccost^{1}_{T,\alpha}(c)$ is equivalent to the minimization of $\rho(c(\alpha))$. Therefore, in the best average scheme, the split of a leaf with a node code α should choose a color that minimizes $\rho(c(\alpha))$. As a special case of this scheme, a scheme is a best average scheme if it selects, for the splitting of a node at the i-th level, the color c_j whose suffix j is congruent to i-1 modulo n. Such a scheme is called a regular best average scheme. We show an example trie in Fig. 4 that is built based on the regular best average scheme. If the node 2 overflows, it will be

split by the color 'b'.

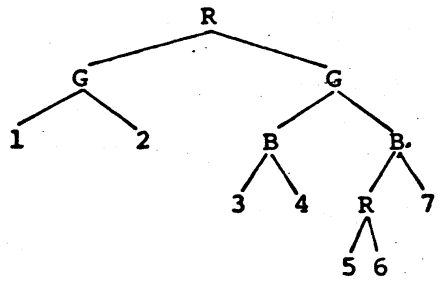


Fig. 4 A colored binary trie with 3 colors that is built based on the best average scheme.

3.2 Best worst scheme

When an overflow occurs at some leaf of a colored trie, the best worst scheme splits this leaf by such a color that minimizes $Ccost^{3}_{T,\alpha}(c)$. If both c' and c'' are different from c then Cworst(new(T, α, c'), c) is equal to Cworst(new(T, α, c''), c). Let c^{+} denote a representative of the colors that are different from c. Then the following theorem holds.

Theorem 3.2 (Appendix A.2)
If a color c maximizes Cworst(new(T, α, c^{+}), c) then it minimizes $Ccost^{2}_{T,\alpha}(c)$.

For each color c, the leaves of a colored trie T can be divided into three classes. The first class consists of those leaves whose node codes do not include c nor $\bar{c}$. A subtree of T that consists of these leaves is denoted by T/c. The second class consists of such leaves whose each node code has $\bar{c}$ that is not preceded by c. A subtree of T consisting of these leaves is denoted by $T|\bar{c}$. Each of the remaining leaves has a node code that has c before any

appearances of $\bar{c}$ in it. A subtree consisting of these remaining leaves is denoted by T|c. An example of a trie T, and its $T|\bar{c}$, T|c and T/c are shown in Fig. 5. Then Cworst(T, c) is recursively calculated as follows.

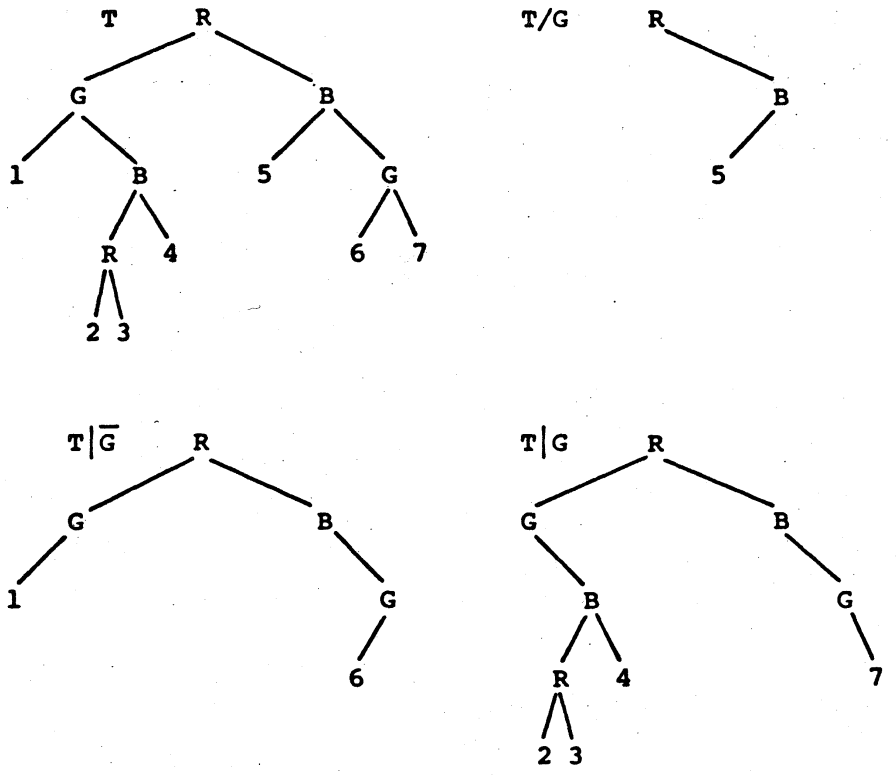


Fig. 5 An example of a colored trie, and its T/G, $T|\bar{G}$, and T|G

Theorem 3.10 (Appendix A.3)

Cworst(T, c)

= card(leaves(T/c)) + max (Cworst($T|\bar{c}$, c), Cworst(T|c, c)),

where leaves(T) denotes a set of the leaves of T while card(S) denotes the cardinality of a set S.

For each finite binary sequence over {0,1}, let $T|_v^c$ denote

$((...((T|c_1)|c_2)...)|c_{k-1})|c_k$,

where k is the length of v, and c_i is c (, $\bar{c}$) if the i-th bit of v is 1 (, 0). We define a set P(v) for a finite binary sequence v as a set of prefixes of v, i.e.,

$$P(v) = \{ u \mid \text{there exists } u' \text{ in } \{0,1\}^*, \text{ and } u \circ u' = v \}.$$

Let $L_c(v)$ and $W_c(v)$ be defined as

$$L_c(v) = \text{card}(\text{leaves}((T|_v^c)/c)),$$

$$W_c(v) = \text{Cworst}(T|_v^c, c).$$

Then they satisfy the following relation;

$$W_c(v) = L_c(v) + \max (W_c(v \circ 0), W_c(v \circ 1)).$$

Let S_c be

$$S_c = \{ c(\alpha) \mid \alpha \in \text{leaves}(T) \}.$$

The algorithm for the best worst scheme is stated as follows.

Algorithm

1. Compute Cworst(new(T, α, c^+), c) for each c.

Let $L_c^\alpha(v)$ and $W_c^\alpha(v)$ be defined as

$$L_c^\alpha(v) = \text{card}(\text{leaves}(\text{new}(T, \alpha, c^+)|_v^c)/c)),$$

$$W_c^\alpha(v) = \text{Cworst}(\text{new}(T, \alpha, c^+)|_v^c, c).$$

Then $W_c^\alpha(\epsilon)$ is Cworst(new(T, α, c^+), c), where ϵ denotes an empty string. Since $\text{new}(T, \alpha, c^+)|_v^c$ is $T|_v^c$ for any $v \notin P(c(\alpha))$, $L_c^\alpha(v)$ and $W_c^\alpha(v)$ are equal to $L_c(v)$ and $W_c(v)$ for such v. For $v \in P(c(\alpha))$, $L_c(v)$ and $W_c(v)$ may have to be updated. Since the leaf α is also a leaf of $T|_{c(\alpha)}^c$, and α is split into two leaves by a color different from c, the number of leaves of $\text{new}(T, \alpha, c^+)|_{c(\alpha)}^c$ increases from that of $T|_{c(\alpha)}^c$, by one. Therefore, $L_c^\alpha(v)$ is

$$L_c^\alpha(v) = \text{if } v=c(\alpha) \text{ then } L_c(v)+1$$
$$\text{else } L_c(v).$$

The computation of $W_c^\alpha(\epsilon)$ is recursively performed by the following formula;

$$W_c^{\alpha}(v) = \text{if } v\in S_c \text{ then } 0$$
$$\text{elseif } v\notin P(c(\alpha)) \text{ then } W_c(v)$$
$$\text{else } L_c^{\alpha}(v)+\max(W_c^{\alpha}(v\circ 0),\ W_c^{\alpha}(v\circ 1)). \qquad (2)$$

Since $v\circ 0$ and $v\circ 1$ can not simultaneously belong to $P(c(\alpha))$, only one of $W_c(v\circ 0)$ and $W_c(v\circ 1)$ needs further computation in (2). Therefore, the number of steps necessary for the computation of $W_c(\alpha)$ is proportional to the length of $c(\alpha)$, which is bounded by the height of the colored trie.

2. Choose a color c that maximizes Cworst(new(T, α, c^+), c).

Since Cworst(new(T, α, c^+), c) is $W_c(\epsilon)$, what we have to do is to find out a color that maximizes $W_c(\epsilon)$. If there are more than one candidates, choose one that minimizes $\rho(c(\alpha))$.

3. Split the overflowing node by the selected color c_0, and update $L_c(v)$ and $W_c(v)$ for each c and $v\in P(c(\alpha))$.

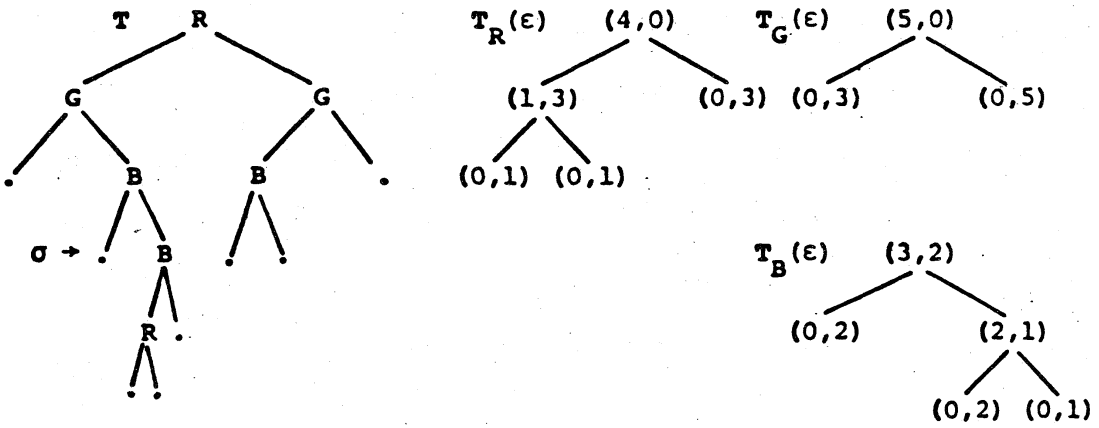
For any c different from c_0, and any $v\in P(c(\alpha))$,

$$L_c(v) \leftarrow L_c^{\alpha}(v),$$
$$W_c(v) \leftarrow W_c^{\alpha}(v).$$

For $c=c_0$,

$$S_c^{new} \leftarrow (S_c-\{c(\alpha)\}) \cup \{c(\alpha)\circ 0,\ c(\alpha)\circ 1\},$$
$$L_c^{new}(v) \leftarrow \text{if } v=c(\alpha) \text{ then } L_c(v)-1$$
$$\text{elseif } v=c(\alpha)\circ 0 \text{ or } v=c(\alpha)\circ 1$$
$$\text{then } L_c(v)+1$$
$$\text{else } L_c(v).$$
$$W_c^{new}(v) \leftarrow \text{if } v\in S_c^{new} \text{ then } 0$$
$$\text{elseif } v\notin P(c(\alpha))$$
$$\text{then } W_c(v)$$
$$\text{else } L_c^{new}(v)+\max(W_c^{new}(v\circ 0),\ W_c^{new}(v\circ 1)).$$

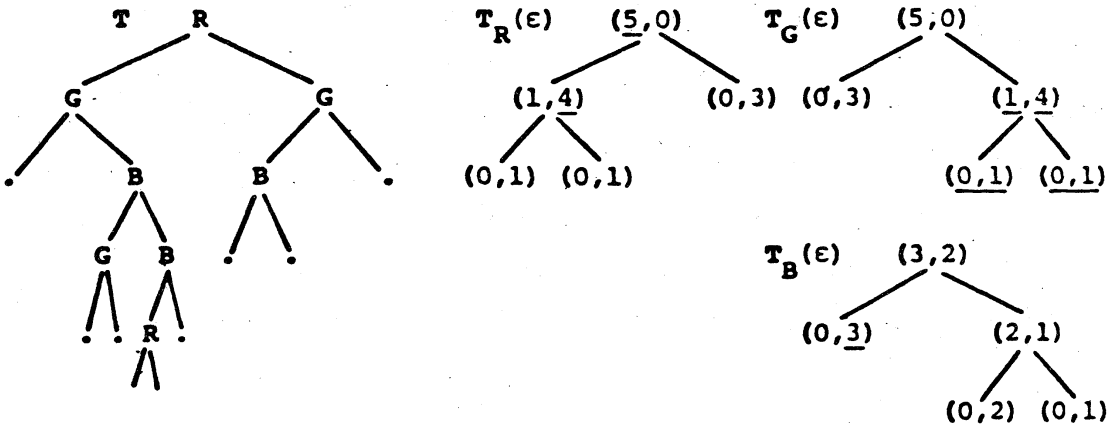
Because of the same reason, the number of steps necessary for the update of $L_c(v)$ and $W_c(v)$ is proportional to the length of $c(\alpha)$, which is bounded by the



(a) a colored trie and its $T_C(\varepsilon)$'s

	R	G	B
Cworst(new(T, σ, C†), C)	5	<u>6</u>	5

(b) selection of a color that minimizes $Ccost^3_{T,\sigma}(C)$



(c) a updated colored trie and its $T_C(\varepsilon)$'s

Fig. 6 The splitting of an overflowing segment based on the best worst scheme

height of the colored trie.

For each $v \in \{c(\alpha) \mid \alpha$ is a node in T.$\}$, let $T_c(v)$ be a tree defined recursively as follows. Its root is labeled with a pair $(W_c(v)-L_c(v), L_c(v))$. Its left and right subtrees are respectively defined as $T_c(v \circ 0)$ and $T_c(v \circ 1)$. For $v \notin \{c(\alpha) \mid \alpha$ is a node on T.$\}$, $T_c(v)$ is considered empty. Fig. 6 (a) shows an example colored trie and its corresponding $T_c(\epsilon)$ for each color c. The overflowing node is indicated by an arrow. For the selection of a color that minimizes $Ccost^3_{T,\alpha}(c)$, the value of Cworst(new(T, α, c^+), c) is computed for each c, which is shown in (b). In this case, G is selected. For each c, the change of T_c by the splitting is shown in (c).

4. ANALYSIS OF COLORED BINARY TRIES

4.1. Theoretical analysis

It is well known that the average height of a randomly constructed binary tree with N external nodes is $2*\ln(N-1) \fallingdotseq 1.386 \log_2(N-1)$ [12]. Since a colored binary trie is also a binary tree, the average height of randomly constructed colored binary tries with N leaves is also $1.386 \log_2(N-1)$.

For the regular best average scheme, we can get a fairly good lower bound and a upper bound of $cost^1(T)$.

Theorem 4.1 (Y. Tanaka [13])
The regular best average scheme makes $cost^1(T)$ as

$$cost^1(T) \leq 2^{(n+1)/n} N^{(n-1)/n}.$$

Theorem 4.2 (Y. Tanaka [13])

The regular best average scheme makes $cost^1(T)$ as

$$cost^1(T) \geq \frac{A\ (1+A^{N-2}-2A^{N-1})}{4(1-A)} \qquad (N \leq n*m),$$

$$cost^1(T) \geq \frac{A\ (1+A^{n*m-2}-2A^{n*m-1})}{4(1-A)} \fallingdotseq \frac{A}{4(1-A)} \qquad (N \geq n*m),$$

where $A = (1/2)^{1/n}$. Especially, if the label values on rosaries are uniformly distributed, $cost^1(T)$ is bounded as

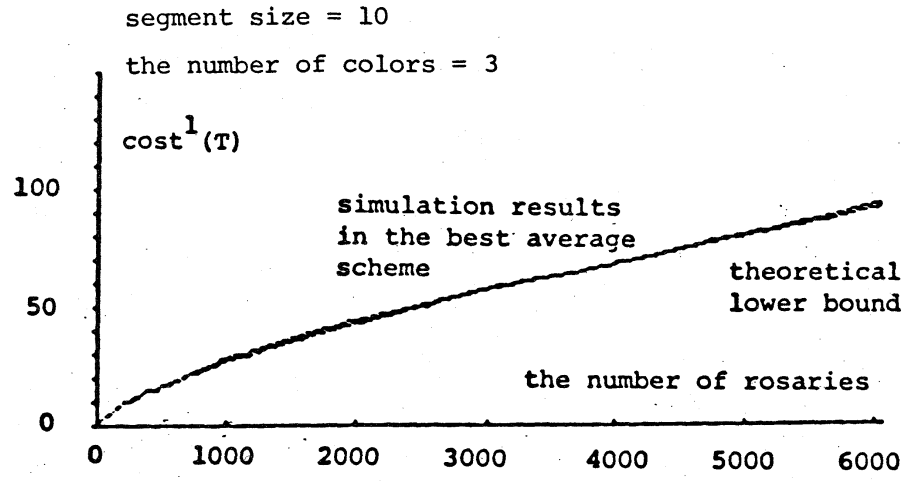
$$cost^1(T) \geq N^{(n-1)/n}.$$

4.2. Experimental analysis

The experimental analysis by computer simulations has shown desirable features of these proposed schemes.

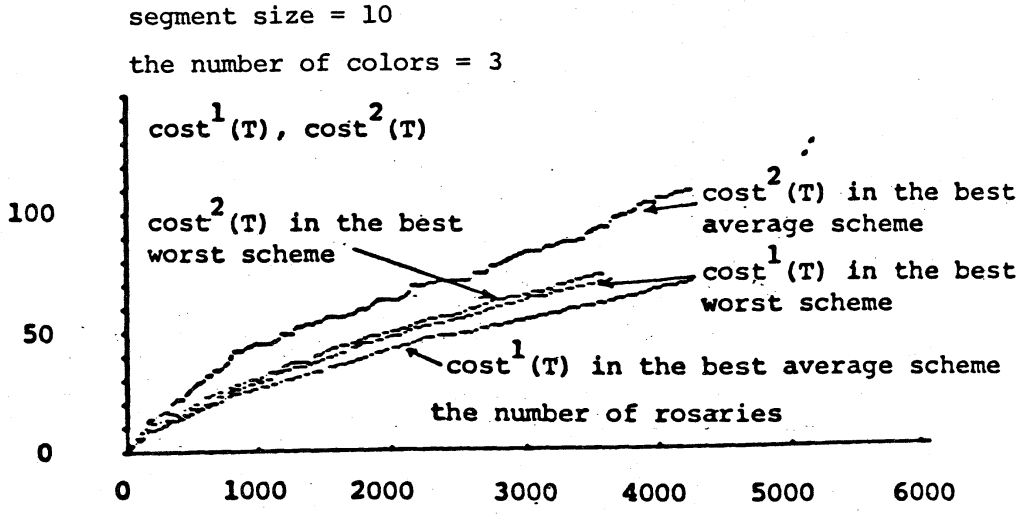
In the best average scheme, the loading factor of a segment is about 70 %, which does not depend on the size of segments, the number of different colors, nor the distribution of the label values on rosaries. Theoretically, if the label values on rosaries are uniformly distributed, the lower bound of $cost^1(T)$ becomes $N^{(n-1)/n}$, which can not be crossed by any segmentation scheme regardless of whether it is static or adaptive. Fig. 7 (a) shows the simulated values of $cost^1(T)$ in the case of the best average scheme together with the theoretical lower bound. The two curves coincide almost everywhere. The average cost $cost^1(T)$ is almost independent from the distribution of the label values on rosaries. However, the worst cost $cost^2(T)$ seriously depends on how the distribution of the label values deviates from uniform distribution.

For a trie constructed by the best worst scheme, both the average cost $cost^1(T)$ and the worst cost $cost^2(T)$ are almost independent from the value distribution (Fig. 7 (b)). Besides, the difference between $cost^1(T)$ and



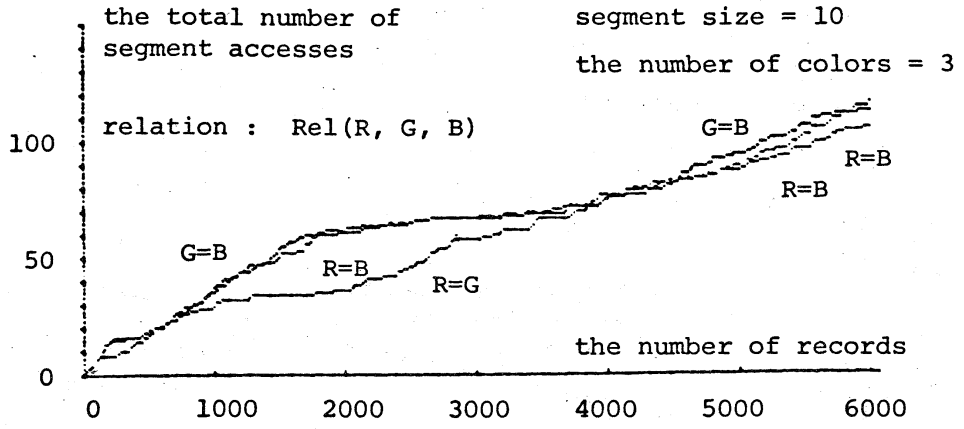
(a) the simulated average number of segment accesses in the best average scheme together with its theoretical lower bound.

(The loading factor is assumed to be 70 % in the computation of the theoretical lower bound.)

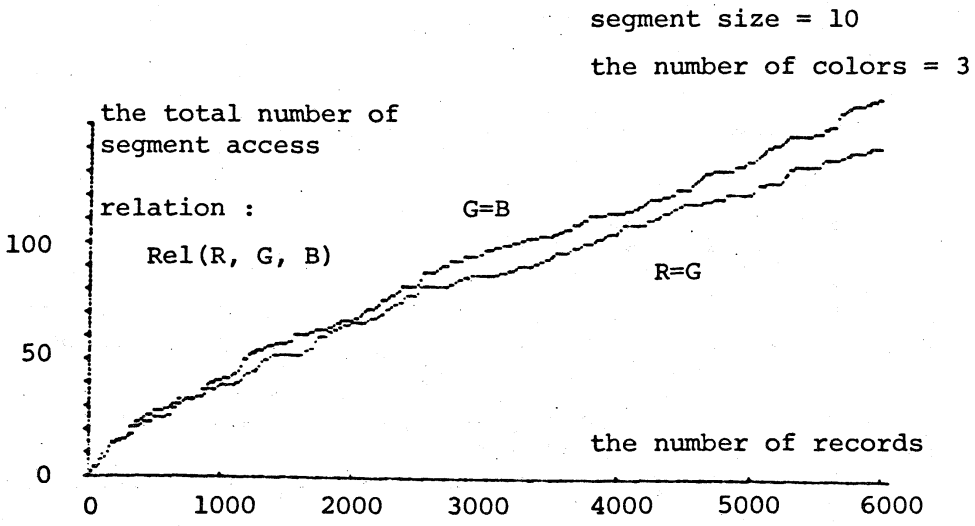


(b) comparison of the two schemes, the best average scheme and the best worst scheme, in the case in which record values are not uniformly distributed.

Fig. 7 Experimental analysis of the proposed segementation schemes.



(a) The total number of segment accesses necessary for the processing of a restriction operation in the case of the best average scheme.



(b) The total number of segment accesses necessary for the processing of a restriction operation in the case of the best worst scheme.

Fig. 8 The improvement of the file access cost in the processing of a restriction operation by using our new adaptive segmentation schemes.

$cost^2(T)$ is very small. The loading factor of a segment is almost same as in the case of the best average scheme.

The two cost functions above correspond to the average and the maximum number of segment accesses necessary for the processing of selection operations on a relation, such as R[A='v'], where R, A, and v are respectively a relation, an attribute of R, and an attribute value of A. We have also simulated our file segmentation schemes to know the number of necessary segment accesses in the processing of a restriction operation on a relation, such as R[A=B], where A and B are two different attributes of a relation R (Fig. 8). The figure shows that the access cost of restriction operations is mostly similar to the cost of selection operations. Although our schemes are designed to decrease the access cost of selection operations, they have also shown a remarkable improvement in the access cost of restriction operations.

5. CONCLUSION

We have proposed new adaptive segmentation schemes for the retrievals on secondary keys. They have the following advantageous features.

1. It is completely adaptive, and has no restrictions on the number of segments and of attributes.
2. It can be arbitrarily chosen either to minimize the average number of segment accesses or to improve the worst case performance. This property is different from an extended k-d tree scheme, which can minimize only the average. Besides, the minimization in an extended k-d tree scheme is performed under the restriction that the node splitting at each level uses a same secondary key.

Our new scheme assumes no such restrictions.

3. If the values of the secondary keys are independently and uniformly distributed, a search of the directory for N segments and its local rewriting need only O(log N) time for large N.

The computer simulations have shown various desirable features of these schemes. Among them, the following features are worth mentioning.

1. The loading factor is bout 70 %, which is fairly good.
2. If the record values are uniformly distributed, the expected number of segment acesses in the best average scheme almost coincides with the lower bound of the average cost.
3. In the best worst scheme, the responce time is almost independent from the distribution of record values. Besides, the maximum number of segment accesses becomes very close to the expected number of segment accesses.

All of these desirable features of our schemes shows their applicability to the practical relational files and also to the large relational database machine.

Acknowledgement

The author wishes to express his thanks to Prof. R. Tagawa of Hokkaido University for his support on this research, and my student Mr. N. Mori for his help in the computer simulations.

REFERENCES

[1] Codd, E.F., 'A relational model for large shared data banks,' Comm. ACM, Vol. 13 No. 6, 1970, pp. 377-387.

[2] Bayer, R. and E. McCreight, 'Organization and maintenance of large unordered indices,' Acta Informatica, Vol. 1 Fasc. 3, 1972, pp. 173-189.

[3] Fredkin, E., 'TRIE memory,' CACM, Vol. 3 NO. 9, 1960, pp. 490-499.

[4] Knott, G.D., 'Expandable open addressing hash table storage and retrieval,' Proc. ACM SIGFIDET Workshop on Data Description, Access and Control, 1971, pp. 186-206.

[5] Larson, P., 'Dynamic hashing,' BIT, Vol. 18, 1978, pp. 184-201.

[6] Litwin, W., 'Virtual hashing: A dynamically changing hashing,' Proc. Very Large Data Bases Conf. Berlin, 1978, pp. 517-523.

[7] Fagin, R., J. Nievergelt, N. Pippenger, and H.R. Strong, 'Extendible hashing--a fast access method for dynamic files,' ACM Transaction on Database Systems, Vol. 4 No. 3, 1979, pp. 315-344.

[8] Rivest, R.L., 'Partial-match retrieval algorithm,' SIAM J. Comput., Vol. 5 No. 1, 1976, pp. 19-50.

[9] Abraham, C.T., S.P. Ghosh, and D.K. Ray-Chaudhuri, 'File organization schemes based on finite geometries,' Information and Control, Vol. 12, 1968, pp. 143-163.

[10] Chang, J., and K. Fu, 'Extended k-d tree database organization: A dynamic multiattribute clustering method,' IEEE Trans. Software Engineering, Vol. SE-7, No. 3, 1981, pp. 284-290.

[11] Bentley, J.L., 'Multi-dimensional binary search trees used for associative searching,' Comm. ACM, Vol. 18, 1975.

[12] Knuth, D.E., The Art of Computer Programming, Vol. 3 / Sorting and Searching, Addison-Wesley, 1973, p. 427.

[13] Tanaka, Y., 'Adaptive segmentation schemes for relational files,' Information Society of Japan, Tech. Rep. on Software Theory, No. 4-5, 1983.

APPENDIX

A.1. Proof of Theorem 3.1.

For a colored trie T, let L(T) and R(T) denote the left subtrie and the right subtrie of its root, and C(T) be a color of its root. Then the following equalities hold.

Lemma A.1

$$\begin{aligned} Cavg(T, c) = {} & \text{if } T \text{ has only one node then } 1 \\ & \text{else} \\ & (1-(1/2)\Delta(c, C(T))) \\ & \quad *(Cavg(L(T), c)+Cavg(R(T), c)), \end{aligned} \tag{A.1}$$

where

$$\Delta(c, c') = \begin{cases} 1 & \text{if } c=c', \\ 0 & \text{otherwise.} \end{cases}$$

proof

Assume that T has more than one nodes. If $c \neq C(T)$ then Cavg(T, c) is

$$Cavg(T, c) = (1/2)^m \Big(\sum_{v \in \{0,1\}^m} naccess(T, c, v) \Big)$$

$$= (1/2)^m \Big(\sum_{v \in \{0,1\}^m} (naccess(L(T), c, v) + naccess(R(T), c, v)) \Big)$$

$$= Cavg(L(T), c) + Cavg(R(T), c).$$

Otherwise, Cavg(T, c) is

$$Cavg(T, c) = (1/2)^m\Big(\sum_{v\in\{0,1\}^{m-1}} naccess(L(T), c, v) + \sum_{v\in\{0,1\}^{m-1}} naccess(R(T), c, v)\Big).$$

Since naccess(L(T), c, v) equals to naccess(L(T), c, v∘0) and also to naccess(L(T), c, v∘1), the first sum in (3) becomes

$$(1/2)\Big(\sum_{v\in\{0,1\}^{m-1}} naccess(L(T), c, v\circ 0) + \sum_{v\in\{0,1\}^{m-1}} naccess(L(T), c, v\circ 1)\Big)$$

$$= (1/2) \sum_{v\in\{0,1\}^{m}} naccess(L(T), c, v)$$

$$= (1/2)Cavg(L(T), c).$$

Similarly, the second sum in (3) becomes

$$\sum_{v\in\{0,1\}^{m-1}} naccess(R(T), c, v) = (1/2)Cavg(R(T), c).$$

Both cases can be summarized as shown in this lemma. Q.E.D

Theorem A.2

Let leaves(T) denote a set of node codes of leaves in T. Then Cavg(T, c) is

$$Cavg(T, c) = \sum_{\alpha\in leaves(T)} (1/2)^{\rho(c(\alpha))}. \qquad (A.2$$

proof

This is proved by a mathematical induction on the number of leaves in T. If th number of leaves in T is one, then α is an empty string, and hence ρ(c(α)) i zero for any c. Therefore, the right hand side of (A.2) becomes one for any c which is consistent with the definition of Cavg(T, c). Suppose that the theore holds for every trie that has less than N leaves. Let T be a trie with leaves. From (A.1), Cavg(T, c) is

Cavg(T, c)

$$= (1-(1/2)\Delta(c, C(T)))*(Cavg(L(T), c) + Cavg(R(T), c)).$$

Since each of L(T) and R(T) has less than N leaves, this equality can be changed to

$$\begin{aligned} & Cavg(T, c) \\ &= (1-(1/2)\Delta(c, C(T))) \\ &\quad *\Big(\sum_{\alpha\in leaves(L(T))} (1/2)^{\rho(c(\alpha))} + \sum_{\alpha\in leaves(R(T))} (1/2)^{\rho(c(\alpha))}\Big). \end{aligned} \tag{A.3}$$

The set of leaves of T is clustered into two subsets;

$$\begin{aligned} & leaves(T) \\ &= \{\bar{C}(\bar{T})\circ\alpha \mid \alpha\in leaves(L(T))\} \\ &\quad \cup \{C(T)\circ\alpha \mid \alpha\in leaves(R(T))\}, \end{aligned}$$

where $c\circ\alpha$ denotes a concatenation of c and α. Besides, the following relation holds;

$$\rho(c(\bar{C}(\bar{T})\circ\alpha)) = \rho(c(C(T)\circ\alpha)) = \rho(c(\alpha))+\Delta(c, C(T)).$$

Therefore, (A.3) becomes

$$\begin{aligned} & (1-(1/2)\Delta(c, C(T))) \\ &\quad *\Big(\sum_{\alpha\in leaves(L(T))} (1/2)^{\rho(c(\bar{C}(\bar{T})\circ\alpha))-\Delta(c, C(T))} \\ &\quad + \sum_{\alpha\in leaves(R(T))} (1/2)^{\rho(c(C(T)\circ\alpha))-\Delta(c, C(T))} \\ &= (1-(1/2)\Delta(c, C(T)))*\Big(\sum_{\alpha'\in leaves(T)} (1/2)^{\rho(c(\alpha'))}\Big)*2^{\Delta(c, C(T))} \\ &= \sum_{\alpha'\in leaves(T)} (1/2)^{\rho(c(\alpha'))}. \end{aligned}$$

Therefore, the theorem also holds for T. This completes the proof. Q.E.D.

Lemma A.3

$$Cavg(new(T, \alpha, c), c') = Cavg(T, c') + (1-\Delta(c, c'))(1/2)^{\rho(c'(\alpha))}. \tag{A.4}$$

proof

From Theorem A.2, Cavg(new(T, α, c), c') is

$$Cavg(new(T, \alpha, c), c') = \sum_{\alpha\in leaves(new(T, \alpha, c))} (1/2)^{\rho(c'(\alpha))}. \tag{A.5}$$

Since it holds that

$$\text{leaves}(\text{new}(T, \alpha, c)) = (\text{leaves}(T)-\{\alpha\}) \cup \{\alpha\circ\bar{c}, \alpha\circ c\},$$

the equality (A.5) becomes

$$\sum_{\alpha'\in\text{leaves}(T)} (1/2)^{\rho(c'(\alpha'))}-(1/2)^{\rho(c'(\alpha))}+2(1/2)^{\rho(c'(\alpha))}+\Delta(c, c')$$

$$= \text{Cavg}(T, c')+(1-\Delta(c, c'))(1/2)^{\rho(c'(\alpha))}. \qquad \text{Q.E.D.}$$

Now the computation of $\text{Ccost}^1_{T,\alpha}(c)$ is an easy task, since it holds from Lemma A.3 that

$$\text{Ccost}^1_{T,\alpha}(c)$$

$$= \underset{c'\in C}{\text{average}}(\text{Cavg}(\text{new}(T, \alpha, c), c'))$$

$$= (1/n)\sum_{c'\in C} (\text{Cavg}(T, c')+(1-\Delta(c, c'))(1/2)^{\rho(c'(\alpha))})$$

$$= \text{cost}^1(T)+(1/n)\sum (1/2)^{\rho(c'(\alpha))}-(1/n)(1/2)^{\rho(c(\alpha))}.$$

A.2. Proof of Theorem 3.2.

Lemma A.4

Each color satisfies the following relation;

$$\text{Cworst}(\text{new}(T, \alpha, c), c) \le \text{Cworst}(\text{new}(T, \alpha, c^+), c),$$

where c^+ denotes a representative of the colors that are different from c.

proof

For each $v\in\{0,1\}^m$, it holds that

$$\text{naccess}(\text{new}(T, \alpha, c), c, v) = \text{naccess}(T, c, v).$$

Since the cost of the new trie is more than the old trie, it holds that

$$\text{Cworst}(\text{new}(T, \alpha, c^+), c) \ge \text{Cworst}(T, c)$$

$$= \max_{v\in\{0,1\}^m} \text{naccess}(T, c, v)$$

$$= \max_{v\in\{0,1\}^m} \text{naccess}(\text{new}(T, \alpha, c), c, v)$$

$$= \text{Cworst}(\text{new}(T, \alpha, c), c),$$

which proves the lemma. Q.E.D.

Assume that c maximizes Cworst(new(T, α, c^+), c). From the definition, $Ccost^2_{T,\alpha}(c)$ is

$$Ccost^2_{T,\alpha}(c) = \max_{c'} Cworst(new(T, \alpha, c), c')$$

$$= \max \Big(\max_{c' \neq c} Cworst(new(T, \alpha, c), c'),$$

$$Cworst(new(T, \alpha, c), c) \Big).$$

From Lemma A.4, this becomes

$$Ccost^2_{T,\alpha}(c) \leq \max \Big(\max_{c' \neq c} Cworst(new(T, \alpha, c'^+), c'),$$

$$Cworst(new(T, \alpha, c^+), c) \Big)$$

$$= \max_{c'} Cworst(new(T, \alpha, c'^+), c').$$

$$\leq Cworst(new(T,\alpha, c^+),c). \qquad (A.6)$$

Let c' be different from c. Then the followings hold.

$$Ccost^2_{T,\alpha}(c') = \max_{c''} Cworst(new(T, \alpha, c'), c'')$$

$$= \max \Big(\max_{c'' \neq c'} Cworst(new(T, \alpha, c'), c''),$$

$$Cworst(new(T, \alpha, c'), c') \Big)$$

$$\geq Cworst(new(T, \alpha, c^+), c). \qquad (A.7)$$

From (A.6) and (A.7), it holds that

$$Ccost^2_{T,\alpha}(c') \geq Ccost^2_{T,\alpha}(c).$$

Therefore, c minimizes $Ccost^3_{T,\alpha}(c)$. Q.E.D.

A.3. Proof of Theorem 3.3.

From the definition of Cworst(T, c), it becomes

$$Cworst(T, c) = \max \Big(\max_{v \in \{0,1\}^m} naccess(T, c, 0 \circ v),$$

$$\max_{v\in\{0,1\}^m} \text{naccess}(T, c, 1\circ v) \;) \qquad \text{(A.8)}$$

Consider a search specified as 'c = 0∘v'. The search fails in T|c, while all the leaves in T/c satisfy this search condition. Therefore, the following equality holds.

$$\max_{v\in\{0,1\}^{m-1}} \text{naccess}(T, c, 0\circ v)$$

$$= \text{card}(\text{leaves}(T/c)) + \max_{v\in\{0,1\}^{m-1}} \text{naccess}(T|\bar{c}, c, 0\circ v).$$

Since a leaf with a node code α does not belong to $T|\bar{c}$ if c(α) begins with '1', the subtree $T|\bar{c}$ satisfies

$$\max_{v\in\{0,1\}^{m-1}} \text{naccess}(T|\bar{c}, c, 1\circ v) = 0.$$

Therefore, the followings hold.

$$\max_{v\in\{0,1\}^{m-1}} \text{naccess}(T, c, 0\circ v)$$

$$= \text{card}(\text{leaves}(T/c)) + \max_{v\in\{0,1\}^{m-1}} \text{naccess}(T|\bar{c}, c, 0\circ v)$$

$$= \text{card}(\text{leaves}(T/c))$$

$$+ \max \Big(\max_{v\in\{0,1\}^{m-1}} \text{naccess}(T|\bar{c}, c, 0\circ v),$$

$$\max_{v\in\{0,1\}^{m-1}} \text{naccess}(T|c, c, 1\circ v) \Big)$$

$$= \text{card}(\text{leaves}(T/c)) + \max_{v\in\{0,1\}^{m}} \text{naccess}(T|\bar{c}, c, v)$$

$$= \text{card}(\text{leaves}(T/c)) + \text{Cworst}(T|\bar{c}, c) \qquad \text{(A.9)}$$

The substitution of (A.9) and the similar result for max(naccess(T, c, 1∘v)) in (A.8) proves the theorem.