Adjacency Matrix Based Full-Text Indexing Models*

ZHOU Shui-geng[1], HU Yun-fa[2], GUAN Ji-hong[3]

[1](*Department of Computer Science and Engineering, Fudan University, Shanghai* 200433, *China*);

[2](*Department of Computer and Information Technology, Fudan University, Shanghai* 200433, *China*);

[3](*School of Computer Science, Wuhan University, Wuhan* 430079, *China*)

E-mail: sgzhou@fudan.edu.cn

Received January 18, 2002; accepted July 1, 2002

Abstract: With the rapid growth of online text information and user accesses, query-processing efficiency has become the major bottleneck of information retrieval (IR) systems. This paper proposes two new full-text indexing models to improve query-processing efficiency of IR systems. By using directed graph to represent text string, the adjacency matrix of text string is introduced. Two approaches are proposed to implement the adjacency matrix of text string, which leads to two new full-text indexing models, *i.e.*, adjacency matrix based inverted file and adjacency matrix based PAT array. Query algorithms for the new models are developed and performance comparisons between the new models and the traditional models are carried out. Experiments over real-world text collections are conducted to validate the effectiveness and efficiency of the new models. The new models can improve query-processing efficiency considerably at the cost of much less amount of extra storage overhead compared to the size of original text database, so are suitable for applications of large-scale text databases.

Key words: information retrieval; full-text indexing; inverted file; PAT array; adjacency matrix; model

The rapid development and wide application of World Wide Web (WWW) leads to explosively increasing of online text information. More and more people surf the Web for information. Such a situation poses new challenges to researchers and scientists in the information retrieval (IR) field. That is, how to effectively and efficiently organize and manage massive text information on Web and how to search what the users need on Web accurately and completely[1]. Currently, the popular solution is Web search engine, which is essentially an IR system where text-indexing model constitutes its core technique[2]. As the amount of text information exponentially increases and the number of user queries rapidly grows, query-processing efficiency becomes the major bottleneck of Web search engines. Take AltaVista[3], one of the largest Web search engine systems under running, for example. In 1998, the overall AltaVista system was running on 20 multi-processor machines, all of them have more than 130Gb of RAM and over 500Gb of disk space. Only query processing consumes more than 70% of these resources[1].

To a great extent, efficiency of information retrieval systems depends on the underlying text indexing models. That is the reason why efficient text-indexing techniques have been the hot research topic in IR field[4]. Up to date,

* Supported by the National Natural Science Foundation of China under Grant No.60173027 (国家自然科学基金); the Natural Science Foundation of Hubei Province of China under Grant No.2001ABB050 (湖北省自然科学基金)

ZHOU Shui-geng was born in 1966. He is an associate professor at the Department of Computer Science and Engineering, Fudan University. His research interests are in database, data warehousing, data mining and information retrieval. **HU Yun-fa** was born in 1940. He is a professor and doctoral supervisor of the Department of Computer and Information Technology, Fudan University. His current research areas are database, knowledge base and digital library system. **GUAN Ji-hong** was born in 1969. She is an associate professor at the School of Computer Science, Wuhan University. Her current research areas are spatial database, spatial data mining and geographic information systems.

several text-indexing models have been developed, in which widely accepted are inverted files[5,6], signature files[1,7], and PAT (Patricia tree) array[1,8,9]. Owing to their advantage of fast response and easy implementation, inverted file and its variations have been used in most text database systems and search engines[2].

Generally, text-indexing models can be classified into two categories, i.e., character-based indexing and word (or keyword)-based indexing. The first step toward word-based indexing is to extract meaningful words from text string, which is not an easy task that may require deep understanding of the context, especially for Chinese text. Conversely, character-based indexing methods index all characters appearing in text database; so character-based indexing is also referred to as full-text indexing. Full-text indexing IR systems' main advantage is that they avoid the complicated and expensive process of semantic indexing. From the end-user point of view, full text searching of on-line documents is appealing because a valid query is just any word or sentence of the document.

This paper intends to develop new full-text indexing models of higher efficiency. By seeing a text database as a directed graph and combining the concept of adjacency matrix of directed graph with the structures of traditional inverted file and PAT array, we propose two new full-text indexing models, which we call adjacency matrix based inverted file and adjacency matrix based PAT array respectively. The new models can improve considerably query-processing efficiency of IR systems at the cost of much less amount of extra space overhead compared to the size of original text database, thus are suitable for applications of large-scale text databases and can be used as an effective way to update the IR systems under running.

1 Preliminaries

Let Σ be an *alphabet*: a finite, ordered set of characters. For an arbitrary character l in Σ, there is an associated natural integer i, which indicates the position of l in Σ, and l is also denoted by l^i. In the case of Chinese text databases, Σ corresponds to the Standard Chinese Characters Base, such as GB2312-80, where the region-position code of each character assigned in GB2312-80 may be used as the character's position identifier in Σ (The alternative may be the Unicode system). For English text collections, Σ is a set of letters, digits, punctuation marks and other symbols that may occur in English text, and the ASCII code of each character in Σ is taken naturally as its position identifier in Σ. A text string or simply string over Σ is a finite sequence of characters from Σ. Specifically, a string has no characters at all is called the empty string and is denoted by ε. To avoid confusion, we generally use u, v, w, x, y, z to denote strings. The length of a string is its length as a sequence. We denote the length of a string w by $|w|$. Alternatively a string w can be considered as a function $w:\{1,\ldots,|w|\}\rightarrow\Sigma$; the value of $w(j)$, where $1\leq j\leq|w|$, is the symbol in the jth position of w. To distinguish identical symbols at different positions in a string, we refer to them as different occurrences of the symbol. That is, the symbol $l\in\Sigma$ occurs in the jth position of the string w if $w(j)=l$.

Two strings over the same alphabet can be combined to form a third by the operation of concatenation. The concatenation of strings x and y, written xy, is the string x followed by the string y; formally, $w=xy$ if and only if $|w|=|x|+|y|$, $w(j)=x(j)$ for $j=1,\ldots,|x|$, and $w(|x|+j)=y(j)$ for $j=1,\ldots,|y|$. A string v is a substring of a string w if and only if there are strings x and y such that $w=xvy$. Both x and y could be ε, so every string is a substring of itself. If $w=xv$ for some x, then v is a suffix of w; if $w=vy$ for some y, then v is a prefix of w. If x is a substring of string w, then we denote $P(w, x)$ the set of positions of x occurring in w, or the occurrences of x in w.

For a string w of limited length, let us pad artificially at its right end with an infinite number of null (or any special characters that is not included in Σ). Then, a semi-infinite string (abbreviated to sistring) of string w is the sequence of characters starting at a certain position within w and continuing to the right. Obviously, two sistrings starting at different positions are always different. For the simplification of description, meanwhile to guarantee that no one sistring be a prefix of another, it is enough to end the string w with a unique end-of-string symbol that does not appear in Σ. Thus, sistrings can be unambiguously identified by their starting position. That is to say, $P(w, x)$ is a

singleton if x is a sistring of string w. The result of a lexicographic comparison between two sistrings is based on the text of the sistrings, instead of their positions.

Definition 1. A text database TB is a collection of text documents, each of which is a string over Σ. Neglecting the boundary between any two adjacent documents, a text database can be seen as a long string, whose length is the sum of the lengths of all documents in the text database. We denote $|TB|$ the length of text database TB. From the point of sistring's view, text database TB corresponds to a sequence of sistrings.

In what follows, we treat a text database as a string when discussing inverted file, and see it in the point of sistring's view while dealing with PAT array.

Definition 2. Suppose V is the set of all unique characters appearing in text database TB, $V=\{t_i \mid i=1\sim|V|\} \subseteq \Sigma$. For character t_i in V, it occurs at different positions in the string of text database TB. Let p_i be the set of positions that t_i occurs in TB, denote $p_i=\{p_{i1}, p_{i2},\ldots, p_{i|pi|}\}$ where $|p_i|$ is the occurrence number of t_i in TB, p_{ij} is the jth position where t_i occurs in TB (counting from the starting point of TB string). The full-text indexing inverted file can be written formally as follows.

$$\{\langle t_i, p_i\rangle \mid (i=1\sim|V|)\}. \tag{1}$$

Practically, indexed terms and the corresponding occurrences are stored separately, *i.e.*, splitting (1) into two parts:

$$\{\langle t_i, pt_i\rangle \mid (i=1\sim|V|)\}, \tag{2}$$

$$\{p_i \mid (i=1\sim|V|)\}. \tag{3}$$

Above, pt_i is a pointer to p_i. In (2), all indexed terms are sorted lexicographically and stored with corresponding pointers sequentially in the indexed file, while occurrences in (3) are stored sequentially in the posting file.

Definition 3. Suppose V is the set of all unique characters appearing in text database TB, $V=\{t_i \mid i=1\sim|V|\} \subseteq \Sigma$. Let TB be denoted by a string $w=c_1c_2\ldots c_n\$(n=|TB|)$ where c_i is the ith character in w and $c_i \in V$, \$ is the assumed unique end-of-string symbol that does not appear in Σ. String w corresponds to a sequence of sistrings:

$$Sis=\langle sis_1, sis_2,\ldots, sis_n\rangle \tag{4}$$

where $sis_i = c_ic_{i+1}\ldots c_n\$$ indicates the ith sistring of w. Sorting the sistrings in (4) lexicographically, we get a new sequence of sistrings:

$$PSis=\langle psis_1, psis_2,\ldots, psis_n\rangle. \tag{5}$$

Obviously, for each sistring $psis_i$ in $PSis$, there is a sistring sis_j in Sis that is equal to $psis_i$ while i and j are not necessarily the same. (5) is the full-text indexing PAT array of text database TB. In practice, the positions of sistrings, instead of the sistrings themselves, are used in (5).

Definition 4. For a text database TB with a set V of all unique characters appearing in TB, there exists a directed graph $TBG=\langle V_g, E_g\rangle$ where V_g is the set of vertices, each of which corresponds to a unique character appearing in TB, i.e., $V_g=V$; E_g is the set of directed edges, each of which corresponds to a pair of adjacent characters appearing in TB and its direction points from the first character to the second one. We call TBG the directed graph of text database TB.

Definition 5. As defined in graph theory, a weighted directed graph is a directed graph in which each edge has an associated value. In the context of this paper, the value associated with each directed edge is the position of the character corresponding to the directed edge's source vertex. Considering that some adjacent-character pairs occur at different positions in the text database, that is, in the directed graph of text database there exists the case of multiple directed edges having similar starting and end vertices. We compact all directed edges with similar starting and end vertices to one directed edge and unite all values associated with these edges to a set of values. We call the result graph weighted directed graph of text database. Formally, text database TB corresponds to a weighted directed graph $WTBG=\langle V_w, E_w, L_w\rangle$ where $V_w = V$ is the set of vertices, E_w is the set of directed edges, and L_w is the set of values associated with all directed edges in E_w. Denote $L_w(l^i, l^j)$ the set of values associated with directed edge l^il^j, then we have

$$L_w(l^i, l^j) = P(TB, \text{“}l^i l^j\text{”}), \tag{6}$$

$$L_w = \{L_w(l^i, l^j) \mid \forall l^i l^j: l^i l^j \in E_w\}. \tag{7}$$

Example 1. Given a Chinese text string *w*:“我们的国家，我们的人民，你们的国家，你们的人民，他们的国家，他们的人民。”. There are totally eleven unique characters appearing in *w*, in which nine are Chinese characters, the rest two are punctuation marks. Theses unique characters constitute the vertices set V_w ={“我”, “你”, “他”, “们”, “的”, “人”, “民”, “国”, “家”, “，”, “。”}. Fourteen unique adjacent-character pairs constitute the directed edges set E_w ={“我们”, “们的”, “的国”, “国家”, “家，”, “，我”, “的人”, “人民”, “民，”, “，你”, “你们”, “，他”, “他们”, “民。”}. Values associated with these directed edges are as follows: L_w (“我”, “们”)={1, 7}, L_w (“你”, “们”)={13, 19}, L_w (“他”, “们”)={25, 31}, L_w (“，”, “我”)={6}, L_w (“，” , “你”)={12, 18}, L_w (“，” , “他”)={24, 30}, L_w (“们” , “的”)={2, 8, 14, 20, 26, 32}, L_w (“的” , “国”)={3, 15, 27}, L_w (“的” , “人”)={9, 21, 33}, L_w (“家” , “，”)={5, 17, 29}, L_w (“民” , “，”)={11, 23}, L_w (“民” , “。”)={35}, L_w (“国” , “家”)={4, 16, 28}, L_w (“人” , “民”)={10, 22, 34}. Figure 1 illustrates the weighted directed graph of string *w*.

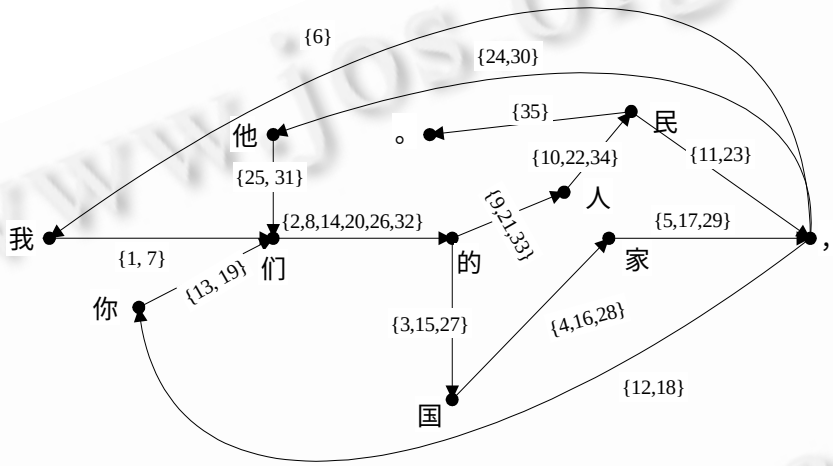

Fig.1 Weighted directed graph of string *w*

2 Adjacency Matrix Based Inverted File and PAT Array

Definition 6. For a text database, there is a corresponding weighted directed graph that associates with an adjacency matrix. Thus, a text database has an adjacency matrix where each matrix element corresponds to a set of values associated with the corresponding directed edge. Formally, let $V \in \Sigma$ be the set of all unique characters appearing in text database *TB*, and denote *D* the adjacency matrix of *TB*, we have

$$D=[d_{ij}], \tag{8.1}$$

$$d_{ij}= d(l^i,l^j)=L_w (l^i, l^j) = P(TB, \text{“}l^i l^j\text{”}). \tag{8.2}$$

We term matrix *D* the adjacency matrix based full text indexing model of text database *TB*.

Example 2. Given the string *w* in example 1, according to definition 6 and the weighted directed graph illustrated in Fig.1, it is easy to build the adjacency matrix indexing model of *w*. Let V_w={“我”, “你”, “他”, “们”, “的”, “人”, “民”, “国”, “家”, “，”, “。”}. The text indexing model *D* is a 11×11 matrix of which the elements are as follows: $d_{14}= L_w$ (“我”, “们”)={1, 7}, $d_{24}= L_w$ (“你”, “们”)={13, 19}, $d_{34}= L_w$ (“他”, “们”)={25, 31}, $d_{45}= L_w$ (“们” , “的”)={2, 8, 14, 20, 26, 32}, $d_{56}= L_w$ (“的” , “人”)={9, 21, 33}, $d_{58}= L_w$ (“的” , “国”)={3, 15, 27}, $d_{67}= L_w$ (“人” , “民”)={10, 22, 34}, $d_{7\ 10}= L_w$ (“民” , “，”)={11, 23}, $d_{7\ 11}= L_w$ (“民” , “。”)={35}, $d_{89}= L_w$ (“国” , “家”)={4, 16, 28}, $d_{9\ 10}= L_w$ (“家” , “，”)={5, 17, 29}, $d_{10\ 1}= L_w$ (“，”, “我”)={6}, $d_{10\ 2}= L_w$ (“，” , “你”)={12, 18}, $d_{10\ 3}= L_w$ (“，” , “他”)={24, 30}. The other matrix elements are empty set Φ. Figure 2 illustrates the adjacency

matrix of string *w*.

$$D=\begin{vmatrix}
\varnothing & \varnothing & \varnothing & d_{14} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & d_{24} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & d_{34} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing & d_{45} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing & \varnothing & d_{56} & \varnothing & d_{58} & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & d_{67} & \varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & d_{710} & d_{711} \\
\varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & d_{89} & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & d_{910} & \varnothing \\
d_{101} & d_{102} & d_{103} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing
\end{vmatrix}$$

Fig.2 Adjacency matrix of string *w*

Practically, there are two ways to implement the adjacency matrix based full-text indexing model:

1) Seeing matrix element d_{ij} in D as a set of positions of adjacent-character pair $l^i l^j$ occurring in *TB*, and sorting all elements of d_{ij} in positional order, then d_{ij} is equivalent to the inverted list of indexed term $l^i l^j$.
2) Treating matrix element d_{ij} as a set of positions of the sistrings whose prefix is the adjacent-character pair $l^i l^j$, and sorting all elements of d_{ij} in lexical order of the corresponding sistrings, then d_{ij} is similar to the PAT array of sistrings with a prefix of $l^i l^j$.

To distinguish this two different implementations, we use $D^1$and D^2 to denote the implemented adjacency matrices of approach 1) and approach 2) respectively, and term D^1 the adjacency matrix based full-text indexing inverted file, D^2 the adjacency matrix based full-text indexing PAT array.

Formally, D^1 is a kind of reorganization of the traditional inverted file, *i.e.*, transforming the character-based inverted file to a kind of adjacent-character pair based inverted file and organizing all inverted lists in the form of adjacency matrix. Conversely, the traditional inverted file is a kind of compression of D^1, that is, to compact the adjacent-character pair based inverted file to the character-based inverted file. Analogously, D^2 is a kind of decomposition of the traditional PAT array, *i.e.*, splitting a long PAT array into a collection of short PAT arrays such that each short PAT array corresponds a set of sistrings having a similar prefix in the form of adjacent-character pair; on the contrary, the traditional PAT array is a kind of aggregation of D^2. Based on the definitions above, it is straightforward to give the algorithms for transformations between D^1 and the traditional inverted file, D^2 and the traditional PAT array as follows.

Algorithm 1. Build D^1 from traditional inverted file.

Input: indexing lists of any two indexing terms in text database: $\{l^i: p_{i1},\ldots, p_{in}\}$, $\{l^j: p_{j1},\ldots, p_{jm}\}$.

Output: $D^1=[d_{ij}]$.

Process:

$$d_{ij} = L_w(l^i, l^j).$$
$$= \{k \mid \forall k\ (\exists\, p_{ik_1} \exists\, p_{jk_2} : p_{ik_1} = k \text{ and } p_{jk_2} = k+1 \text{ while } 1 \le k_1 \le n \text{ and } 1 \le k_2 \le m)\}.$$

Algorithm 2. Build traditional inverted file from D^1.

Input: $D^1=[d_{ij}]$.

Output: $\{l^i: p_i\}$.

Process:

$$p_i = \bigcup_{j=1}^{|V|} d_{ij}.$$

Here, "$\cup$" indicates the set union operator.

Algorithm 3. Build D^2 from traditional PAT array.

Input: PAT array $PSis=\{psis_1, psis_2,\ldots, psis_n\}$.

Output: $D^2=[d_{ij}]$, $d_{ij} = L_w(l^i, l^j)$.

Process:

1) Find the first sistring $psis_k$ matching with "$l^i l^{j*}$" in *PSis* by binary search.
2) Starting from $psis_k$, search continuously in *PSis* the other sistrings matching with "$l^i l^j$*" in the right and left directions till no such sistring can be found.
3) Assume $psis_{k1}$ and $psis_{k2}$ are the two sistrings that match with "$l^i l^{j*}$" and locate at the furthest right and left positions in *PSis* respectively, then

$d_{ij} = L_w(l^i, l^j)$

$= \{psis_{k1}, psis_{k1+1}, ..., psis_k, ..., psis_{k2-1}, psis_{k2}\}(1 \le k1 \le k \le k2 \le n)$.

Algorithm 4. Build traditional PAT array from D^2.

Input: $D^2=[d_{ij}]$.

Output: PAT array $PSis=\{psis_1, psis_2, \ldots, psis_n\}$.

Process:

$PAT = \bigcup_{i=1}^{|V|} \bigcup_{j=1}^{|V|} d_{ij}$.

Similarly, "$\cup$" is the set union operator.

3 Query Processing

3.1 Query processing based on D^1

Note that in this paper a query is an arbitrary character string and the result is a set of documents containing the query string. Query processing based on D^1 is essentially a kind of set operation. Following is a theorem about D^1 based query processing.

Theorem 1. Let "$l_1 l_2 \ldots l_n$" be a query string, $q(l_1 l_2 \ldots l_n)$ the query result, then

1) When $n=2k$, $q(l_1 l_2 \ldots l_n) = d^0_{12} \cap d^2_{34} \ldots \cap d^{2k-4}_{2k-3\,2k-2} \cap d^{2k-2}_{2k-1\,2k}$;
2) When $n=2k+1$, $q(l_1 l_2 \ldots l_n) = d^0_{12} \cap d^2_{34} \ldots \cap d^{2k-4}_{2k-3\,2k-2} \cap d^{2k-2}_{2k-1\,2k} \cap d^{2k-1}_{2k\,2k+1}$.

Here, $d^m_{ij} = \langle d(l_i, l_j) - m \rangle = \{(x-m) \mid \forall x : x \in d(l_i, l_j)\}$.

Proof. According to Definition 6, it is easy to prove Theorem 1. We omit the details here.

In order to improve query-processing efficiency, we must reduce the number of disk accesses or set intersections. Some techniques are available as follows.

1) Reducing the number of set intersections. That is equivalent to reducing the number of matrix elements involved in set intersection. We can use the optimizing technique proposed in Ref.[4] to cut down matrix elements involved in query processing, which utilizes a directed graph method to optimize the search path at the cost of a little more space overhead.
2) Reducing the number of elements in the sets involved in intersection operation. By using of the properties of query string, it is possible to cut down the number of elements in the sets involved in intersection operation. Here consider a special case: substrings occurring repeatedly in the query string. Suppose "$l_i l_{i+1}$"$(1 \le i \le n-1)$ occurs twice in the query string $l_1 l_2 \ldots l_n$ at an interval of k, then the valid elements for intersection operation in $d(l_i, l_{i+1})$ is $d(l_i, l_{i+1}) \cap \langle d(l_i, l_{i+1}) + k \rangle$. Actually, there exist many of such cases, which we cannot enumerate completely here.
3) Improving the efficiency of set intersection. If the text database is very large, the sizes of sets involved in query processing will be large too. Note that intersecting large sets is also time consuming. Given two sets *A*

and B that include m and n elements respectively (Let $m \le n$), a more efficient way to carry out intersection between A and B is 1) sorting the elements of set A and set B in advance; 2) comparing the elements in the larger set with the elements in the smaller one. In such a way, m and n are the lower-bound and upper-bound of element comparisons needed for $A \cap B$ respectively.

3.2 Query processing based on D^2

Algorithm 5. Query processing based on D^2.

Input: Full-text indexing model D^2 and query string "$l_1 l_2 ... l_n$".

Output: $q(l_1 l_2 ... l_n)$.

Process:

1) Based on D^2 and "$l_1 l_2$", obtain PAT array $d(l_1, l_2)=\{psis_1, psis_2, \ldots, psis_m\}$.
2) Search the first sistring $psis_k$ matching with "$l_1 l_2 ... l_n$*" in $d(l_1, l_2)$ by binary search.
3) Starting from $psis_k$, search continuously in $d(l_1, l_2)$ the other sistrings matching with "$l_1 l_2 ... l_n$*" in the right and left directions till no such sistring can be found.
4) Assume $psis_{k1}$ and $psis_{k2}$ are sistrings that match with "$l_1 l_2 ... l_n$*" and locate at the furthest right and left positions in $d(l_1, l_2)$ respectively, then

$$q(l_1 l_2 .. l_n)=\{psis_{k1}, psis_{k1+1}, ..., psis_k, ..., psis_{k2-1}, psis_{k2}\} (1 \le k1 \le k \le k2 \le m).$$

4 Comparisons with Traditional Indexing Models

Suppose the size of text database TB is N ($N=|TB|$), the number of unique characters in the text database is m ($m=|V|$), and the length of query is l_q. Furthermore, for the simplicity of analysis, we assume all characters occurring in the text database at equal probability. Thus the average size of elements in adjacency matrix D^2 is $E \approx N/(m*m)$.

4.1 Traditional inverted file vs. D^1

1) Time cost for indexing building: $O(N)$.
2) Space overhead

a) D^1: $O(N+m*m)$.

b) Inverted file: $O(N+m)$.

3) Search time cost

a) D^1: At most $(l_q+1)/2$ disk accesses and $(l_q-1)/2$ set intersections.

b) Inverted file: l_q disk accesses and (l_q-1) set intersections. Furthermore, considering that the average size of matrix elements in D^1 is about 1/m of the size of inverted file, which will result in higher query-processing efficiency.

Note that 1) disk access is the major factor that influences the value of query efficiency; 2) for large-scale text database, we have $N>>m$ because m is limited and N can grow as the text database expands. That is to say, the extra space overhead of D^1 is much less than the size of original text database. In the case of English text collections, m is not greater than 256. While in Chinese text environment, m is generally between 6000 and 8000. Thus, adjacency matrix based inverted file model can achieve more 50% benefit of query efficiency at the cost of much less extra space overhead compared with the size of original text database.

4.2 Traditional PAT array vs. D^2

1) Time cost for indexing building: $O(N \log(N))$.
2) Space overhead

a) D^2: $O(N+m*m)$.

b) PAT array: $O(N)$.

3) Search time cost

a) D^2: At most $4\log_2(E)$ disk accesses and $2\log_2(E)-1$ comparisons.

b) PAT array: At most $4\log_2(N)$ disk accesses and $2\log_2(N)-1$ comparisons.

Let us consider the worst case. For large-scale text databases, we have $N >> m$, which means that the extra storage overhead of D^2 is negligible compared with the size of original text database. At the same time, D^2 benefits $8\log_2(m)$ disk accesses and $4\log_2(m)$ comparisons less than traditional PAT array. Specifically, suppose the size of text database N=640Mb, at most 117 disk accesses are requested for traditional PAT array. In the case of English text collections, let m=256, then 64 disk accesses are cut down by D^2, *i.e.*, the improvement ratio of query efficiency is 54%. For the Chinese text environment, let m=6763 (Taking GB2312-80 for example), then 101 disk accesses are cut down by D^2, that is, a query efficiency improvement ratio of 86%. Certainly, the results are approximate estimation, which may deviate somewhat from the realistic values. However, because each matrix element in D^2 is only part of the entire PAT array of the text database, D^2 can still outperform the traditional PAT array.

Finally, we have a comparison between D^1 and D^2. Note that there is no much difference in space overhead between D^1 and D^2. However, it is noteworthy to analyze the difference in their search time cost. Still, we consider disk access the major bottleneck of query efficiency. Obviously, query efficiency of D^1 is related to the length of query string, while query efficiency of D^2 depends on the size of matrix element involved in query processing. When disk accesses of the two models are equal, we have

$$(l_q+1)/2 \approx 4\log_2(E). \tag{9}$$

That is,

$$(l_q+1)/2 \approx 4\log_2(N/(m*m)). \tag{10}$$

From (10), we obtain

$$l_q \approx 8\log_2(N/(m*m))-1. \tag{11}$$

Equation (11) sets an approximate criterion about when to handle query using D^1 and D^2. We carry out further estimation in the following two specific cases:

1) In the case of English text database, let N=640M and m=256, we have $l_q \approx 105$. That is to say, when the length of query is shorter than 105 characters, it is more efficient to handle queries using D^1 than using D^2. On the contrary, if the length of query is longer than 105 characters, D^2 is favorable.

2) Under the Chinese text environment, let N=640M and m=6763, we can obtain $l_q \approx 30$, *i.e.*, when the length of query is shorter than 30 characters, it is more efficient to handle query using D^1 than using D^2. Otherwise, it is favorable to use D^2 to handle query.

For a certain text database, if the query is short, it is more efficient to process query by using D^1 than using D^2. Otherwise, it is favorable to use D^2. In realty, it is likely to accommodate the two query modes simultaneously in the same text database. In implementation, we can establish the indexing matrix according to D^2, then choose query mode D^1 or D^2 in terms of query length.

5 Experimental Results

Experiments are carried out over five real world Chinese text collections (listed in Table 1) to validate the feasibility and efficiency of the new models. These text collections contain mainly Chinese classics, contemporary Chinese novels and documents from several Chinese BBS sites. The experiments are conducted on a PC with a PIII

500MHz CPU and 512Mb RAM. The goal of experiments is to measure the improvement ratio of query processing efficiency of the new indexing models. We define the query processing efficiency improvement ratio as follows.

$$r = (t_{old} - t_{new})/t_{old} \times 100\%. \quad (12)$$

Here, t_{old} represents the time consumed for handling one or more queries with the traditional indexing models, i.e., inverted file and PAT array, while t_{new} indicates the time used for handling the same queries with the new indexing models, i.e., adjacency matrix based inverted file and adjacency matrix based PAT array. To make the experimental results more reasonable and reliable, we chose manually 1000 different queries for testing. The query processing efficiency is evaluated on the average time cost in handling the 1000 different queries. Each query is a Chinese text string with from 2 to 25 Chinese characters, and its queried result is requested not to be empty. Table 2 lists the query number distribution over query length of the 1000 different queries. The number of short queries is larger than that of the long queries, which conforms to the realistic situation of user query delivery.

Table 1 Test text collections

Text database	TC-1	TC-2	TC-3	TC-4	TC-5
Size (Mb)	14.9	39.0	97.6	182.2	500.4

Table 2 Query number distribution over query length of the 1000 processed queries

l_q (characters)	2	5	7	10	12	15	17	20	22	25
N	150	150	150	100	100	100	100	50	50	50

The experimental results are presented in Fig.3, which show that the new models can improve considerably the query processing efficiencies of the traditional inverted file and PAT array.

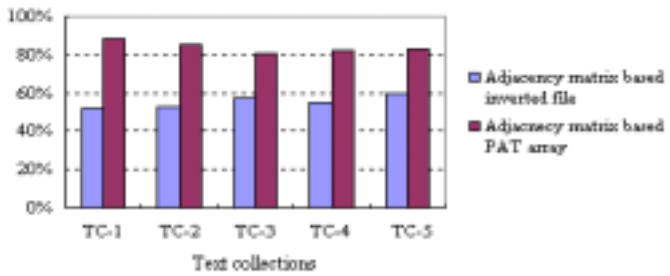

Fig.3 Experimental results of query processing efficiency improvement ratio

6 Conclusions

In this paper, we proposed, investigated and implemented two new full-text indexing models that can improve the query efficiency of IR systems considerably at the cost of much less amount of extra storage overhead compared to the size of original text database. Experiments over real world Chinese text collections were carried, which validated the effectiveness and efficiency of the new models. Further research will focus on exploring efficient building and updating algorithms as well as query optimization approaches for the new indexing models over large-scale text databases.

References:

[1] Baesa-Yates, R., Ribeiro-Neto, B. Modern Information Retrieval. Reading, MA: Addison Wesley, 1999.

[2] Sullivan, D. Search Engine Watch. http://www.searchenginewatch.com.

[3] AltaVista, http://www.altavista.com.

[4] Zhou, Shui-geng. Key techniques of Chinese text databases [Ph.D. Thesis]. Shanghai: Fudan University, 2000 (in Chinese).

[5] Tomasic, A., Garcia-Molina, H., Shoens, K. Incremental updates of inverted lists for text document retrieval. In: Snodgrass, R.T., Winslett, M., eds. Proceedings of the SIGMOD'94. New York: ACM Press, 1994. 289~300.

[6] Ribeiro-Neto, B.A., Silva de Moura, E., Neubert, M.S., Ziviani, N. Efficient distributed algorithms to build inverted files. In: Hearst, M., Tong, R., eds. Proceedings of the SIGIR'99. New York: ACM Press, 1999. 105~112

[7] Faloutsos, C. Signature-Based text retrieval methods: a survey. Data Engineering Bulletin, 1990,13(1):25~32.

[8] Manber, U., Myers, E. Suffix arrays: a new method for on-line string searches. SIAM Journal of Computing, 1993,22(5):935~948.

[9] Chavez, E, Navarro, G., *et al*. Searching in metric spaces. ACM Computing Surveys, 2001,33(3):273~321.

附中文参考文献：

[4] 周水庚.中文文本数据库若干关键技术研究[博士学位论文].上海:复旦大学,2000.

基于邻接矩阵的全文索引模型

周水庚[1]，胡运发[2]，关佶红[3]

[1](复旦大学 计算机科学与工程系,上海 200433);

[2](复旦大学 计算机与信息技术系,上海 200433);

[2](武汉大学 计算机学院,湖北 武汉 430079)

摘要: 文本信息的急剧增加和越来越多的用户通过在线方式获取文本信息,使得查询效率成为信息检索系统一个突出瓶颈.提出两种新型全文索引模型,用于改善信息检索系统的查询效率.通过使用有向图表示文本串,引出关于文本串的邻接矩阵;采用两种不同的方式实现文本串邻接矩阵,导出了两种基于邻接矩阵的新型全文索引模型,即基于邻接矩阵的倒排文件和基于邻接矩阵的 PAT 数组.给出了基于新模型的文本查询算法;分析了新模型的存储空间和查询时间的开销,并分别与两种传统索引模型进行了比较.对实际文本库进行了测试以证实新模型的效能.新模型能够以相对于原文较小的空间代价获得较大幅度的查询效率的提高,因此适合于在大规模文本检索系统中应用.

关键词: 信息检索;全文索引;倒排文件;PAT 数组;邻接矩阵;模型

中图法分类号: TP311 文献标识码: A