ICOT Technical Report: TR-347

TR-347

Structural Superimposed Codeword as an Indexing Scheme for Terms

by
A. Nakase, S. Shibayama, H. Sakai,
Y. Morita, H. Monoi & H. Itoh

March, 1988

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191－5
Telex ICOT J32964

Institute for New Generation Computer Technology

Structural Superimposed Codeword as an Indexing Scheme for Terms

Akihiko Nakase, Shigeki Shibayama, Hiroshi Sakai (Toshiba corporation)
Yukihiro Morita, Hidetoshi Monoi, Hidenori Itoh
(Institute for New Generation Computer Technology)

Abstract

This paper describes the structural superimposed codeword (SSCW) as an indexing scheme for terms. It reports the results of performance evaluation of SSCW by experimental retrieval of terms by SSCW. These experiments showed several characteristics of SSCW for use in indexing for terms.

1. Introduction

For the advance of knowledge information processing, it is necessary to realize databases which describe models of the real world.

Basic elements of most conventional database systems, are non structured data, such as atomic values. Sometimes, atomic values alone are insufficient to describe models of the real world.

On the other hand, in the study of logic programing such as Prolog, logic is a very powerful method to describe complex objects.

Recently, some database models which manipulate terms to import logic into database have been reported. [Tsur86], [Yokota86], and [Morita86a].

In database models which manipulate terms, it is useful to make indices to terms for effective retrieval of terms.

However, it is very difficult to make an index to a term by using indexing techniques of conventional database systems, because a term has a complex structure and may include variables.

[Ramamohanarao86], and [Wise84] report how to apply the superimposed codewords (SCW) [Roberts79] to the index of a term. In naive implementation of SCW for text retrieval, we can make indices to terms if the terms in a database are variable free. This paper explains how

to make an index to a term using the naive implementation of SCW instead of explaining the indexing scheme of SCW.

We assume that the terms illustrated in Figure 1.1 are stored in the database.

First, the hashed value of each constant symbol in a term is calculated. For example, for term (1) in Figure 1.1, we calculate h(a), h(b), h(c), and h(d), where "h" denotes the hashing function and "h(a)" is the binary hashed value of "a" using "h". The bit length of each hashed value is set at the same length. To synthesize the information of h(a), h(b), h(c), and h(d), we use the technique of superimposed code words. We assume that h(a), h(b), h(c), and h(d) are binary values and have the same bit length, and we OR h(a), h(b), h(c), and h(d) and make its result an index to (1), as illustrated in Figure 1.2. Indices for (2) to (4) are also generated by this method.

To retrieve the database by a term, we also make an index to term to retrieve, in the same way that we made indices to terms to be retrieved. (In this paper, to distinguish the terms to be retrieved and terms to retrieve, the former is called data terms and the latter query terms. The index to a data term is called a data index, and the index to a query term is called a query index.)

To use SCW for the index to terms, data terms should be variable free, but query terms may include variables.

For example, if the query term is "a(b,X)", the index to the query term is made by superimposing h(a) and h(b). The variables in the query term are not superimposed for the index. (In other words, the hashed value of variables in a query term is a sequence of bit 0')

Retrieval of unifiable terms to query terms is performed as follows.

We assume that the index of the i-th data term is Di and the index of the query term is Q; if Di^Q=Q holds, the term with Di is a unifiable candidate to the term with Q. (^ denotes AND in this paper)

This is because, if all constants in the query term appear in the data term, the above equation always holds.

However this scheme cannot deal with data terms which have variables.

To realize a database which can manipulate all kinds of terms, like the models described in [Yokota86] and [Morita86a], indices to terms which can include variables in data terms and query terms are useful.

This paper proposes a new indexing scheme for terms named structural superimposed codeword(SSCW) which is developed from SCW and can deal with variables for data terms.

Section 2, gives an overview and explains the SSCW construction method of with some examples. Section 3, explains how to evaluate SSCW using test data, test methods and test items. Section 4 describes and analyses the results of section 3, Section 5 compares the SSCW to other indexing methods to terms.

Section 6 gives the conclusion and describes future research.

2. SSCW Scheme

2.1 SSCW Construction Method

In an SCW, the hashed values of each symbol are merely superimposed because the main interest of the SCW is character information of the constant symbols in a term. However, to make an index to a term, we must consider two more items of information: information on the structure of a term and information of variables in a term.

By superimposing the hashed values of each constant symbol in a term according to the structure of the term, the SSCW takes the structure information of the term.

Information of variables in a term is included in the index by providing special hashed values for variables.

A term is defined recursively using the following three rules.

(1) An atom is a term.

(2) A variable is a term.

(3) A functor which has several arguments, where each argument is a term, is a term.

An SSCW is a bit string and is defined recursively as follows.

(1) The SSCW of an atom is its hashed value.

(2) The SSCW of a variable is the bit sequence of 1 for data term and the bit sequence of 0 for a query term.

(3) The SSCW of a compound term is obtained as follows.

First, the hashed value of the functor of a compound term is calculated.

Next, the SSCWs of each argument of this functor are calculated. Here, the bit length of the SSCW of an argument of this functor should be shorter than the functor itself.

Last, the SSCWs of each argument of this functor are superimposed onto the hashed value of the functor. Here, the hash bit range of the functor should cover the hash bit ranges of its arguments.

In (1) and (3), the same hashing function is used to calculate hashed values of each constant symbol in the data terms and query terms. This hashing function is from a constant symbol and its information on structure to the hash bit string.

Figure 2.1.1 shows a sample algorithm to make an SSCW.

Procedure "make_scw" is a procedure for three arguments.

The input is 'Term' and 'Hashlen', and the output is 'Index'.

'Term' is a term for which we want to make an SSCW, 'Hashlen' is the bit length of SSCW of 'Term', and 'Index' is the SSCW of 'Term'.

In this algorithm, the SSCW of each argument of a functor is concatenated horizontally. This paper evaluates the performance of an SSCW which is based on this algorithm. (Concatenating each argument of a functor, is one special version of the SSCW. Of course, there are other ways to make an SSCW, for example to superimpose each argument of a functor.)

For example, the SSCW of term "a(b,c(d))" is constructed as illustrated in Figure 2.1.2.

2.2 SSCW Design Parameters

The hash bit field of some functor in a term (some functor means not only the outer most functor of a term, but also the functors of subterms of a term) can be divided into two fields. One is the field which is superimposed by the hashed value of its arguments, and the other is the field which is not superimposed by the hashed value of its arguments. The former field is called the "superimposed field" (SF) of the hash bit field of the functor, and latter field the "non superimposed field" (NSF) of the hash bit field of the functor. The following two parameters are used to make an SSCW.

(1) Relationship between SF and NSF

To decide the ratio between the length of the SF and NSF, we introduce a parameter superimposing ratio (SR). Let the hash bit length of a functor be LF, the number of arguments of this functor be N, the hash bit length of an argument be A, the superimposing ratio be SR, the bit length of the superimposed field be LSF, and the bit length of the non-superimposed field be LNSF. Then the following equation holds, in the previous algorithm.

$$A = \langle LF*SR/N \rangle \qquad (2.2.1)$$

$$LSF = A*N \qquad (2.2.2)$$

$$LNSF = L-A*N \qquad (2.2.3)$$

$\langle X \rangle$ denotes rounding the fraction number of X.

(2) Number of "bit 1"s to be set in the hash bit field

An important factor for indices using the superimposition technique is to decide how many "bit 1"s should be set in a hash bit field [Roberts79] and [Morita87].

In the hash bit field of bit length L, if N of bit 1 are set, the bit setting ratio (BSR) is N/L.

If hash bit length L and BSR are given first, the number of "bit 1"s to be set in the hash bit field can be determined by the following

equation.

$N = \langle L*BSR \rangle$ (2.2.4)

There are two hashing methods to decide the BSR of a functor.

1. Uniformly Distributed Hash

This method uses one BSR for the hash bit field of one functor.

2. Field Separated Hash

This hashing method divides the hash bit field of a functor into the SF and NSF. Bit 1 of SF and NSF are set using different BSR.

In field separated hash, the interference of bit 1 of the SSCW of the argument to the functor is less than that of uniformly distributed hash. Hence the SF, BSR are expected to be significantly lower than that of the NSF.

Figures 2.2.1 and 2.2.2 illustrate these two hashing methods.

2.3 Retrieval Using SSCW

SSCW retrieval is the same as that of the SCW, that is

$D \hat{} Q=Q$ (2.3) where D is the index of the data term, DT, and Q is the index of the query term, QT.

If DT and QT are unifiable, (2.3) holds [Morita86b].

If DT and QT are unifiable, there is some substitution "s" which makes DTs=QTs, and DTs is the ground instance of DT.

We assume that the SSCWs of DTs and QTs are S.

For S and D, $D \hat{} S=S$ always holds because the hashed values of variables in the DT are the bit sequence of 1.

For S and Q, $S \hat{} Q=Q$ always holds because the hashed values of variables in the QT are the bit sequence of 0.

By $D \hat{} S=S$ and $S \hat{} Q=Q$, $D \hat{} Q=Q$ always holds if the DT and QT are unifiable.

2.4 Examples

2.4.1 Example Obtaining an SSCW

The following example shows how an SSCW is obtained.

For example, we make an SSCW of "a(b(X),c)" using the two hashing methods.

(1) Uniformly distributed hash

Assume that the SR is 70%, the hash bit length of SSCW is 16 bits, and the BSR is 30%. Term "a(b(X),c)" is expanded into a tree. Then, the hash values of "a", "b", "c", and "X" are calculated. First, the hash bit length of each hashed value is determined. The hash bit length of "a" is determined as 16 bits.

For the hash bit length of "b" and "c", by equation (2.2.1), we obtain

A = <16 bits * 0.7/2> = 6 bits.

Hence, the hash bit length of "b" and "c" is determined as 6 bits.

The hash bit length of "X" is

A = <6 bits * 0.7/1> = 4 bits.

Next, we decide how many bits should be set in the hash field.

The number of "bit 1"s is calculated by applying equation (2.2.4).

For "a", <16 bits * 0.3> = 5 bits, 5 bits should be set to 1.

For "b" and "c", <6 bits * 0.3> = 2 bits, 2 bits should be set to 1.

For "X", all bits should be set to 1 for a data term or all bits should be set to 0 for a query term.

We assume that we have the following hashed values,

h(a) = 0010010000010101

h(b) = 100010

h(c) = 010001

h(X) = 1111 (term in data).

By superimposing them as illustrated in Fig 2.4.1, we obtain the SSCW '0010111111010101'.

(2) Field separated hash

Let SR be 70%, BSR of SF be 45%, and BSR for NSF be 20%.

In field separated hash, the hash bit lengths of "a", "b", "c", and "X" are same as those of uniformly distributed hash. However, the length of SF and NSF of each constant symbol must be calculated.

For "a", SF = 6 bits * 2 = 12 bits, NSF = 16 bits - 6 bits * 2 = 4 bits.

For "b", SF = 4 bits * 2 = 4 bits, NSF = 6 bits - 4 bits = 2 bits.

For "c", there is no SF NSF = 6 bits.

Next, we must calculate the bit numbers to be set to 1.

For "a", at SF <12 bits * 0.2> = 2 bits,

at NSF <4 bits * 0.45> = 2 bits.

For "b", at SF <4 bits * 0.2> = 1 bit,

at NSF <2 bits * 0.45> = 1 bit.

For "c", at NSF <6 bits * 0.45> = 3 bits.

We assume that we have obtained the following hashed values.

h(a) = 1010 (NSF) '+' 000100000010 (SF)

h(b) = 10 (NSF) '+' 0100 (SF)

h(c) = 011001 (NSF)

h(X) = 1111 (variable in data term)

In this calculation, '+' means concatenation of bit fields.

Lastly, we superimpose them as illustrated in Figure 2.4.2, and obtain the SSCW '1010101111011011'.

2.4.2 Example of Retrieval Using SSCW

We assume that the set of terms to be retrieved is as follows.

a(b(X),c) (1)

a(b(e),X) (2)

a(X,b(d)) (3)

a(c(d),c) (4)

a(f(e),X) (5)

We assume that the query term is as follows.

a(b(d),X) (6)

"h(x,n-bits)" means the hash value of "x" whose hash bit length is n bits. We assume that the hashed value of each constant symbol is as follows.

h(a,16-bits) = '1001000100110000'

h(b,6-bits) = '001010'

h(c,6-bits) = '100010'

h(f,6-bits) = '000011'

h(d,4-bits) = '0001'

h(e,4-bits) = '1000'

Figure 2.4.2 shows the process of making an SSCW index using the above hashed values.

By using retrieval criterion "Di^Q=Q", (1), (3), and (5) are selected as unifiable candidates to (6).

In fact, however, (5) is not unifiable to (6). The SSCW uses the hash function and superimposes the hashed values. Hash collision and interference of hashed values may occur here. Such mis-selections are called false drops.

In the following section, the performance of SSCW is measured by counting the false drops.

3. Evaluation

3.1 Purpose of evaluation

As stated in the previous section, certain parameters must be determined to make an SSCW.

Therefore, we must evaluate the effect of parameters for the selectivity of SSCW. Next, the selectivity of SSCW depends on the term set to be indexed. Therefore we must also evaluate the effect of the characteristics of the term set to on the selectivity of the SSCW.

(1) Selection of the best parameters of the SSCW

As stated above, an SSCW has two parameters: SR and BSR.

By changing the SR and BSR, we can observe how the selectivity of the SSCW changes, and predict the best SSCW parameters, in uniformly distributed hash and field separated hash.

(2) Observation of the SSCW performance when applied to term sets with various characteristics

The selectivity of the SSCW seems to change if the characteristics of

the term set change. By changing the number of kinds of constant symbols in a term set, figure of terms, and number of variables in a term set, we observe how the selection ability of the SSCW changes.

From (1) and (2), we predict which hash method is better for dealing with term sets with various characteristics.

3.2 Evaluation Methodology

To evaluate the SSCW performance, we prepared several term sets, each with 100 terms. These 100 terms can be 100 data terms, or 100 query terms. We retrieved 100 data terms by 100 query terms. This is equivalent to 10000 selection with unification or 100*100 join with unification. The following R and S were counted.

-R number of pairs that can be unifiable

-S number of pairs that seem to be unifiable by SSCW

We defined the false drop ratio (FDR) as follows.

FDR = (S-R)/S

FDR means the extent to which false drop is included in pairs selected by an SSCW.

For the actual evaluation of this paper, most experiments are performed by retrieving a term set using itself for query terms. This is called u-self-retrieval.

3.3 Sample Term Sets

We provided 23 term sets for the test.

We selected three parameters for the term sets.

Parameter 1: Number of different kinds of constant symbols in a term set

Parameter 2: Average number of nodes in one term in a term set (term size)

Parameter 3: Ratio of variable nodes to the total number of nodes in a term set

The 23 term sets are categorized into four blocks.

(1) Term block 1 (term 00 and term 01)

Term sets in term block 1 are normal. The three parameters are set to medium value.

(2) Term block 2 (term 10 to term 20)

In term sets in term block 2, to vary the ratio of variables in the term set, parameter 1 is set to 30, parameter 2 is set to about 4 nodes/term, and parameter 3 varies from 0% to 60%.

(3) Term block 3 (term 21 to term 25)

In term sets in term block 3, to vary the size of the terms in the term set, parameter 1 is set to 30, parameter 2 varies from 3 nodes/term to 10 nodes/term, and parameter 3 is set to about 15%.

(4) Term block 4 (term 26 to term 30) In term set in term block 4, to vary the number of kinds of constant symbols in it, parameter 1 varies from 40 to 300, parameter 2 is set to about 4 nodes/term, and parameter 3 is set to about 3%. Details of the test data are illustrated in Figure 3.

3.4 Test Item

We performed the following five experiments. In each experiment, we measured the FDR of uniformly distributed hash and field separated hash.

Experiment 1 (test to observe the effect of the SR)

To observe the effect of the SR on the FDR, we fixed the BSR, varied the SR from 50% to 100% by 10%, and measured the FDR. For uniformly distributed hash, the BSR is fixed to 30%. For field separated hash, the BSR for the SF is set to 10%, and BSR for the NSF is set to 45%. We used term 00 for the test data.

Experiment 2 (test to observe the effect of the BSR)

To observe the effect of the BSR on the FDR, we fixed the SR to 70%, varied the BSR, and measured the FDR. For uniformly distributed hash, we varied the BSR from 10% to 60%. For field separated hash, we varied the BSR of the SF NSF, from 0% to 20% by 10% and 40% to 55% by 5%,

respectively. We used term 01 as test data.

Experiment 3 (test to observe the effect of variables in terms)

To observe the effect of the ratio of variables in terms on the FDR, we fixed the SR to 70%, the BSR of uniformly distributed hash to 35%, and the BSR of field separated hash to 10% for the SF and 50% for the NSF. We used term 10 to term 19 for the test data.

Experiment 4 (test to observe the effect of the size of terms)

To observe the effect of the size of terms in the term set on the FDR, we fixed the SR to 70%, the BSR of uniformly distributed hash to 35%, and the BSR of field separated hash to 10% and 50% for the SF and NSF. Generally, by increasing the size of a term, the hashed bit length of each symbol in a term is shortened. In this experiment, we wanted to observe the effect of reducing the hash bit length in a large term. We used term 20 to term 23 for test data.

Experiment 5 (test to observe the effect of number of kinds of constant symbols in terms)

To observe the effect of the number of kinds of constant symbols in term set to FDR, we fixed the SR to 70%, the BSR of uniformly distributed hash to 35%, and the BSR of field separated hash to 10% and 50% for the SF and NSF. The more kinds of constant symbols in terms, the more hash collisions occur. In this experiment, we wanted to observe the effect of hash collision on the FDR. We used term 30 to term 33 for test data.

We set the length of SSCW index to 32 bits. However, in experiment 5, we wanted to observe the hash collision. Therefore, we used a 24 bit SSCW index for experiment 5.

4. Results

(1) Effect of the SR on the FDR

Figure 4.1.1 shows the result of the experiment for uniformly distributed hash and Figure 4.1.2 shows the result for field separated hash. In both hashing methods, FDR is very large at SR = 100%, because of the effect of variables in data terms. Suppose an SSCW of "a(X,Y)" whose SR is 100%. The SSCW is the sequence of all "bit 1"s. (See Figure 4.1.3.) This index matches everything. For the join operation in particular, its effect on the FDR is great. An SR of less than 60% shows bad FDR, because the small SR makes the hash bit length of arguments of a functor too small, and it cannot contain the argument information effectively. (See Figure 4.1.4)

In this experiment, the best SR is about 60% to 80%. However we must be careful with this result, because it was obtained from randomly generated term sets. In the test data we used, the average number of arguments of a functor is 2 to 3. Hence, an SR of 70% can share the hash bit field of the functor and arguments well. If the number of arguments increases in a term set, the SR should be 70% or more. However, because of the effect of variables in data terms, too large an SR is assumed to show bad selectivity. We assume that SR should not exceed 80% or 90%.

(2) Effect of the BSR on the FDR

Figure 4.2.1 shows the result of experiment 2 for uniformly distributed hash and Figure 4.2.2 shows the result for field separated hash. In experiment 2, in uniformly distributed hash, a BSR of 30% to 40% shows excellent performance. Theoretically, a BSR of 50% provides the best resolution capability for the hashed value of one constant symbol [Roberts79]. However, in our method, we superimpose each hashed value, so that the BSR of the resultant SSCW is too large if we set a BSR of 50% to each constant symbol. Too small a BSR, however, provides poor resolution capability for a constant symbol. [Morita87] reports that the best BSR for one constant symbol is where the BSR of a constructed SSCW without variable symbols is 50%.

In the test data of this experiment, the average depth of terms is 2

to 3, and the average number of constant symbols to be superimposed is 1.5 to 2.5, therefore, by using a BSR of 30% to 40%, we can make the BSR of the resultant SSCW about 50%. The maximum number of constant symbols to be superimposed is determined by the index length and average number of arity. For example, in a 32 bit SSCW for a 2-arity term, the maximum depth of the hashed value to be superimposed is 4. By using the index length of terms and average number of arity, we can predict the maximum number of constant symbols to be superimposed and can predict the best BSR.

In field separated hash, it seems that a BSR of 50% in NSF and a BSR of 10% in SF is good. However, compared with uniformly distributed hash, the difference between the tested BSR of the NSF and SF in field separated hash is not very great.

(3) Effect of variables in term sets on the FDR

Figure 4.3.1 shows the result of experiment 3 for uniformly distributed hash and field separated hash. In experiment 3, uniformly distributed hash is weak in terms with many variables because of the effect of variables in data terms.

Suppose that for an SSCW of "a(X,Y)" whose SR is 70%, 70% of the hashed value of "a" is masked by the hashed value of "X" and "Y". (See Figure 4.3.2.) In uniformly distributed hash, the information about "a" is distributed uniformly in the SF and NSF. However, in field separated hash, most of the information on "a" is concentrated in the NSF. Thus even for terms with many variables, information on a functor in the SSCW is not masked significantly.

(4) Effect of the size of terms on the FDR

Figure 4.4.1 shows the result of experiment 4 for uniformly distributed hash and field separated hash. In experiment 4, field separated hash shows better selectivity than uniformly distributed hash. This is because of the effect of the variables explained in

4.(3). Next, we observe the result of uniformly distributed hash and field separated hash independently. For uniformly distributed hash, the FDR increases by the increment of the size of the term, because reducing the hash field also reduces the resolution capability of a leaf node of a term. For field separated hash, FDR does not vary by the effect of the size of terms.

For the resolution capability of a leaf node of a term, field separated hash is superior to uniformly distributed hash, so the uniformly distributed hash is more easily affected by the size of terms than field separated hash.

(5) Effect of the number of kinds of constant symbols in a term set

Figure 4.5.1 shows the result of experiment 5 for uniformly distributed hash and field separated hash. In experiment 5, for terms with only a few kinds of constant symbols, hash collision did not occur in either method. Therefore, by the effect of variables, field separated hash shows better selectivity than uniformly distributed hash. In terms with many kinds of constant symbols, field separated hash shows a worse FDR. This is quite natural because the resolution capability of field separated hash is generally less than that of uniformly distributed hash.

As a result, we can conclude that for hash collision, uniformly distributed hash is stronger than field separated hash. To deal with term sets with many variables, field separated hash is superior to uniformly distributed hash. The effect of variables on the FDR seems to be greater than that of hash collision.

5. Comparison with Other Indexing Scheme

[Wise84] proposes field encoded words (FEW) for the index to terms. In this method, the hashed value of each symbol in a term is concatenated as illustrated in Figure 5.1. In FEW, the BSR of each hashed value is uniformly 50%.

BSR of the NSF = 50% (FEW).

The test cases we used are as follows.

(a) Retrieval of a term set with only a few variables by a term set with many variables

	Data term set	Query term set
Case 1	Term 11	Term 19
Case 2	Term 11	Term 18
Case 3	Term 12	Term 19
Case 4	Term 12	Term 18

Figure 5.4 shows the results of this experiment.

(2) Retrieval of a term set with many variables by a term set with only a few variables

	Data term set	Query term set
Case 5	Term 19	Term 11
Case 6	Term 19	Term 12
Case 7	Term 18	Term 11
Case 8	Term 18	Term 12

Figure 5.5 shows the results of this experiment.

As is obvious from Figures 5.4 and 5.5, FEW shows a larger FDR than the SSCW with field separated hash for retrieving term sets with only a few variables by term sets with many variables, while there is no significant difference in retrieving term sets with many variables by term sets with only a few variables.

[Ramamohanarao86] and [Ohmori87] report indexing methods for terms. In both methods, terms to be indexed should be expanded first, according to a provided template term, and the hashed values of a term are synthesized by superimposing [Ramamohanarao86] or concatenating [Ohmori87] them.

The advantages of the above methods are efficient resolution

In FEW, the hashed value of variables in a data term can be expressed as a sequence of 1, and those of a query term as a sequence of 0. The retrieval method of FEW is also D^Q=Q, where D is the index for data terms and Q is the index for query terms. FEW is considered to be one special version of field separated hash of the SSCW.

Figure 5.2 shows the field separated hash of an SSCW where the BSR of the SF is 0% and the BSR of the NSF is 50%. As is obvious from this figure, we can obtain FEW in the same way as constructing an SSCW.

The advantage of FEW (in other words, the SSCW of field separated hash with a BSR of 50% to the NSF and a BSR of 0% to the SF) is that the hashed value of a functor is not interfered with by those of the arguments, and vice versa.

We made additional experiments for the term set, where the number of kinds of functors (they cannot be leaf nodes of terms) is fixed to five and the number of kinds of leaf node is 30, 50, or 100.

We made u-self-retrieval for these term sets by using the following two indexing methods.

(1) SSCW field separated hash where the BSR of the SF = 10% and the BSR of the NSF = 50%

(2) SSCW field separated hash where the BSR of the SF = 0% and the BSR of the NSF = 50% (FEW)

Figure 5.3 shows the result of this experiment. Because of the non-interference of the hashed value of the functor to the hashed value of arguments, (2) shows better performance than (1), if the number of different kinds of leaf nodes of a term increases.

The disadvantage of FEW is that when applied to query terms with many variables, there are many unused bits.

We retrieved a term set with only a few variables by a term set with many variables, and vice versa.

The indexing methods we used in this experiment are as follows.

(1) SSCW field separated hash where the BSR of the SF = 10% and the BSR of the NSF = 50%

(2) SSCW field separated hash where the BSR of the SF = 0% and the

capability of hashed values, and no interference of the hashed values of variables to the hashed values of constants.

The disadvantages are that their retrieval operation is rather complicated. They need logical operation with the same number of nodes as the template term. The FDR of such indexing methods is very much dependent on the figure of the template term. If the nodes of terms exceed the size of the template term, information on those nodes is ignored.

The advantage of the SSCW is simple retrieval operation. In the SSCW, a unifiable pair of terms can be checked by one logical operation which is provided in almost all micro-processors.

The SSCW can also flexibly determine the hash bit field of all nodes in the term, preventing nodes in a term set from being ignored.

6. Conclusion

This paper introduced details of the SSCW and showed its characteristics when applied to various kinds of term sets. This study showed that the effect of variables in term sets is stronger than that of the size of terms and hash collision in an SSCW. However, using field separated hash rather than uniformly distributed hash prevents performance reduction by the variables in term sets.

In future, we will use this SSCW method in an experimental knowledge base machine [Shibayama87], and measure the performance of the index to terms.

REFERENCES

[Ohmori87] Ohmori, T. et al., An Algebraic Deductive Database Managing a Mass of Rule Clauses, Proc. 5th International Workshop on Database Machine, 1987

[Morita86a] Morita, Y. et al., Retrieval-By-Unification Operation on a Relational Knowledge Base, Proc. 12th VLDB, 1986

[Morita86b] Morita, Y. et al., Structure Retrieval via the Method of Superimposed Codes.", Proc. 33th Annual Convention IPS Japan, 6L-8, 1986 (in Japanese)

[Morita87] Morita, Y. et al., A Knowledge Base Machine with an MFPM(3) An Indexing Scheme for Terms, Proc. 35th Annual Convention IPS Japan, 2C-7, 1987, (in Japanese)

[Roberts79] Roberts, C.S., Partial-Match Retrieval via Method of Superimposed Codes, Proc. of IEEE, 67(12), pp.1624-1642, 1979

[Ramamohanarao86] Ramamohanarao, K. and Shepherd, J., A Superimposed Codeword Indexing Scheme for Very Large Prolog Database, Proc. 3rd International Logic Programming Conference, 1986

[Shibayama87] Shibayama, S. et al., An experimental Knowledge Base Machine with Unification-based Retrieval Capability, Proc. 2nd France-Japan AI and CS Conf., 1987

[Tsur86] Tsur, S. and Zaniolo, C., LDL: A Logic-Based Data-Language, Proc. 12th VLDB, 1986

[Wise84] Wise, M.J. and Powers, D.M.W., Indexing PROLOG Clauses via Superimposed Code Words and Field Encoded Words, Proc. IEEE Conf. Logic Programming, Atlantic City, NJ, January 1984, pp.203-210

[Yokota86] Yokota, H. and Itoh, H., A Model and Architecture for a Relational Knowledge Base, Proc. 13th International Symposium on Computer Architecture, 1986

```
a(b, c(d))          (1)
a(b, c(f))          (2)
a(k(d), c(d))       (3)
a(k(f), c(f))       (4)
```

Figure 1.1 Example of a term set in a database which manipulates terms

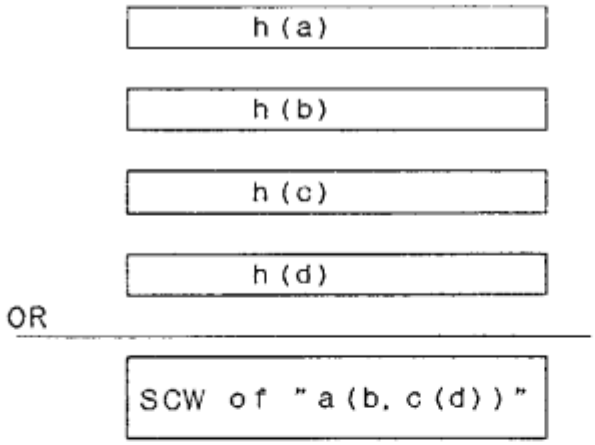

Figure 1.2 Index of "a(b,c(d))" using SCW

```
procedure make_scw(Term, Hashlen, Index)
{
    var Funct    /* hash value of a functor */
        Arglen   /* hash bit length of an argument */
        Arg_n    /* n-th argument of the functor    */
        Argindex_n   /* SSCW of Arg_n */
    if Term is a variable
       Index is the hashed value of Term whose length is Hashlen;
       return;
    if Term is an atom
       Index is the hashed value of Term whose length is Hashlen;
       return;
    else
       Funct is the hashed value of the functor of Term, whose length is Hashlen;
       Arglen is the hash bit length of Index of each argument of the functor;
       for all arguments of the functor
         make_scw(Arg_n,Arglen,Argindex_n);
         superimpose the concatenation of all Arg-n onto Funct, and make it Index;
         return;
}
```

Notes:

1. The hash bit length of a functor should be longer than the sum of the hash bit lengths of the SSCW of its arguments.

2. The hash bit field of each argument of a functor should be the same length and be concatenated using the order of arguments of the functor.

Figure 2.1.1 Sample algorithm to make an SSCW

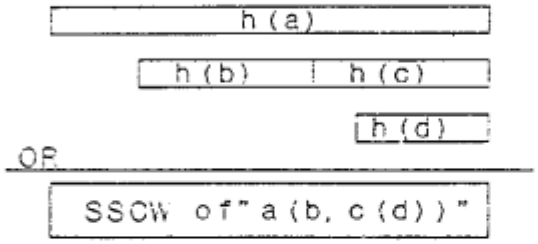

Figure 2.1.2 Construction process of the SSCW of term "a(b,c(d))"

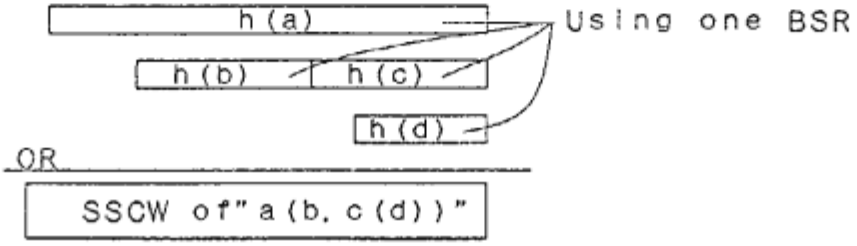

Figure 2.2.1 BSR for uniformly distributed hash

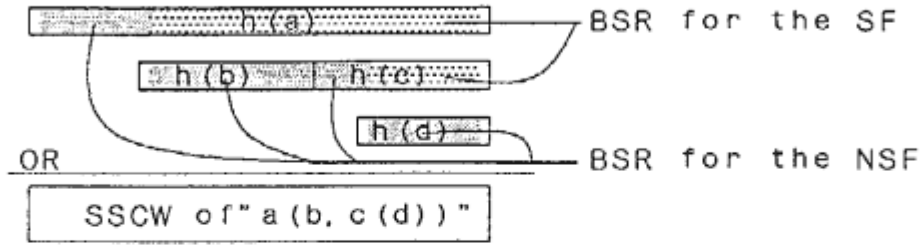

Figure 2.2.2 BSR for field separated hash

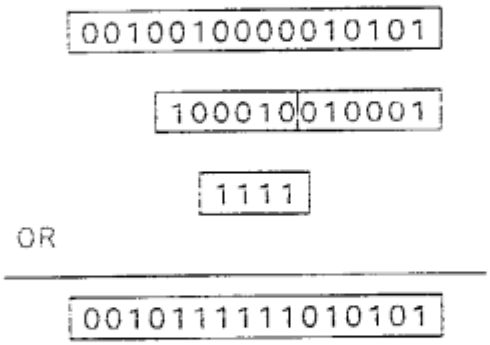

Figure 2.4.1 Sample SSCW of "a(b(X),c)" by uniformly distributed hash

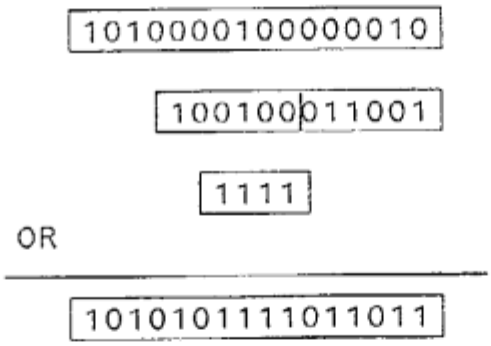

Figure 2.4.2 Sample SSCW of "a(b(X),c)" by field separated hash

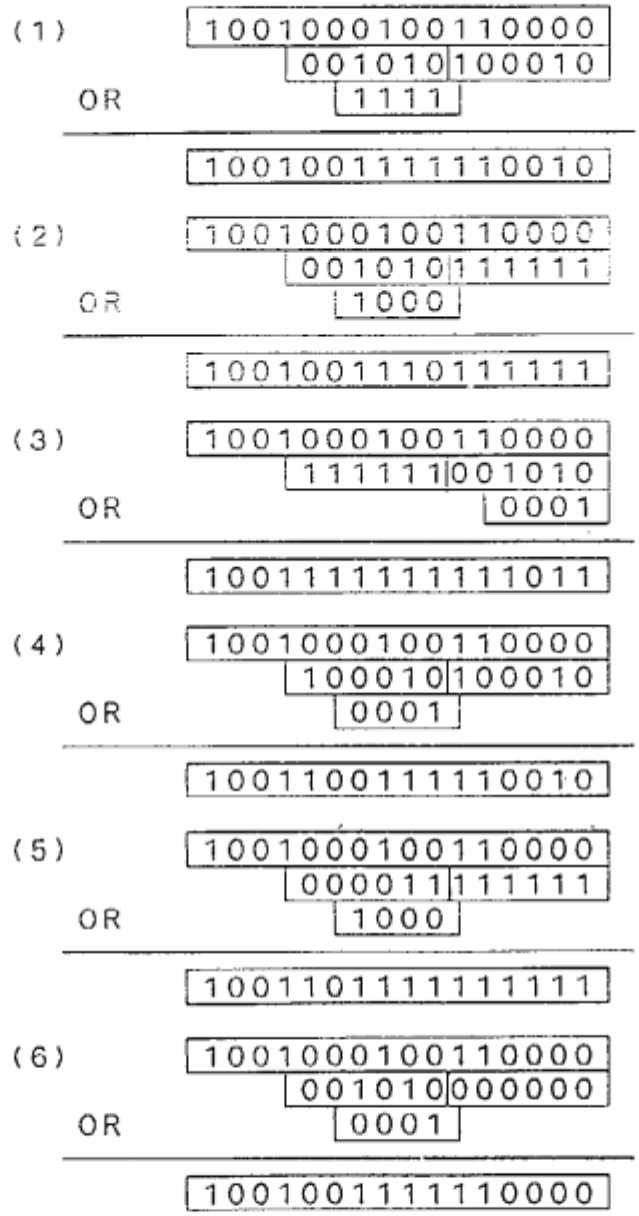

Figure 2.4.3 Process of making SSCWs

	Ratio of variables in term sets (%)	Size of a term (nodes/term)	Number of kinds constants
Term 00	14.8	2.8	30
Term 01	16.0	3.6	15
Term 10	0.0	4.1	30
Term 11	5.3	4.2	30
Term 12	11.0	4.0	30
Term 13	15.9	3.8	30
Term 14	22.4	3.9	30
Term 15	30.5	3.7	30
Term 16	34.5	3.9	30
Term 17	38.7	3.9	30
Term 18	44.7	3.6	30
Term 19	49.9	3.7	30
Term 20	56.1	3.7	30
Term 21	14.9	2.8	30
Term 22	14.9	3.8	30
Term 23	16.6	5.3	30
Term 24	14.6	6.7	30
Term 25	16.5	11.6	30
Term 26	2.9	4.2	40
Term 27	2.9	4.2	60
Term 28	2.9	4.2	100
Term 29	2.9	4.2	200
Term 30	2.9	4.2	300

Figure 3 Details of term sets

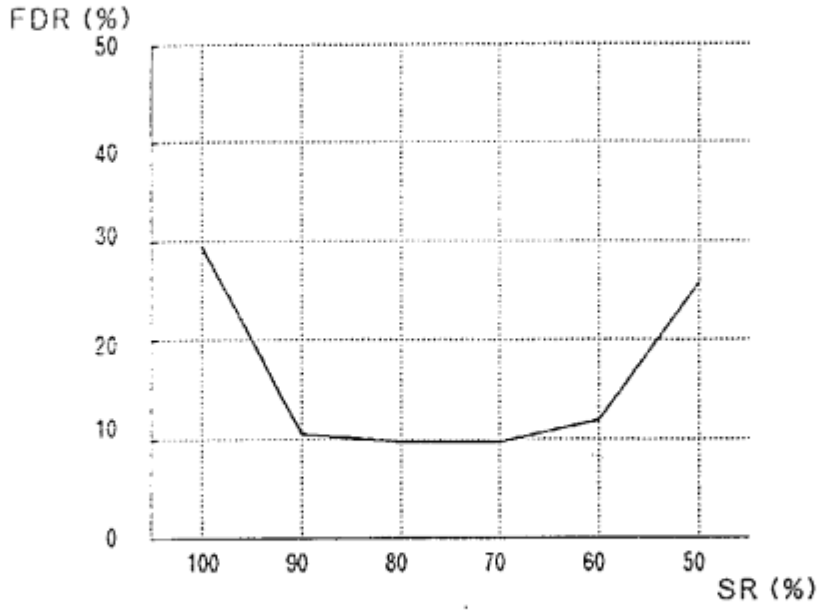

Figure 4.1.1 Relationship between SR and FDR
(in uniformly distributed hash)

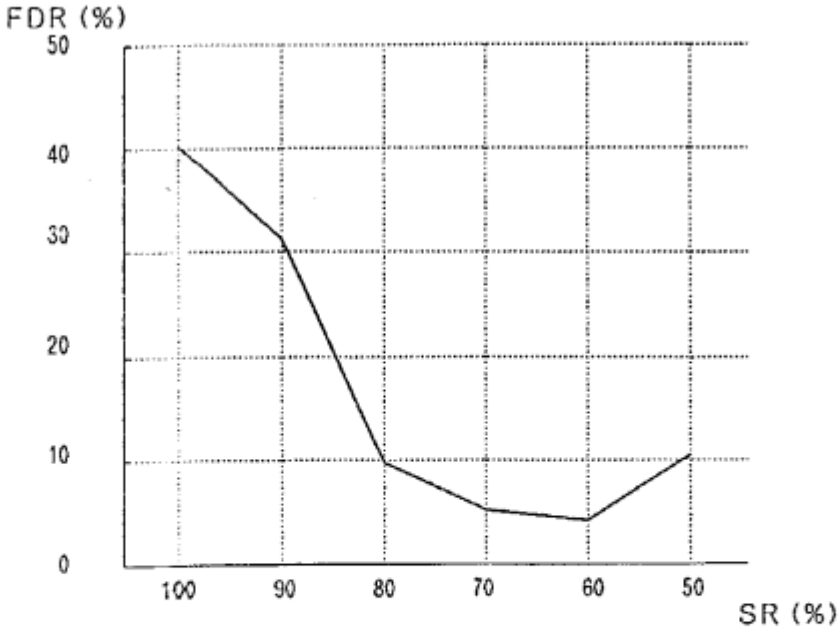

Figure 4.1.2 Relationship between SR and FDR
(in field separated hash)

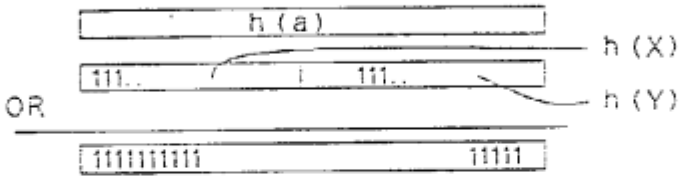

Figure 4.1.3 SSCW of "h(X,Y)" where SR=100%

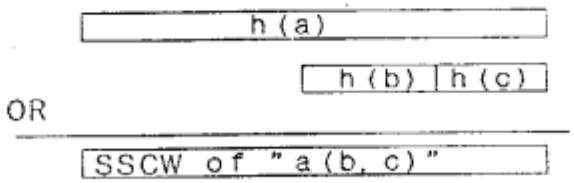

Figure 4.1.4 SSCW of "h(X,Y)" where SR=50%

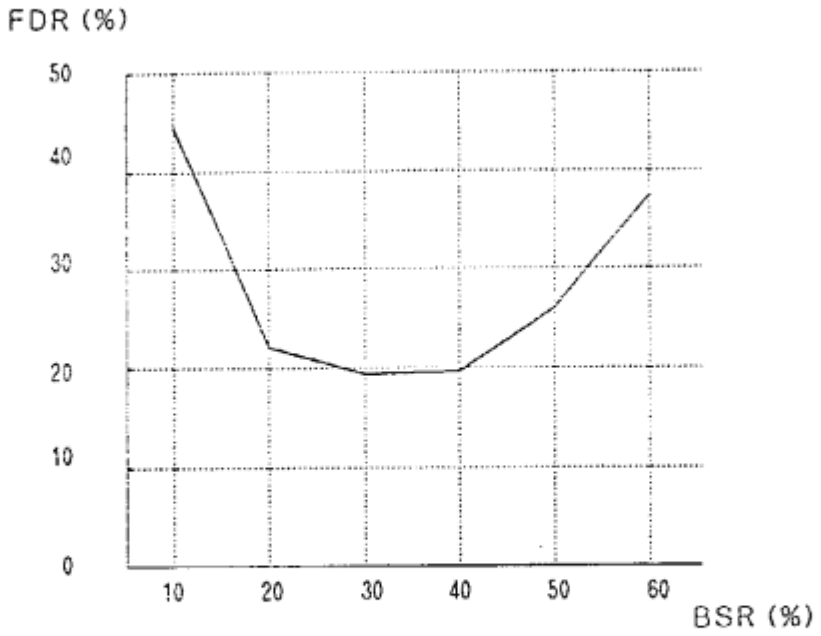

Figure 4.2.1 Relationship between BSR and SR
(in uniformly distributed hash)

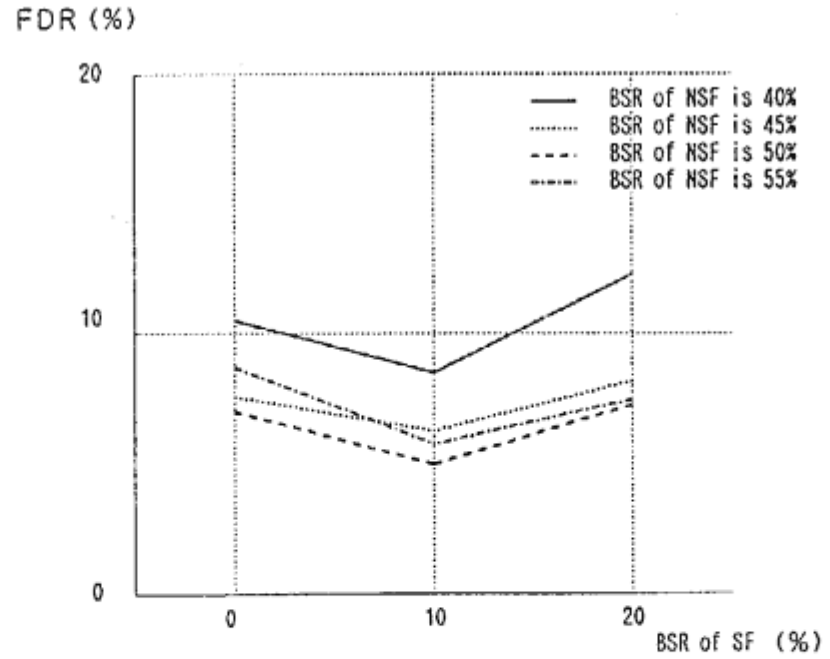

Figure 4.2.2 Relationship between BSR and FDR
(in field separated hash)

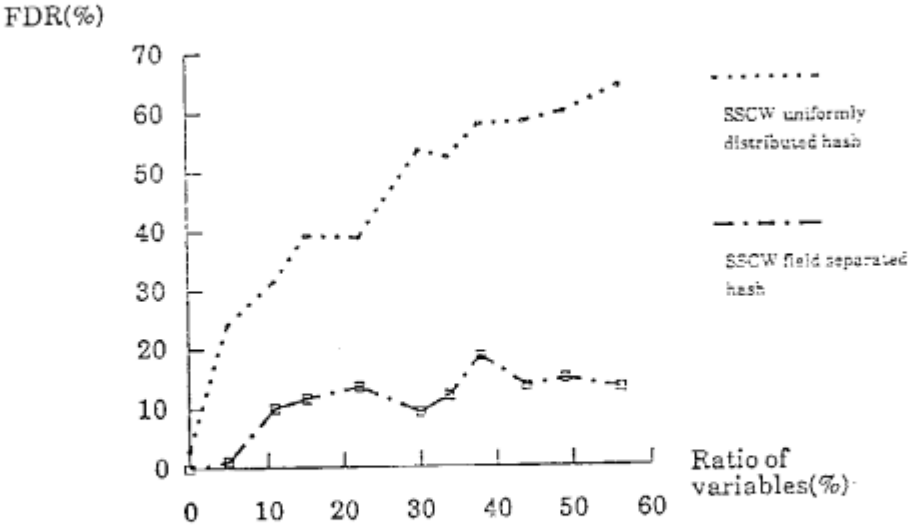

Figure 4.3.1 Effect of the variables in a term set on the FDR

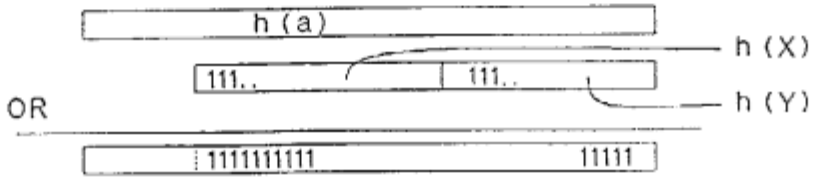

Figure 4.3.2 SSCH of "h(X,Y)"

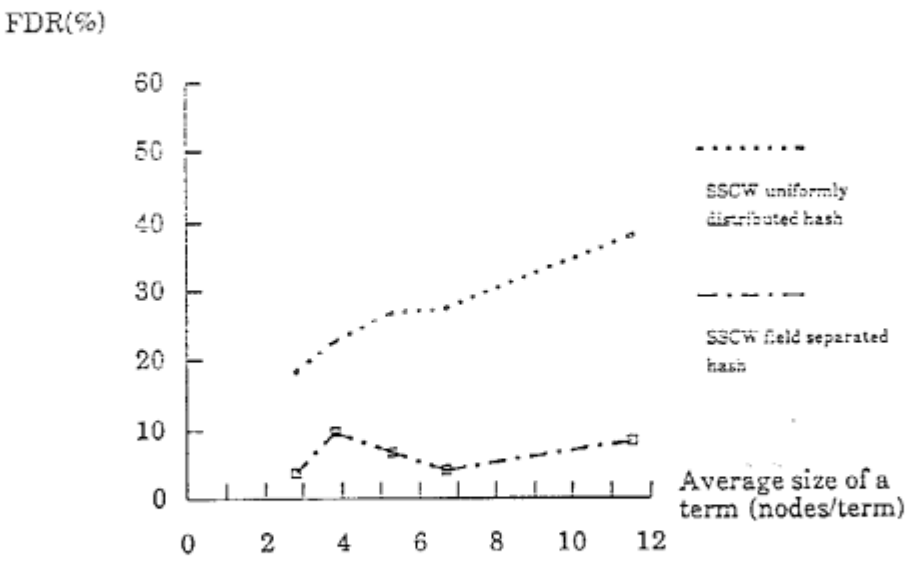

Figure 4.4.1 Effect of size of a term on the FDR

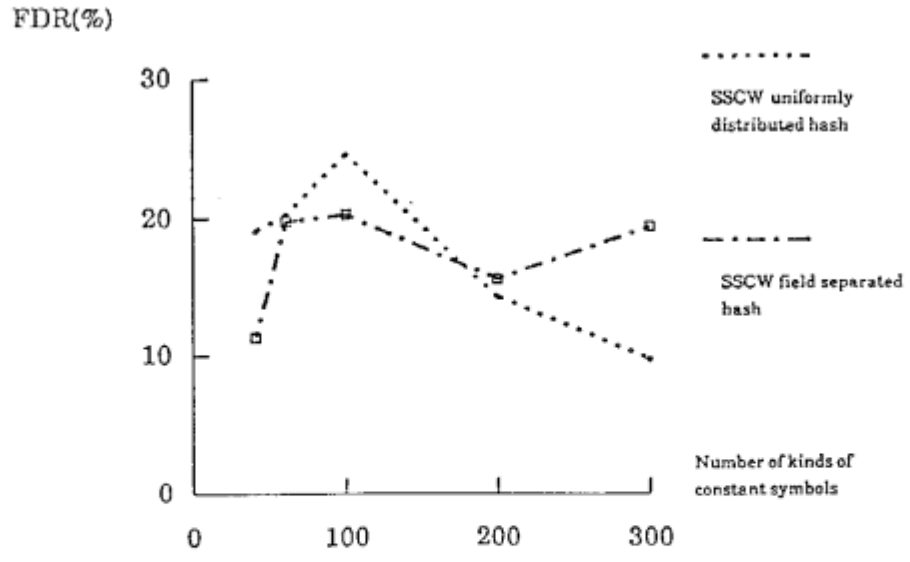

Figure 4.5.1 Effect of the number of kinds of constant symbols in a term set on the FDR

Figure 5.1 FEW of "a(b(X),c)"

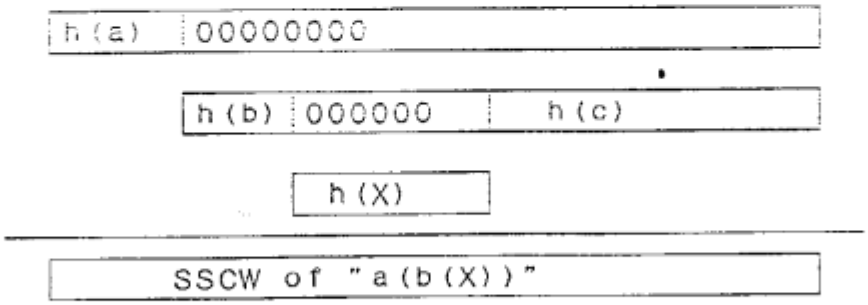

Figure 5.2 SSCW of "a(b(X),c)" with BSR of SF=0% and BSR of NSF=50%

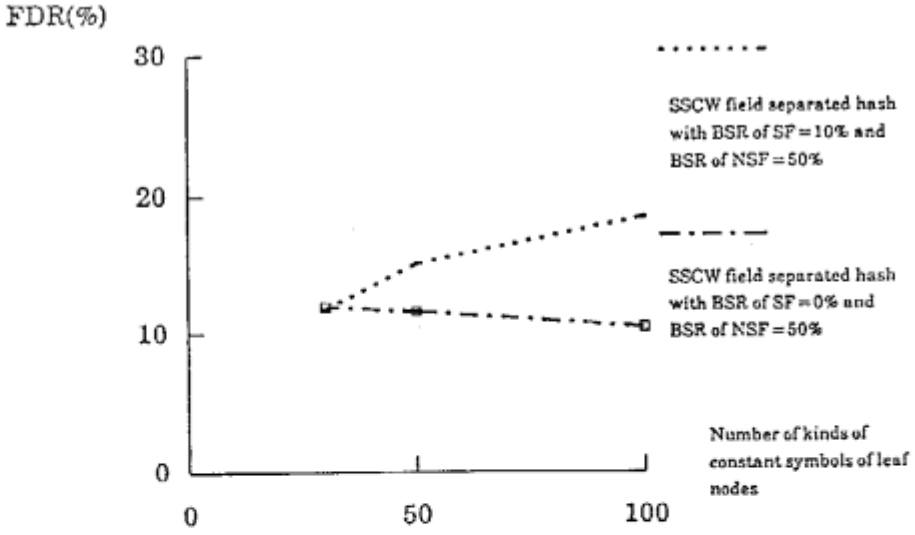

Figure 5.3 Effect of the number of kinds of constant symbols of leaf nodes to FDR

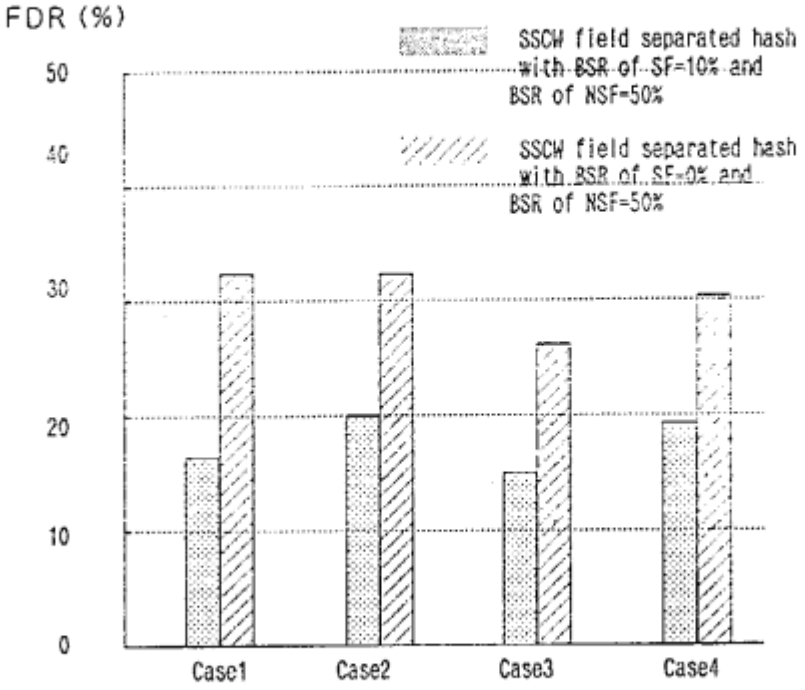

Figure 5.4 FDR where a term set with only a few variables is retrieved by a term set with many variables

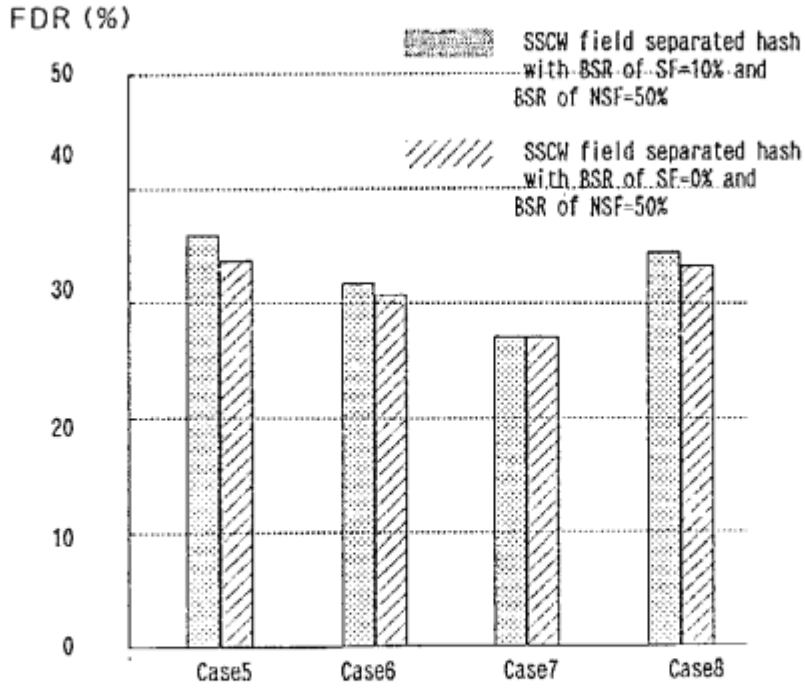

Figure 5.5 FDR where a term set with many variables is retrieved by a term set with only a few variables