An Efficient Indexing Scheme for Bit-String Signatures

C. H. Lee[1] and P. W. Huang[2]

[1]Department of Information Management, Chaoyang University of Technology

168 Gifeng E.Rd., Wufeng, Taichung, Taiwan, R.O.C.

[2]Department of Applied Mathematics, National Chung-Hsing University

Taichung 40227, Taiwan, R.O.C.

Abstract

Searching for the desired images similar to a query picture is very time-consuming in an image database system. To speed up the search process, a signature file containing the signatures associated with database images is frequently used as a filter to prune off non-promising images at the early stage of query processing. In this paper, we propose a novel structure for organizing signatures. By using this new indexing structure, the number of signatures to be examined per query is reduced significantly. In particular, the reduction ratio in examining signatures is about 51% as compared to the quick filter. As a result, image retrieval in image database systems becomes very efficient by using our method.

Keywords

Signature *Filter* *Image database* *Image retrieval* *Symbolic image*

This paper is intended for the Workshop on Databases and Software Engineering.

Contact author: C. H. Lee

[1]Email: chlee@cyut.edu.tw, Phone: 886-4-23323000 ext 7122, Fax: 886-4-23742337

[2]Email: huang@amath.nchu.edu.tw, Phone: 886-4-22860133 ext 612, Fax: 886-4-22873028

1. INTRODUCTION

In designing image database systems, many powerful spatial knowledge structures for image representation have been proposed such as 2D string [3], 2D C-string[11], and 2D C^+-string [7], etc. The common characteristic of these spatial knowledge representations is that they preserve the spatial relationships among objects in a picture to facilitate spatial reasoning and similarity retrieval. For these approaches, retrieving images similar to a query picture is done by iconic indexing based on a pre-selected spatial knowledge representation and the problem of similarity retrieval is reduced to a task of string subsequence matching which is still a very time-consuming process.

One way of improving the efficiency of similarity retrieval is to use a signature file as the spatial filter to prune unqualified images at the early stage of searching [1], [2], [8], [9], [10]. A bit-string signature is a coded binary word which records the major characteristic of a picture. Bit-string signatures are usually generated by superimposed and disjoint coding techniques [14]. Thus, if $S_q \cap S_p = S_q$, where S_q is the signature of a query picture q and S_p is the signature of a database picture p, then p is possibly a picture matched to q. With a signature file as the spatial filter, the number of images required for detailed inspection is thus reduced significantly. Since the purpose of the signature file is to reduce the search space in the image database, a sequential organization was assumed in most of the analytical works on signature extraction method [4], [5], [6]. However, searching the signature file itself may still be inefficient. There are two well-known techniques to avoid sequential search: the bit-slice approach and the two-level approach. These methods assumed that signatures are also disk-resident and emphasized the ways of reducing the number of disk accesses. In recent years, the capacity of main memory in a computer becomes much larger while the price of memory devices is significantly decreased. Thus, storing signatures in the main memory as the directory for searching the image database becomes possible and practical. In this paper, we assume that the signatures in a signature file can be loaded and stored in the main memory. Based on this assumption, we proposed a novel structure for organizing a signature file. With this new indexing

structure, the number of signatures to be examined per query is the minimum. Thus, the process of searching for qualified signatures becomes very efficient.

The remainder of this article is organized as follows. A review for the previous work about signature organization is given in Section 2. A novel structure for organizing signatures is presented in Section 3. We analyze the efficiency of our method in Section 4. Experimental results demonstrating the efficiency of our method is presented in Section 5. Finally, concluding remarks are given in the last section.

2. A REVIEW OF THE PREVIOUS WORK

Traditionally, three techniques are used to improve the performance of searching a signature file [12], [15], [16], [17]: the bit-slice approach, the two-level approach, and the partitioning approach. In bit-slice approach, let N be the total number of signatures with each containing b bits. Then, a signature file can be viewed as b strings of N bits. Assume that w is the number of 1's contained in a query signature. Since we need to examine these w bit positions when answering the query, only wN bits rather than bN bits are inspected. Let b be the length of a signature. The ratio w/b is called the weight of a signature. The bit-slice approach is not suitable for the multimedia database because query signatures in such an application usually have very high weight [17].

On the other hand, the two-level signature file structure consists of block signatures and record signatures. The record signatures are partitioned into blocks. Each block is associated with a block signature. During the signature matching process, the block signatures are searched first. Then, the record signatures of the matched blocks are searched to find the totally matched signatures. On an average the two-level signature scheme has better performance than the bit-slice approach [10]. However, there are still two major problems with the two-level signature scheme. One is the increasing density of bits set to "1" in the block signatures. The other is the rate of combinatorial errors caused by a large class of queries. They both cause query signature to qualify more objects, thereby increasing the false drop rate [17].

The third type of signature scheme is called the partitioning approach [12], [17]. In this approach, the signatures are divided into partitions in such a way that all signatures in a partition hold the same part which can be treated as the key for searching. As a result, it is possible to determine whether the signatures in a partition satisfy a query by merely examining the key. Partitions not matching the key need not be searched. Based on the partitioning approach, a quick filter for similarity retrieval of symbolic images was proposed [1].

Since the quick filter seems to have better performance, we describe this approach in more detail in the following paragraphs and use it as the benchmark for comparison with our method. In a quick filter, the signatures are clustered into blocks by a linear hashing function [13]. Let N be the number of signatures in the signature file and S_i $(i = 1, 2, \ldots, N)$ be a sequence of w binary digits b_1,b_2,...,b_w. A linear hashing function h maps the signature S_i onto the address space $\{0, 1, 2, \ldots, n-1\}$, where $2^l - 1 < n \leq 2^l$ for some integer l. The value of l is called the level of the signature file. Thus, the linear hashing function for signatures can be defined as follows:

For $l = 0$, n=1: $h(S_i, 0, 1) = 0$.

For $l > 0$:

$$h(S_i, l, n) = \begin{cases} \sum_{j=0}^{l-1} b_{w-j} 2^j, & \text{if } \sum_{j=0}^{l-1} b_{w-j} 2^j < n; \\ \sum_{j=0}^{l-2} b_{w-j} 2^j, & \text{otherwise.} \end{cases}$$

The following example illustrates the process of building a quick filter. Assume that each block contains two signatures. The six signatures to be inserted into the signature file are: $R_1 = 100001$, $R_2 = 001100$, $R_3 = 010001$, $R_4 = 000110$, $R_5 = 100010$, and $R_6 = 010011$. The signatures are inserted in the order of R_1, R_2, R_3, R_4, R_5 and R_6. The content of each block, the values of l and n are shown as below:

1. Initially, we have $P_0 = \emptyset$, $l = 0$ and $n = 1$.

2. After inserting R_1, we have $P_0 = \{R_1\}$, $l = 0$ and $n = 1$.

3. After inserting R_2, we have $P_0 = \{R_1, R_2\}$, $l = 0$ and $n = 1$.

4. After inserting R_3, we have $P_0 = \{R_2\}$, $P_1 = \{R_1, R_3\}$, $l = 1$ and $n = 2$.

5. After inserting R_4, we have $P_0 = \{R_2, R_4\}$, $P_1 = \{R_1, R_3\}$, $l = 1$ and $n = 2$.

6. After inserting R_5, we have $P_0 = \{R_2\}$, $P_1 = \{R_1, R_3\}$, $P_2 = \{R_4, R_5\}$, $l = 2$ and $n = 3$.

7. After inserting R_6, we have $P_0 = \{R_2\}$, $P_1 = \{R_1, R_3\}$, $P_2 = \{R_4, R_5\}$, $P_3 = \{R_6\}$, $l = 2$ and $n = 4$.

The organization of the resulting quick filter is shown in Table 1. The important feature of this organization is that all signatures in a block have the same 2-bit suffix.

The algorithm of signature matching by using a quick filter is re-stated as below where the following notations are used:

- l: the level of a signature file.
- P: a l-bit binary integer.
- h: a linear hashing function.
- Q: a query signature.
- $l(Q)$: the l-bit suffix of Q.
- n: the number of addressable blocks.

Algorithm: Signature Matching by Quick Filter
Input: The signature Q of a query picture.
Output: Signatures matched with Q.
For $P := h(Q, l, n)$ To $n - 1$.
    If $l(Q) \cap P \equiv l(Q)$ then
    /* access block P to find qualified signatures */
        For each signature R in P do
            If $Q \cap R \equiv Q$ then
                Output the qualified signature R.

Assume that we have a quick filter with $l = 2$ and $n = 4$ as shown in Table 1. Let Q=0100<u>10</u> be the query signature. The hash value for Q is $h(Q, 2, 4) = 2$. Thus,

signature searching starts with block P_2. Since $l(Q) \cap P_i \equiv l(Q)$ for $i = 2, 3$, the signatures in $P_2 = \{000110, 100010\}$ and $P_3 = \{010011\}$ are examined. Because $Q \cap 000110 \neq Q$, $Q \cap 100010 \neq Q$, and $Q \cap 010011 = Q$, only 010011 is returned as a qualified signature.

Although the quick filter was originally designed for disk-based signature files, it is equally valid for implementing the quick filter in the main memory. However, the performance of the filter will be measured in terms of the number of signatures examined rather than the number of disk accesses. In this paper, we compare the performance of our approach with that of the quick filter in terms of the number of signatures examined.

3. HIERARCHICAL RELATION GRAPH

A Hierarchical Relation (HR) Graph is a directed graph with the following two properties: (1) A node contains a signature; (2) If S_i and S_j are two signatures contained in nodes A and B, respectively, and node A is an immediate predecessor of node B, then $S_i \cap S_j = S_i$ with only one bit difference between S_i and S_j.

A node of an HR graph may contain a *real* or *virtual* signature. Only the real signatures are associated with database pictures. Virtual signatures has no corresponding images in the database. An example of HR graph is shown in Fig. 1.

3.1. Constructing an HR Graph

The following notations are used in the HR graph construction algorithm:

- $S = (b_1, b_2, \ldots, b_w)$ is a w-bit signature, where $b_k = 0$ or 1 for $1 \leq k \leq w$.
- $Z_a(S) = (z_1, z_2, \ldots, z_w)$ is a signature modified from S with $z_k = 0$ if $k = a$, otherwise $z_k = b_k$.
- $N(S)$ is a node containing signature S.

Algorithm: Constructing an HR graph.
Input: A set of signatures $S = \{S_1, S_2, \ldots, S_n\}$
Output: An HR graph $G = (V, E)$ for S.

(1) $V = \emptyset$; $E = \emptyset$.

(2) For each $S_i \in S$ do

(a) Generate node N to contain S_i and mark $N(S_i)$ as a "real signature" node.

(b) If $N(S_i) \notin V$, then call Insert_node($N(S_i)$).

(3) Return $G = (V, E)$.

Function: Insert_node($N(S)$)

(1) Add the node $N(S)$ to V.

(2) $c = \sum_{k=1}^{w} b_k$, where $S = (b_1 \cdots b_w)$. Let $b_{a_1} = b_{a_2} = \ldots = b_{a_c} = 1$.

(3) For $k = 1$ to c do

If $N(Z_{a_k}(S)) \notin V$ then

(a) Generate node $N(Z_{a_k}(S))$ and mark it as a "virtual signature" node.

(b) Add the edge $(N(Z_{a_k}(S)), N(S))$ to E.

(c) Insert_node($N(Z_{a_k}(S))$).

The root of an HR graph will contain a signature in which all bits are "0". The root is the only node at level 0. A node at level i contains a signature in which the number of 1's in the signature is i. To see how the above graph construction algorithm works, let us look at example shown in Fig. 2. Assume that signature 1010100 will be inserted into an empty HR graph. Because the node $N(1010100) \notin V$, we add this node to the HR graph. Since the signature "1010100" has three non-zero bits at positions 1, 3, and 5, respectively, the three predecessors of node $N(1010100)$ can be obtained by changing "1" to "0" at these positions one at a time. Thus, $N(0010100)$, $N(1000100)$, and $N(1010000)$ will be added into the HR graph subsequently. Similarly, the predecessors of $N(0010100)$ are $N(0000100)$ and $N(0010000)$; the predecessors of $N(1000100)$ are $N(0000100)$ and $N(1000000)$; the predecessors of $N(1010000)$ are $N(0010000)$ and $N(1000000)$. So the nodes $N(0000100)$, $N(0010000)$, and $N(1000000)$ are added into the graph. Finally, the root $N(0000000)$ is added into the graph. In this HR graph, there are three nodes at level 1, three nodes at level 2, and one node at level 3. The node at level 3 contains a real signature and other seven nodes contain virtual signatures.

3.2. Implementing the HR Graph

An HR graph captures the relationships of signatures in a signature file to facilitate signature matching. Assume that a signature is w-bit long. An HR graph can be implemented by an array A of size 2^w. Each element $A[i]$ of the array consists of two fields denoted by $A[i].string$ and $A[i].tag$, respectively. The content of $A[i].string$ is either *null* or a string coded by three symbols 0, 1, and x. A query signature $b = (b_1, b_2, \ldots, b_w)$ can be used to index the array. $A[b].string$ records all immediate successors of node $N(b_1, b_2, \ldots, b_w)$ in the corresponding HR graph. We replace an "x" by an "1" one at a time to find an immediate successor of the signature $(b_1, b_2, \ldots, b_w)$. The content of $A[b].tag$ is either "0" or "1". An "1" indicates that $N(b_1, b_2, \ldots, b_w)$ is a real signature node in the HR graph. Otherwise, it is a virtual signature node. For example, assume that $A[0000100].string = xxxx100$. Then, we can obtain $N(1000100)$, $N(0100100)$, $N(0010100)$, and $N(0001100)$ as the children nodes of $N(0000100)$ in the corresponding HR graph. The array structure corresponding to a HR graph is called the *adjacency-coded representation* of a signature file. Figure 3 shows an HR graph and its corresponding adjacency-coded representation.

3.3. Access Method

Assume that a database contains m pictures. Each picture p_i $(1 \leq i \leq m)$ is associated with a signature p_i^s. Let q^s be the signature corresponding to query picture q. Our goal is to find all signatures p_i^s such that $q^s \cap p_i^s \equiv q^s$. We present the algorithm of searching for all qualified signatures in an HR graph implemented by the *adjacency-coded representation* as follows.

Algorithm: Signature matching based on HR graph with adjacency-coded representation.

Input: A query signature $q^s = (q_1, q_2, \ldots, q_w)$ and a signature file A in adjacency-coded representation.

Output: The set R of all qualified signatures.

(1) $R = \emptyset$; $Uninspected\_Q = \emptyset$.

(2) If $A[q^s].tag = 1$, then $R = R \cup \{q^s\}$.

(3) Let $C = A[q^s].string$.

(4) For each symbol "x" in C

(a) Replace this "x" by "1" and all other x's by "0" to get a new string s.

(b) If $s \notin Uninspected\_Q$, then add s to $Uninspected\_Q$.

(5) If $Uninspected\_Q = \emptyset$, then return R.

Otherwise, remove an item l from $Uninspected\_Q$.

(6) If $l \cap q^s = q^s$ and $A[l].tag = 1$, then $R = R \cup \{l\}$.

(7) Let $C = A[l].string$.

If $C = null$, then Goto 5.

Otherwise, Goto 4.

Assume that we have three picture signatures {0100, 1100, 1001} in the database. The HR graph for organizing these signatures, as well as its *adjacency-coded representation* A for this graph is shown in Fig. 3. Let $q^s = 1000$ be a given query signature. Since $A[1000].tag = 0$, so 1000 is not a database signature. Because $A[1000].string = 1x0x$ and by elaborating "$1x0x$", we have $Uninspected\_Q = \{1001, 1100\}$. Since $A[1001].string = null$ and $A[1100].string = null$, no more signatures will be added to the $Uninspected\_Q$. When we examine the signatures in $Uninspected\_Q$, we can see that $1001 \cap 1000 = 1000$ and $A[1001].tag = 1$, so 1001 is a matched signature. Furthermore, $1100 \cap 1000 = 1000$ and $A[1100].tag = 1$, so 1100 is also a matched signature. After removing the two signatures 1001 and 1100 from $Uninspected\_Q$, the queue becomes empty and $\{1001, 1100\}$ is returned as the set of matched signatures.

4. ANALYSIS OF PROPOSED METHOD

In this section, we will analyze the effectiveness of our method in terms of the reduction ratio of the search space in the signature matching process.

Definition 4.1. The number of nodes of the subgraph rooted at a node x in an HR graph is called the *graph-size* of x.

Definition 4.2. A *full* HR graph for w-bit signatures is an HR graph with 2^w nodes containing signatures from $(00\ldots0)_2$ to $(11\ldots1)_2$. Thus, the number of nodes at level i in an HR graph for w-bit signatures is at most $\binom{w}{i}$.

Lemma 1: Given a full HR graph for w-bit signatures, the graph-size of the subgraph rooted at a node x of level i is 2^{w-i}.

Proof: The number of descendants at level $i+a$ of node x is $\binom{w-i}{a}$, where $1 \leq a \leq w-i$. So the graph-size of the subgraph rooted at node x of level i is equal to:

$$1 + \sum_{a=1}^{w-i} \binom{w-i}{a} = 1 + \binom{w-i}{1} + \binom{w-i}{2} + \ldots + \binom{w-i}{w-i} = 2^{w-i}.$$

Theorem 1: The average graph-size of a subgraph of a full HR graph for w-bit signatures is $(\frac{3}{2})^w$.

proof: Since

$$(2+x)^w = 2^w \binom{w}{0} x^0 + 2^{w-1} \binom{w}{1} x^1 + \ldots + 2^0 \binom{w}{w} x^w = \sum_{k=0}^{w} 2^{w-k} \binom{w}{k} x^k$$

So we have (for $x = 1$)

$$(3)^w = 2^w \binom{w}{0} + 2^{w-1} \binom{w}{1} + \ldots + 2^0 \binom{w}{w} = \sum_{k=0}^{w} 2^{w-k} \binom{w}{k} \tag{1}$$

From Lemma 1 and equation (1), the average graph-size of a subgraph of a full HR graph is

$$\frac{2^w \binom{w}{0} + 2^{w-1} \binom{w}{1} + \ldots + 2^0 \binom{w}{w}}{2^w} = \frac{\sum_{k=0}^{w} 2^{w-k} \binom{w}{k}}{2^w} = (\frac{3}{2})^w \tag{2}$$

Theorem 2: The reduction ratio of the search space of w-bit signatures organized as a full HR graph is $(3/4)^w$ on an average.

Proof: From Theorem 1, we know that if the node containing a signature matched with the query signature is at level i, then the number of nodes need to be visited is at most $(3/2)^w$. So the reduction ration of the search space is equal to $\frac{(3/2)^w}{2^w} = (3/4)^w$.

Note that signatures for multimedia databases have very high weights (Zezula et al., 1991). They can be organized as a structure close to a full HR graph. Thus, the above analytical results are valid for signatures in image database domains.

5. EXPERIMENTAL RESULTS

In this section, we evaluate the efficiency of our signature file indexing method as compared to that of a quick filter. In our experiment, there were 15 different objects from which a set of 1000 images were randomly created as the database pictures. A database image contained 5 to 12 objects. We also generated eight groups of query images; each group contained 3-5, 4-6, 5-7, 6-8, 7-9, 8-10, 9-11, or 10-12 objects, respectively. There were 100 randomly generated query images in each group. We evaluated the performance of a signature filtering method by counting the average number of signatures accessed per query. To compare with a quick filter, we assumed that each block in a quick filter contained four signatures. Let q and o be the average number of signatures accessed per query by the quick filter method and by our method, respectively. The reduction ratio is defined as $e = [(q-o)/q] \times 100\%$. This ratio represents the improvement of our method over the quick filter method.

Table 2 shows the experimental results of our method as compared to the quick filter method. As we can see, our method is always superior to the quick filter method in terms of the average number of signatures accessed per query. For example, in the case of 5-7 objects in a query picture, the average number of signatures accessed per query by the quick filter method is 70.16 while our method is 40.6. The reduction ratio is 42.13%. In the case of 8-10 objects in a query picture, the average number of signatures accessed per query by the quick filter method is 17.34 while our method is only 7.28. The reduction ratio is 58.02%. On an average, 58.89 signatures are accessed per query by the quick filter method while 35.98 signatures are accessed per query by our method. Therefore, a 50.53% reduction ratio was achieved.

6. CONCLUSIONS

Similarity retrieval is one of the most important functions in image database systems. Computing the similarity (or dissimilarity) between a query picture and the database pictures is a very time-consuming process. To speed up the query processing time, a signature file containing the signatures associated with the database images is

frequently used as a filter to prune non-promising images at the early stage of query processing.

In this paper, we proposed a novel indexing structure for organizing the signatures associated with the database images to improve the efficiency of its usage as a spatial filter. Algorithms of generating and using this indexing structure were also discussed in details in this paper. Our experimental results show that the number of signatures examined per query by using our method is much less than that of using the quick filter. Only 35.98% of signatures are examined by using our indexing structure while 58.89% of signatures need to be examined by using the quick filter method. In a multimedia database, the signatures always have high weights (i.e. many number of 1's in a signature). Our approach is particularly suitable for multimedia databases. For signatures with high weight, we showed that the reduction ratio of the search space of searching qualified signatures approaches to $(3/4)^w$ where w is the length of a signature.

References

[1] Chang, C.C., Jiang, J.H., 1996. A Spatial Filter for Similarity Retrieval, International Journal of Pattern Recognition and Artificial Intelligence, 10(6), 711-730.

[2] Chang, C.C., Lee, C.F., 1998. A Two-Level Signature File Based on a Block-Oriented Data Model for Spatial Match Retrieval, Journal of the Chinese Institute of Engineers, 21(4), 467-478.

[3] Chang, S.K., Shi, Q.Y., Yan, C.W., 1987. Iconic Indexing by 2-D Strings, IEEE Trans. on Pattern Analysis and Machine Intelligence, 9(3), 413-428.

[4] Christodoulakis, S., Faloutsos, C., 1984. Signature files: An access method for documents and its analytical performance evaluation, ACM Trans. on Info. Syst., 2(4), 267-288.

[5] Christodoulakis, S., Theodoridov, M., Ho, F., Papa, M., Pathria, et. al., 1986. Multimedia document presentation, information extraction and document for-

mation in MINOS: A model and a system, ACM Trans. off. Info. Syst., 4(4), 345-383.

[6] Faloutsos, C., Christodoulakis, S., 1987. Description and Performance analysis of signature file methods for office filing, ACM Trans. off. Info. Syst., 5(3), 237-257.

[7] Huang, P.W., Jean, Y.R., 1994. Using 2D C^+-Strings as Spatial Knowledge Representation for Image Database Systems, Pattern Recognition, 27(9), 1249-1257.

[8] Huang, P.W., Jean, Y.R., 1996. Design of Large Intelligent Image Database Systems, International Journal of Intelligent Systems, 11 , 347-365.

[9] Huang, P.W., 1997. Indexing Pictures by Key Objects for Large-Scale Image Database, Pattern Recognition, 30(7), 1229-1237.

[10] Lee, S.Y., Shan, M.K., 1990. Access Methods of Image Database, International Journal of Pattern Recognition and Artificial Intelligence, 4(1), 27-44.

[11] Lee, S.Y., Hsu, F.J., 1992. Spatial Reasoning and Similarity Retrieval of Images using 2D C-String Knowledge Representation, Pattern Recognition, 25(3), 305-318.

[12] Lee, D.L., Leng, C., 1989. Partitioned signature files: Design issues and performance evaluation, ACM Trans. Off. Info. Syst., 7(2), 158-180.

[13] Litwin, W., 1980. Linear hashing: A new tool for files and table addressing, Proc. 6th Intl. Con. on VLDB, Montreal, 212-223.

[14] Roberts, C.S., 1979. Partial-match retrieval via the method of superimposed codes, Proc. of the IEEE, 67(12), 1624-1642.

[15] Sacks-Davis, R., Kent, A., 1987. Multikey access methods based on superimposed coding techniques, ACM Trans. on Database System, 12(4), 655-696.

[16] Sacks-Davis, R., Ramamohanarao, K., 1983. A two level superimposed coding scheme for partial match retrieval, Information Systems, 8(4), 273-280.

[17] Zezula, P., Rabitti, F., Tiberio, P., 1991. Dynamic partitioning of signature files, ACM Trans. Information Systems, 9(4), 336-369.

Table 1: Organization of a quick filter.

P_0	001100	
P_1	100001	010001
P_2	000110	100010
P_3	010011	

Table 2: Experimental results

Number of objects in a query picture	Quick filter	Our method	Reduction ratio
[3,5]	186.7	127.68	31.61%
[4,6]	107.59	69.67	35.24%
[5,7]	70.16	40.6	42.13%
[6,8]	45.35	23.75	47.63%
[7,9]	28.02	13.69	51.14%
[8,10]	17.34	7.28	58.02%
[9,11]	10.36	3.78	63.51%
[10,12]	5.63	1.41	74.96%
Average	58.89	35.98	50.53%

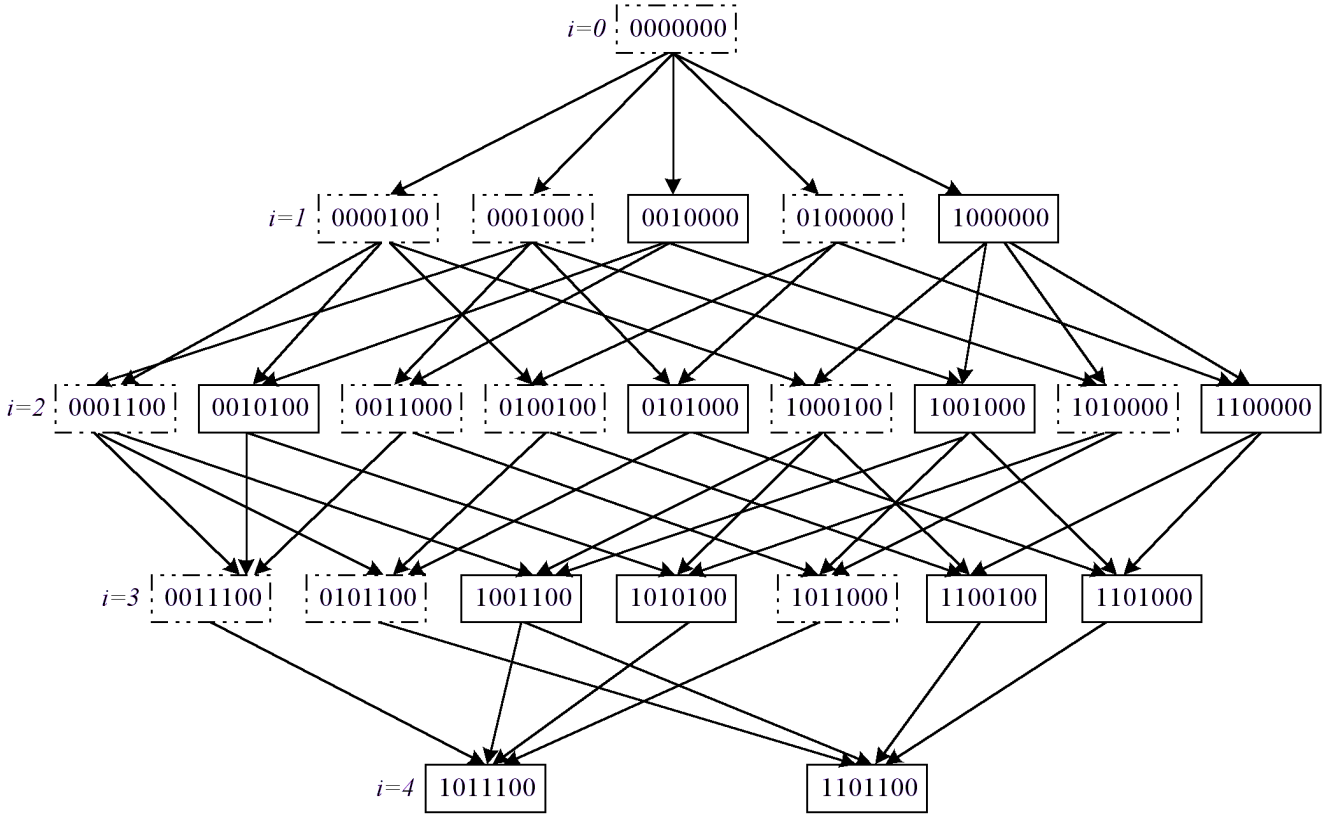


Figure 1: An example of HR graph: nodes containing real signatures are represented by solid-line boxes and nodes containing virtual signatures are represented by dotted-line boxes

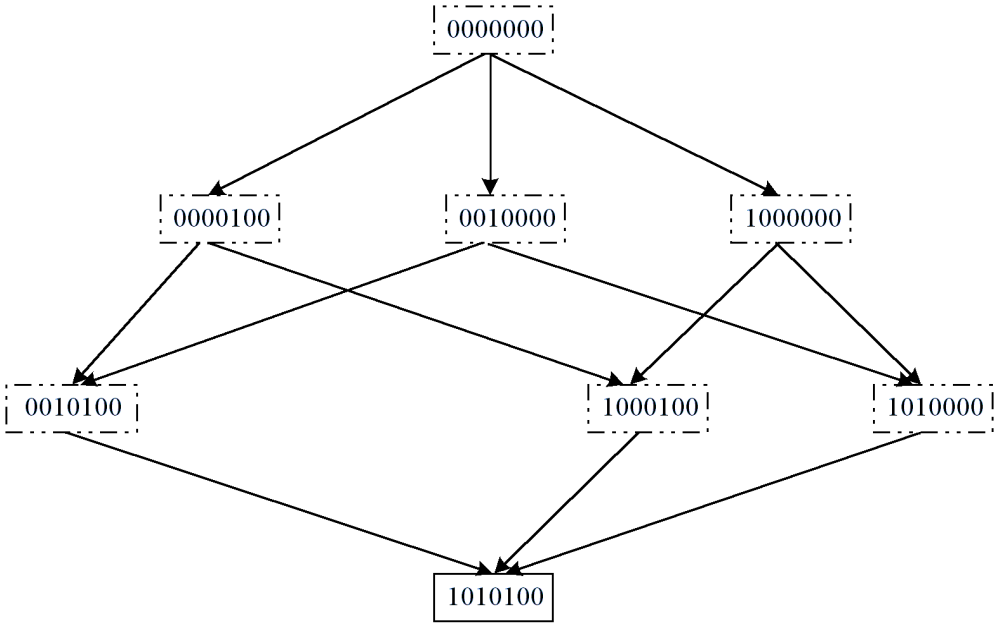
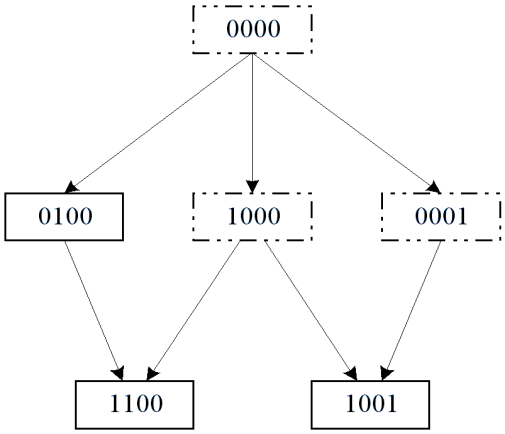


Figure 2: An HR graph created by inserting signature the "1010100".



0000	xx0x	0
0001	x001	0
	...	
0100	x100	1
	...	
1000	1x0x	0
1001	null	1
	...	
1100	null	1
	...	
1111	null	0

Figure 3: An HR graph and its adjacency-coded representation.