PERGAMON

Expert Systems with Applications 23 (2002) 95–102

Expert Systems with Applications

www.elsevier.com/locate/eswa

A parallelized indexing method for large-scale case-based reasoning

Wei-Chou Chen[a], Shian-Shyong Tseng[a,*], Lu-Ping Chang[a], Tzung-Pei Hong[b], Mon-Fong Jiang[a]

[a]*Department of Computer and Information Science, National Chiao Tung University, Hsinchu 300, Taiwan, ROC*
[b]*Department of Electrical Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan, ROC*

Abstract

Case-based reasoning (CBR) is a problem-solving methodology commonly seen in artificial intelligence. It can correctly take advantage of the situations and methods in former cases to find out suitable solutions for new problems. CBR must accurately retrieve similar prior cases for getting a good performance. In the past, many researchers proposed useful technologies to handle this problem. However, the performance of retrieving similar cases may be greatly influenced by the number of cases. In this paper, the performance issue of large-scale CBR is discussed and a parallelized indexing architecture is then proposed for efficiently retrieving similar cases in large-scale CBR. Several algorithms for implementing the proposed architecture are also described. Some experiments are made and the results show the efficiency of the proposed method. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Case-based reasoning; Parallelized indexing; Bitwise indexing; Case retrieval; Performance

1. Introduction

Case-based reasoning (CBR) is a problem-solving methodology commonly seen in AI (Waston, 1999). Just like human reasoning, CBR uses prior cases to find out suitable solutions for new problems. It can take advantage of the situations and methods in former cases to handle unexpected new situations. Additionally, CBR can achieve human learning behaviors by constantly adding cases, thus raising the accuracy of problem solutions.

CBR has been successfully applied to the areas of planning (Gardingen & Watson, 1999; Suh, Jhee, Ko, & Lee, 1998), diagnosis (Li, 1999), law (Cercone, An, & Chan, 1999; Daengdej, Lukpse, Tsui, Beinat, & Prophet, 1997) and decision making (Dutta, Wierenga, & Dalebout, 1997), among others. The major tasks of CBR can be divided into five phases, including *Case Representation*, *Indexing*, *Matching*, *Adaptation* and *Storage*. CBR must accurately retrieve similar prior cases for getting a good performance. Many researchers have proposed useful technologies to handle this problem (Cercone et al., 1999; Gupta & Montazemi, 1997; Shin & Han, 1999). However, the performance of retrieving similar cases in large-scale CBR has seldom been discussed. When the number of cases in a case base becomes large, the processing time for retrieving similar cases rapidly increases. The process of retrieving similar cases thus becomes a critical task in CBR. We proposed an indexing method to improve the performance of indexing and retrieving in data warehousing (Chan, Tseng, Chang, & Jiang, 2000). In this paper, we propose a novel parallelized indexing method with a suitable similarity-measuring function for CBR. Several corresponding algorithms have also been described to accelerate the performance of case indexing and retrieving. Finally, experiments on CBR with two and four processors have been made, with the results showing the scalability and efficiency of the proposed method.

2. Review of case-based reasoning

As mentioned earlier, the major tasks of CBR can be divided into five phases. When a new problem arrives, the situation of this problem is identified in the Case Representation phase. After that, the important features of the new case are extracted as its indexes in the Indexing phase. These indexes are then passed to the Matching phase for retrieving similar cases in the case base. The Adaptation phase then adapts the solutions of similar cases by adaptation rules to fit the new problem. After the final solution of the new case is confirmed by users, it is stored in the case base via the Storage phase.

The success of a CBR system mainly depends on effective and efficient retrieval of similar cases for a new problem. Indexing and matching are thus very important to CBR (Shin & Han, 1999). Indexing usually uses some

* Corresponding author. Tel.: +886-3-5731966; fax: +886-3-5721490.
E-mail addresses: sstseng@cis.nctu.edu.tw (S.-S. Tseng), tphong@nuk.edu.tw (T.-P. Hong).

features of cases for identification and matching uses a pre-defined matching function for case retrieval. Each feature is given a weight to represent its importance. Based on weighted sums of features matched, similar cases in a case base can then be retrieved (Cercone et al., 1999; Gupta & Montazemi, 1997; Shin & Han, 1999).

Several useful approaches have been proposed to retrieve similar cases. Two methods for assigning the weights of features were proposed by Cercone et al. (1999) and Shin and Han (1999). Gupta discussed that the weights of features were different between a new case and prior cases (Gupta & Montazemi, 1997). The performance of retrieving similar cases has, however, seldom been discussed. Retrieving similar cases needs much computation time when a matching function becomes complex or when the number of cases in a case base grows large. Retrieving similar cases efficiently thus becomes an important issue in large-scale CBR.

3. Architecture of bitwise indexing CBR

An indexing method, called bitwise indexing, is proposed here to speed up retrieving similar cases in CBR. The architecture of the bitwise indexing CBR (BWI-CBR) is shown in Fig. 1. It is the same as general CBR except for the following aspects:

1. The proposed bitwise indexing method serves as a new indexing method in CBR. It can speed up retrieving similar cases in the Matching phase.
2. The matching phase is divided into two sub-phases to reduce the matching time. In the first sub-phase, all irrelevant cases are filtered out to avoid calculation of their similarities by *Mask Vectors*. In the second sub-phase, the similarities between a new case and relevant cases are calculated.
3. The cost of computing similarities between a new case and all relevant prior cases is usually high since the amount of cases is large in a large-scale case base. We can then pre-compute all possible similarities and construct the *Similarity Mapping List* to speed up the calculation of similarities. Accordingly, the similarity of a new case and each prior case can be quickly found out by seeking the Similarity Mapping List. The computational overhead can thus be largely reduced.

In our approach, the bitwise indexing method is first used to replace the traditional indexing method in CBR. The bitwise operations are then used to select relevant prior cases in the matching phase. In this way, irrelevant cases can be filtered out quickly. The number of prior cases needing similarity calculation can thus be reduced. Therefore, the similarities between a new case and relevant prior cases can be quickly measured in the similarity measurement phase.

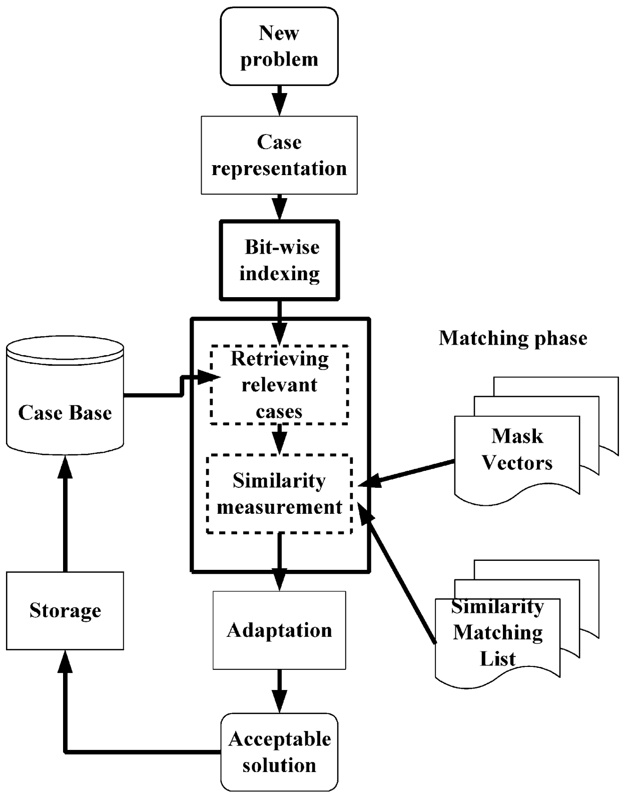

Fig. 1. The architecture of BWI-CBR.

4. Notation and definitions

Assume a set of cases C is stored in a CBR system for a specific domain, denoted DOM. The ith case in C is represented by C_i. Also assume all the cases in C can be abstracted by a set of attributes A, denoted $A = \langle A_1, A_2, \ldots, A_r \rangle$, where r is the number of attributes. The value of an attribute A_k for a case C_j is denoted $V_k(j)$, which cannot be null. The attribute values of a case C_j can then be represented as $V(j) = \langle V_1(j), V_2(j), \ldots, V_r(j) \rangle$. The set of possible values for attribute A_i, called *attribute value domain*, is denoted $V_i = \langle V_{i1}, V_{i2}, \ldots, V_{i\alpha(i)} \rangle$, where $\alpha(i)$ is the number of values for A_i, and V_{ij} is the jth possible attribute value of A_i.

In a CBR system, a set of prior cases is stored in a case base for solving a new problem. A matching function is used to evaluate cases based on a weighted sum of matched attributes with a new case. Attribute value can thus be used for indexing a case. An index of a case can be formally defined as follows:

Definition 1 (Case Index). The index IND_k of a case C_k in a CBR system for domain DOM is defined as

$$\mathrm{IND}_k = \{A_1 = V_1(k), A_2 = V_2(k), \ldots, A_r = V_r(k)\}.$$

Table 1
Five cases in Example 1

	OS	PL	DB
Case 1	WinNT	C	SQL-Server
Case 2	OS2	Basic	ORACLE
Case 3	Linux	Java	SYBASE
Case 4	Mac	Java	ORACLE
Case 5	Solaris	Pascal	SQL-Server

A case in CBR can be formally defined as follows:

Definition 2 (Case). A case C_k in a CBR system for domain DOM is a pair (IND_k, cv_k), where cv_k is the actual contents of case C_k and $C_k \in C$.

A bitwise indexing vector used in the proposed indexing method is defined as follows:

Definition 3 (Bitwise indexing vector of an attribute). The bitwise indexing vector B_i of the ith attribute for case C_k is a bit string $B_i = b_{i1}b_{i2}, \ldots, b_{i\alpha(i)}$, where $b_{ij} = 1$ if $V_i(k) = V_{ij}$ and $b_{ij} = 0$ otherwise.

Definition 4 (Bitwise indexing vector of a case). The bitwise indexing vector BWI_k of a case C_k is the concatenation of the bitwise indexing vectors of all the attributes for case C_k. That is, $\mathrm{BWI}_k = B_1B_2\ldots B_r$, where r is the number of attributes.

Definition 5 (Matrix of bitwise indexes for case-based reasoning). A matrix T_{BWI} of bitwise indexes for CBR is represented as

$$\begin{bmatrix} \mathrm{BWI}_1 \\ \mathrm{BWI}_2 \\ \vdots \\ \mathrm{BWI}_{|C|} \end{bmatrix}$$

where $|C|$ is the number of cases.

Example 1. Assume that a CBR system containing five cases is shown in Table 1.

The bitwise indexes for the above cases are shown in Table 2.

Table 2
The bitwise indexes of the cases in Table 1

BWI_1	10000	1000	100
BWI_2	01000	0100	010
BWI_3	00100	0010	001
BWI_4	00010	0010	010
BWI_5	00001	0001	100

5. Indexing phase in bitwise indexing CBR

The bitwise indexes for prior cases are generated by the following two algorithms:

Algorithm 5.1 (Bitwise index creation algorithm).

```
Input: A case $C_i$.
Output: A bitwise index $\mathrm{BWI}_i$ of $C_i$.
Step 1: Create a bitwise vector of length $r$, where $r$ is the number of attributes.
Step 2: For each bit $b_{jk}$ in the vector, set $b_{jk} = 1$ if $V_j(i) = V_{jk}$; set $b_{jk} = 0$ otherwise.
Step 3: Return the vector $\mathrm{BWI}_i$.
```

Algorithm 5.2 (Bitwise index matrix creation algorithm).

```
Input: A set of cases.
Output: A bitwise index matrix $T_{\mathrm{BWI}}$ of the cases.
Step 1: Create an empty matrix $T_{\mathrm{BWI}}$.
Step 2: Repeat the following sub-steps for each case $C_i$ until all cases are processed.
    Sub-step 2.1: Use the bitwise index creation algorithm to get the index $\mathrm{BWI}_i$ of $C_i$.
    Sub-step 2.2: Add $\mathrm{BWI}_i$ into $T_{\mathrm{BWI}}$.
Step 3: Return $T_{\mathrm{BWI}}$.
```

After a bitwise index matrix is built, bitwise operations can easily be used to retrieve similar cases in CBR.

6. Matching phase in bitwise indexing CBR

Calculating the similarities between a new case and prior cases is a time-consuming task. A two-phase matching approach, called the *Similar-cases-seeking algorithm*, is thus proposed here to reduce the matching time. It includes the *relevant-cases-retrieving* phase and the *similarity-computing* phase. In the first phase, all irrelevant cases are filtered out to avoid calculation of their similarities. The time of calculating the similarities of useful prior cases can then be decreased. The similarities of the new case with remaining prior cases are then computed efficiently in the similarity-computing phase. The algorithm is described as follows:

Algorithm 6.1 (Similar-cases-seeking algorithm).

```
Input: A bitwise index matrix $T_{\mathrm{BWI}}$ and a new case $C_N$.
Output: A set of similar cases $Rc$ with their similarity degrees with $C_N$.
Step 1: Use the bitwise index creation algorithm to get the index $\mathrm{BWI}_N$ of the new case $C_N$.
Step 2: Initialize the counter $j$ to 1 and $Rc$ to an empty set.
Step 3: For each $\mathrm{BWI}_j$ in $T_{\mathrm{BWI}}$, do the following sub-steps ($1 < j \leq |C|$):
    Sub-step 3.1: Call the *search-relevant-cases algorithm* (described later) to compute the relevance degree $rdi_j$ between $\mathrm{BWI}_N$ and $\mathrm{BWI}_j$.
    Sub-step 3.2: If $rdi_j = 0$, ignore the case $C_j$ and go to Sub-step 3.5.
    Sub-step 3.3: Call the similarity-computing algorithm (described later) to compute the similarity $sim_j$ between $C_N$ and $C_j$.
    Sub-step 3.4: Add case $C_j$ with its similarity $sim_j$ to $Rc$.
    Sub-step 3.5: Add 1 to $j$.
Step 4: Sort the cases in $Rc$ in descending order of their similarities.
Step 5: Output $Rc$.
```

6.1. Retrieving relevant cases

A prior case is relevant to a new case if they have at least one same attribute value. These two cases are then similar in a certain degree. The bits in the corresponding positions of the matched attributes should be set as '1' in their bit vectors. This can easily be found by using the 'AND' bitwise operation to compare the two bit vectors. The following *Search-relevant-cases algorithm* is thus proposed to achieve this purpose.

Algorithm 6.2 (Search-relevant-cases algorithm).

```
Input: The bitwise indexing vector $\mathrm{BWI}_N$ of a new case $C_N$ and the index $\mathrm{BWI}_j$ of a prior case $C_j$ in $C$.
Output: The relevant degree $rdi_j$ between $C_N$ and $C_j$.
Step 1: Use the AND bitwise operation on $\mathrm{BWI}_N$ and $\mathrm{BWI}_j$ and store the result as $rdi_j$, which is also a bit string.
Step 2: Return $rdi_j$.
```

Since the AND bitwise operation is fast, the Search-relevant-cases algorithm selects relevant prior cases quickly. If *rdi* is zero, then the prior case is thought of as irrelevant and will be filtered out.

6.2. Computing similarities

After all relevant prior cases have been retrieved, the similarities between the new case and them are computed. As mentioned earlier, a matching function based on a weighted sum of matched attributes is defined to calculate the similarity degrees. Each attribute has its own weight. Since a case has only one value for an attribute, at most one bit in the bit string *rdi* is set for each attribute after the Search-relevant-cases algorithm is executed. Accordingly, a special bitwise vector, called the *Mask Vector*, is proposed to help compute similarities. Let $\langle 1 \rangle$ be the string of length α with all bits being 1 and $\langle 0 \rangle$ be the string of length α with all bits being 0. The definition of the Mask Vector is shown later.

Definition 6 (Mask Vector). A bitwise indexing mask vector *Mask* is a set of $Mask_k$, where $0 < k \leq r$ and r is the number of attributes. Each $Mask_k$, denoting the mask vector of attribute A_k, is a concatenation of r bit strings as $Mask_k = S_1 S_2 \ldots S_r$, where $S_i = \langle 1 \rangle$ for $i = k$ and $S_i = \langle 0 \rangle$ for $i \neq k$.

By applying the AND operation on $Mask_k$ and the bitwise vectors *rdi*s generated from the search-relevant-cases algorithm, the similarities between a new case and prior cases for attribute A_k can easily be found by the following similarity-measuring function:

$$SIM(Case_i) = \frac{\sum_{j=1}^{r} (\mathrm{PC}_{ij} W_j)}{\sum_{j=1}^{r} W_j},$$

where $SIM(Case_i)$ is the similarity between the ith prior case and the new case, W_j is the weight of the jth attribute, $\mathrm{PC}_{ij} = 0$ if the result of performing the AND bitwise operation on rdi_i and $Mask_j$ is 0, and $\mathrm{PC}_{ij} = 1$ otherwise.

Several prior cases may have the same similarity with a new case as long as they have the same attributes matched. This is especially common when the numbers of possible values for attributes are large. For this situation, the cost for calculating similarities of prior relevant cases can be reduced if all possible similarities are pre-computed and stored into the Similarity Mapping List. Each element in the Similarity Mapping List is a similarity value for some attributes matched. Thus, the similarity of a prior case with a new case for known attributes matched can easily be found from the list, instead of calculating by the earlier formula. The Similarity Mapping List is formally defined as follows:

Definition 7 (Similarity Mapping List). Let L be a Similarity Mapping List and L_i be an element in L with an index value i, which is determined from the attributes matched, $1 \leq i \leq 2^{|r|} - 1$. Let i be represented as a binary code $b_{i1} b_{i2} \ldots b_{ir}$, with $b_{ij} = 1$ if the jth attribute is matched and $b_{ij} = 0$ otherwise, $1 \leq j \leq r$. Thus the value of L_i is thus

$$\frac{\sum_{j=1}^{r} b_{ij} W_j}{\sum_{j=1}^{r} W_j}.$$

Algorithm 6.3 (Similarity mapping list creation algorithm).

```
Input: Weights of attributes $W_1, W_2, \ldots, W_r$ of CBR.
```

```
Output: A similarity mapping list $L$.
Step 1: Initialize the counter $i$ to 1 and the list $L$ to be empty.
Step 2: For each $i$, $1 \leq i \leq 2^{|r|} - 1$, do the following sub-steps:
    Sub-step 2.1: Encode $i$ into a binary string $\langle b_{i1} b_{i2} \ldots b_{ir} \rangle$.
    Sub-step 2.2: Calculate the similarity degree $L_i$ by the formula in Definition 7.
    Sub-step 2.3: Put $L_i$ into the list $L$ with index $i$.
Step 3: Return $L$.
```

After the Similarity Mapping List has been built, the similarity of each prior case and a new case can be quickly found by the following algorithm:

Algorithm 6.4 (Similarity-computing algorithm).

```
Input: The relevant degree $rdi_j$ of case $C_j$ with a new case, the Mask Vector, and the Similarity Mapping List $L$.
Output: The similarity of $C_j$ with a new case.
Step 1: Initialize a zero binary string of length $r$.
Step 2: For each $i$, $1 \leq i \leq r$, set the $i$th position in the string to 1 if the result of using the AND bitwise operation on Mask$_i$ and $rdi_j$ is not all 0.
Step 3: Transform the binary string into an integer $j$.
Step 4: Get $L_j$ from the Similarity Mapping List.
Step 5: Return $L_j$.
```

Since the Similarity Mapping List and the Mask Vector are constructed in the pre-processing step, and since only the AND bitwise operations are executed on Mask Vectors and bitwise vectors of relevant cases in the Similarity-computing algorithm, the computational time for finding the similarities can thus be significantly reduced.

Example 2. Continuing from Example 1, the BWI$_N$ of a new case C_N, which is {OS = Solaris, PL = Java, DB = ORACLE}, is {00001 0010 010}. Each BWI$_j$ in T_{BWI} in Table 2 is processed as follows:

- For BWI$_1$: The relevant degree rdi_1 between BWI$_1$ and BWI$_N$ is found as {00000 0000 000} by the Search-relevant-cases algorithm. Since all the bits in rdi_1 are '0', Case 1 is filtered out.
- For BWI$_2$: $rdi_2 = \{00000\ 0000\ 010\}$. Since one of the bits in rdi_2 is '1', the similarity-computing algorithm is invoked to find the similarly sim_2 of Case 2 with the new case as 0.333. Case 2 is then a relevant case.
- For BWI$_3$: $rdi_3 = \{00000\ 0010\ 000\}$. Since one of the bits in rdi_3 is '1', Case 3 is a relevant case. Its similarity is found as 0.333.
- For BWI$_4$: $rdi_4 = \{00000\ 0010\ 010\}$. Since more than one bit in rdi_4 are '1', Case 4 is a relevant case. Its similarity is found as 0.667.
- For BWI$_5$: $rdi_5 = \{00001\ 0000\ 000\}$. Since one of the bits in rdi_5 is '1', Case 5 is a relevant case. Its similarity is found as 0.333.

Table 3
Four relevant cases and their similarities

Relevant case	Case 4	Case 2	Case 3	Case 5
Similarity	0.667	0.333	0.333	0.333

After the relevant cases are sorted in decreasing order of similarities, the results are shown is Table 3.

6.3. Discussion of filtering and storage

As mentioned earlier, the proposed matching algorithms include two-phases to reduce the computational time. At the *retrieving-relevant-cases* phase, irrelevant prior cases are filtered out. Thus, only the similarities between relevant prior cases and the new case are computed at the similarity-computing phase. Assume that the number of cases in the case base is N and the average filtering percentage is M. The time needed to retrieve relevant prior cases and to calculate their similarities in STEP 3 of Algorithm 6.1 is analyzed as

$$\mathrm{Time}_{\text{with filtering}} \approx (Nt_{\text{and}} + NM(rt_{\text{and}}) + NMt_{\text{c}})$$

$$= N\left(M\frac{1}{M}t_{\text{and}} + M(rt_{\text{and}}) + Mt_{\text{c}}\right)$$

$$= NM\left(\left(\frac{1}{M} + r\right)t_{\text{and}} + t_{\text{c}}\right),$$

where t_{and} is the time needed for an AND bitwise operation and t_c is the seek time in the Similarity Mapping List. If no filtering is performed, the time needed to calculate their similarities in STEP 3 of Algorithm 6.1 is analyzed as

$$\mathrm{Time}_{\text{without filtering}} \approx (Nt_{\text{and}} + N(rt_{\text{and}}) + Nt_{\text{c}})$$

$$= N((1 + r)t_{\text{and}} + t_{\text{c}}).$$

The performance due to the filtering is then

$$\frac{\mathrm{Time}_{\text{with filtering}}}{\mathrm{Time}_{\text{without filtering}}} \approx \frac{NM\left(\left(\frac{1}{M} + r\right)t_{\text{and}} + t_{\text{c}}\right)}{N((1 + r)t_{\text{and}} + t_{\text{c}})} \approx M.$$

The proposed method can indeed improve the performance of CBR although some extra storage spaces are required. These storage spaces are used for storing the bitwise indexes and the Similarity Mapping List. The sizes of extra storage spaces required in our method are analyzed as follows:

- The storage space required for the bitwise indexes $T_{\mathrm{BWI}} = |C| \sum_{i=1}^{r} \alpha(i)$, where $\alpha(i)$ is the number of bits used for attribute A_{is}, r is the number of attributes, and $|C|$ is the number of cases in CBR. For example, assume that there are 100 000 cases in a case base and 16 attributes to describe each case. Also assume each attribute has four possible values. The storage space required for $T_{BWI} = ((100\,000)\sum_{i=1}^{16} 4)$ *bits* $= (6\,400\,000/8)$ *bytes* $=$ 800 000 bytes $\cong$ 0.8MB.

- The storage space of the Mask Vector $= r\sum_{i=1}^{r} \alpha(i)$. For the earlier example, the storage space required for the Mask Vector $= (16 \times \sum_{i=1}^{16} 4)$ bits $= (1024/8)$ bytes $= 128$ bytes.
- The storage space required for the Similarity Mapping List $L = f(2^r - 1)$, where f is the storage space required for storing a similarity value. Assume that f is a four-byte real number. For the earlier example, the storage space required for the Similarity Mapping List *List* $L = 4 \times (2^{16} - 1)$ bytes $= 262\,140$ bytes $\cong$ 256KB.

Note that the size of the extra storage space required for the Similarity Mapping List is exponential to r. Therefore, the Similarity Mapping List is not suitable for domains with large numbers of attributes.

7. Parallelized bitwise indexing CBR

As described earlier, similarity calculations between all prior cases and a new case are independent. They can thus be executed in parallel to speed up the performance of CBR. A parallel bitwise indexing method is proposed to achieve this purpose.

7.1. The architecture of parallel bitwise indexing CBR

The bitwise indexing method of CBR treats each case as a bit vector and calculates the similarity between each prior case and a new case independently. Therefore, our parallel BWI-CBR indexing method can directly use a subset of cases as a vector matrix for a processor. The architecture of the parallel BWI-CBR Indexing method is shown in Fig. 2.

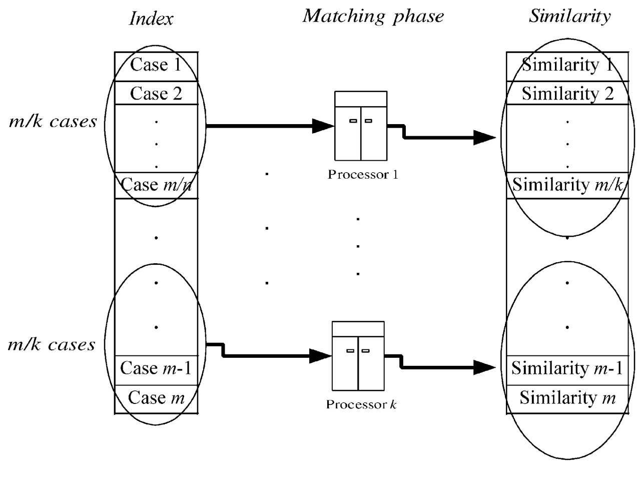

Fig. 2. The architecture of parallel BWI-CBR.

There are m cases in a case base and k processors to be used for matching. Each processor thus handles about m/k cases.

7.2. Indexing phase in parallel bitwise indexing CBR

The indexes (BWIs) for the cases are equally distributed in the processors given. Each processor thus has sub-matrix of indexes to process. The Bitwise-index-sub-matrix creation algorithm is described later:

Algorithm 7.1 (Bitwise-index-sub-matrix creation algorithm).

```
Input: The bitwise indexing matrix $T_{\mathrm{BWI}}$ of the case base and the number $k$ of processors.
Output: The bitwise indexing sub-matrix $\mathrm{ST}^i_{\mathrm{BWI}}$ for each processor $i$.
Step 1: Set $l = \lceil |C/k| \rceil$ and $h$ is the remainder of $|C|/k$, where $|C|$ is the number of cases.
Step 2: For each processor $i$, do the following sub-steps:
  Sub-step 2.1: Create an empty sub-matrix $\mathrm{ST}^i_{\mathrm{BWI}}$.
  Sub-step 2.2: Copy the indexes from $\mathrm{BWI}_{1-l(i-1)}$ to $\mathrm{BWI}_{li}$ in $T_{\mathrm{BWI}}$ to $\mathrm{ST}^i_{\mathrm{BWI}}$ if $i \le h$; copy the indexes from $\mathrm{BWI}_{1+lh+(l-1)(i-1-h)}$ to $\mathrm{BWI}_{lh+(l-1)(i-h)}$ in $T_{\mathrm{BWI}}$ to $\mathrm{ST}^i_{\mathrm{BWI}}$ if $i > h$.
Step 3: Return the set of $\mathrm{ST}^i_{\mathrm{BWI}}$, $0 < i \le k$.
```

Example 3. Assume there are two processors to handle the task in Example 2. The bitwise indexing sub-matrix $\mathrm{ST}^i_{\mathrm{BWI}}$ for each processor i is shown in Table 4.

7.3. Matching phase in parallel bitwise indexing CBR

In this section, the proposed *parallel similar-cases-seeking algorithm* is stated later.

Algorithm 7.2 (Parallel similar-cases-seeking algorithm).

```
Input: The set of $\mathrm{ST}_{\mathrm{BWI}}$s for $k$ processors and a new case $C_N$.
Output: A set of similar cases $Rc$ with their similarity degrees.
Step 1: Execute the similar-cases-seeking algorithm for each processor $i$ on $\mathrm{ST}^i_{\mathrm{BWI}}$ to get the similar case set $Rc_i$ with similarities.
Step 2: Merge all $Rc_i$s into $Rc$ in decreasing order of similarities.
Step 3: Output $Rc$ with their similarities.
```

Example 4. Continuing from Example 3, the relevant cases in both processors are shown in Table 5.

These relevant cases are then merged and sorted in decreasing order of similarities, with results shown in Table 6.

Table 4
The bitwise indexing sub-matrix in each processor

ST^1_{BWI}	Case 1	10000	1000	100
	Case 2	01000	0100	010
	Case 3	00100	0010	001
ST^2_{BWI}	Case 4	00010	0010	010
	Case 5	00001	0001	100

Table 5
The relevant cases and their similarities in both processors

Rc_1	Case 2	0.333
	Case 3	0.333
Rc_2	Case 4	0.667
	Case 5	0.333

Table 6
The merged relevant cases and their similarities

Relevant case	Case 4	Case 2	Case 3	Case 5
Similarity	0.667	0.333	0.333	0.333

8. Experiments and discussions

In this section, the performance of the BWI-CBR and the two-processor parallel BWI-CBR is compared. A Pentium-166 dual-processor system, running the Microsoft Windows NT multithreaded OS, is used for the parallel BWI-CBR. The system includes 512K L2 cache and 128MB shared-memory. The experimental results along with different numbers of cases are show in Fig. 3. It can easily be seen that the speed-up increases along with the increase of cases, finally converting to 1.6.

Next, the performance of the BWI-CBR and the four--processor parallel BWI-CBR is compared. A Pentium-Pro 200 quadruple-processor system, running the Microsoft Windows NT multithreaded OS, is used for the parallel BWI-CBR. The system includes 1M L2 cache and 512MB shared-memory. The experimental results along with different numbers of cases are shown in Fig. 4. It can easily be seen that the speed-up increases along with the increase of cases, finally converting to 3.5.

It is obvious that BWI-CBR is quite suitable for parallelization since the proposed bitwise indexing matrix can easily be separated into several independent, nearly equal-sized sub-matrixes. When the BWI-CBR is implemented in a multiple CPU machine, the workloads can be easily shared on the processors. The workloads of all processors are thus almost balanced.

9. Conclusion and future work

In addition to accuracy, performance should also be taken

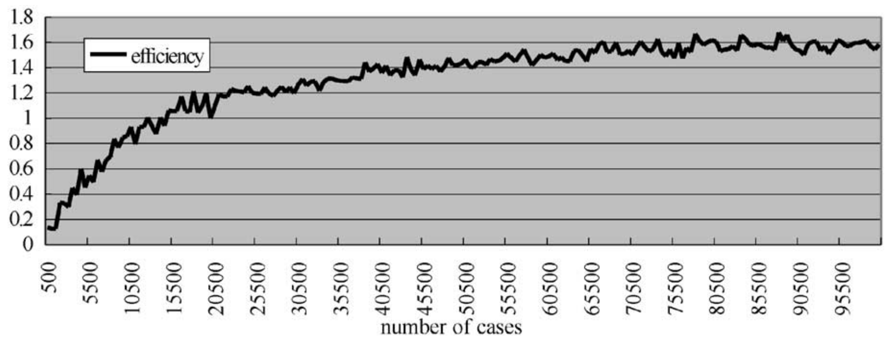

Fig. 3. Speed-up of parallel BWI-CBR on two processors.

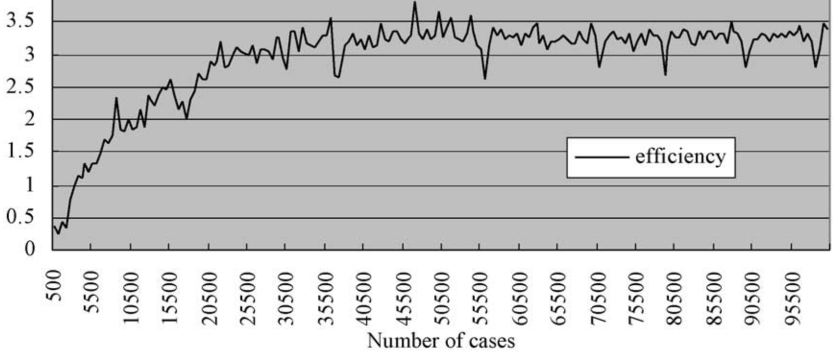

Fig. 4. Speed-up of parallel BWI-CBR on four processors.

into consideration in CBR, especially when the number of prior cases is large. In this paper, the performance issue of large-scale CBR is discussed and a new parallelized method based on bitwise indexing has been proposed. Several corresponding algorithms, including the index creation and the case retrieval algorithms, are described. Experiments have also been made for comparing the performance on different numbers of processors with the results showing the proposed parallel method is quite efficient. In the future, we will attempt to modify and apply the indexing method and the corresponding retrieving algorithms to CBR in the data warehousing. Also, since the queries of OLAP/OLTP in a data warehouse may be complex and time-consuming, we will also attempt to design a good indexing strategy that can reduce the query time and the computational overhead.

Acknowledgements

This research was partially supported by the National Science Council of the Republic of China under contract NSC90-2213-E-009-132.

References

Cercone, N., An, A., & Chan, C. (1999). Rule-induction and case-based reasoning: hybrid architectures appear advantageous. *IEEE Transactions on Knowledge and Data Engineering*, *11* (1), 166–174.

Chen, W. C., Tseng, S. S., Chang, L. P., & Jiang, M. F. (2000). A similarity indexing method for the data warehousing—bitwise indexing method. *The fifth Pacific-Asia conference on knowledge discovery and data mining* (pp. 525–537).

Daengdej, J., Lukpse, D., Tsui, E., Beinat, P., & Prophet, L. (1997). Combining case-based reasoning and statistical method for proposing solution in RICAD. *Knowledge Based Systems*, *10*, 153–159.

Dutta, S., Wierenga, B., & Dalebout, A. (1997). Case-based reasoning systems: from automation to decision-aiding and stimulation. *IEEE Transactions on Knowledge and Data Engineering*, *9* (6), 911–922.

Gardingen, D., & Watson, I. (1999). A web based CBR system for heating ventilation and air conditioning systems sales support. *Knowledge-Based Systems*, *12*, 207–214.

Gupta, K. M., & Montazemi, A. R. (1997). Empirical evaluation of retrieval in case-based reasoning systems using modified cosine matching function. *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans*, *27* (5), 601–612.

Li, L. L. X. (1999). Knowledge-based problem solving: an approach to health assessment. *Expert Systems with Applications*, *16*, 33–42.

Shin, K. S., & Han, I. (1999). Case-based reasoning supported by genetic algorithms for corporate bond rating. *Expert Systems with Applications*, *16*, 85–95.

Suh, M. S., Jhee, W. C., Ko, Y. K., & Lee, A. (1998). A case-based expert system approach for quality design. *Expert Systems with Applications*, *15*, 181–190.

Waston, I. (1999). Case-based reasoning is a methodology not a technology. *Knowledge Based Systems*, *12*, 303–308.