A Parallelized Indexing Method for Large-scale Case-based Reasoning

Wei-Chou Chen, Shian-Shyong Tseng, Lu-Ping Chang, Tzung-Pei Hong and Mon-Fong Jiang

Department of Computer and Information Science
National Chiao Tung University
Hsinchu 300, Taiwan, R. O. C.
E-mail: sstseng@cis.nctu.edu.tw

Department of Electrical Engineering
National University of Kaohsiung
Kaohsiung 811, Taiwan, R. O. C
E-mail: tphong@nuk.edu.tw

Abstract

Case-based reasoning (CBR) is a commonly seen problem-solving methodology 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 proposed method.

Keywords: case-based reasoning, parallelized indexing, bit-wise indexing, case retrieving, performance.

1. Introduction

Case-based reasoning (CBR) is a commonly seen methodology of problem-solving in AI [13]. Just like human reasoning, CBR uses prior cases to find out suitable solutions for new problems. Other problem-solving methodologies of in AI must find out the general relationship between problem situations and problem-solving methods, to construct suitable solutions [11]. Unlike the others, CBR pays much attention to the characteristics of each prior case. It can correctly take advantage of the situations and methods in former cases to can handle unexpected 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 [6][12], diagnosis [9], law [1] and decision making [4][5] among others. It uses useful prior cases to solve the new problems. 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 [1][7][11]. However, performance of retrieving similar cases in

*Corresponding Author
large-scale CBR was seldom been discussed. When the number of cases in the case base becomes large, the processing time for retrieving similar cases rapidly increases. The process of retrieving similar cases thus becomes a critical task of CBR. In [2], we had proposed an indexing method to improve the performance of indexing and retrieving in the data warehousing. In this paper, we propose a novel parallelized indexing method with a suitable similarity-measuring function. Several corresponding algorithms have also been described to accelerate the performance of case indexing and retrieving. Finally, experiments on comparison 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

The major tasks of CBR can be divided into five phases, including Case Representation, Indexing, Matching, Adaptation and Storage. When a new problem arrives, the situation of this problem is identified in the by 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 according to the similarity of the indexes. The Adaptation phase then adapts the solutions of similar cases by some adaptation rules to fit the new problem. After the 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 an effective retrieval on similar cases for a new problem. Indexing and matching are thus very important in CBR [11]. Indexing usually uses some features of cases to identify them, and matching uses a predefined matching function to retrieve cases. Each feature is given a weight to represent its importance. Based on weighted sums of features matched, similar cases in the case base can then be retrieved [1][7][11].

Several useful approaches to retrieve similar cases accurately have been proposed. Two method to assign the weights of features were proposed in [1][11]. Gupta [7] discussed that the weights of features were different between new case and prior cases. 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. Retrieve similar cases efficiently thus becomes an important issue in large-scale CBR.

3. Architecture of Bit-Wise Indexing CBR

A novel indexing method, called bit-wise indexing, is proposed here to speed up retrieving similar cases in CBR. The architecture of bit-wise Indexing CBR (BWI-CBR) is shown in Fig. 1. BWI-CBR is the same as general CBR except for the following:

1) Bit-wise indexing phase: The proposed bit-wise indexing method serves as a new indexing method in CBR. It can highly speed up retrieving similar cases in the Matching phase.

2) Matching phase:
   2.1) Retrieving-relevant-cases sub-phase: Relevant prior cases are selected and irrelevant prior cases are filtered out. Moreover, the matching results can be used to calculate the similarities of a new case with prior cases in the following similarity-measurement sub-phase.
   2.2) Similarity-measurement sub-phase: This phase computes the similarities between a new case and relevant cases in case base.

Moreover, The cost of computing similarities between a new case and all relevant prior
cases is usually high since the amount of case is large in a large-scale case base. To solve the problem, we use the Mask Vector. We can pre-compute all-possible similarities and construct the Similarity Mapping List. Accordingly, the similarity of each prior case and new arrival case can be quickly found out by seeking the Similarity Mapping List. The computing overhead can thus be largely reduced.

In our approach, the bit-wise indexing method is first used to replace the traditional indexing method in CBR. The bit-wise operations are then used to select relevant prior cases in the phase for retrieving relevant cases. By this way, irrelevant cases can be filtered out quickly. The number of prior cases which need to compute their similarities with the new case can thus be reduced. Therefore, the similarities between relevant prior cases and a new case can be quickly measured in the similarity measurement phase.

![Fig. 1 The architecture of BWI-CBR](image)

4. Notation and definition

Assume a set of cases $C$ is stored in a CBR system for a specific domain, denoted $DOM$, for reasoning. The $i$-th 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 as $A=\langle A_1, A_2, \ldots, A_r \rangle$. The value of an attribute $A_k$ for a case $c_j$ is denoted $V_k(j)$, which can not be null. The attribute values of a case $C_j$ can then be represented as $V_j(j)=\langle V_{1j}, V_{2j}, \ldots, V_{dj} \rangle$. 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 a possible attribute value of $A_i$.

In a CBR system, the set of prior cases need to be stored in the case base for solving the new case. The significance attribute set $A$ of the case helps the CBR system to select relevant cases, which can provide assistance in solving the new case. The matching function is used to evaluate cases based on a weighted sum of matched attributes with the new case. Attribute value can thus be used for indexing a case. An index of 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:

$$IND_k = \{ A_1 = V_1(k), A_2 = V_2(k), \ldots, A_\partial = V_\partial(k) \}.$$ 

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 bit-wise indexing vector used in the proposed indexing method is defined as follows:

DEFINITION 3 (Bit-wise indexing vector of an attribute):
The bit-wise indexing vector $B_i$ of the $i$-th attribute for case $C_k$ is a bit string $B_i = b_1 b_2 \ldots b_{i(\partial i)}$, where $b_j = 1$ if $V_i(k) = V_{ij}$ and $b_j = 0$ otherwise.

DEFINITION 4 (Bit-wise indexing vector of a case):
A bit-wise indexing vector $BWI_k$ of a case $C_k$ is the concatenation of the bit-wise indexing vectors of all the attributes for case $C_k$. That is, $BWI_k = B_1 B_2 \ldots B_r$, where $r$ is the number of attributes.

DEFINITION 5 (Matrix of bit-wise indexes for case-based reasoning):
A matrix $T_{BWI}$ of bit-wise indexing for CBR is represented as

$$\begin{bmatrix}
BWI_1 \\
BWI_2 \\
\vdots \\
BWI_{|C|}
\end{bmatrix},$$

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

Example 1: Assume that a CBR system contains five cases are shown in Table 1.

<table>
<thead>
<tr>
<th>OS</th>
<th>PL</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>WinNT</td>
<td>C</td>
</tr>
<tr>
<td>Case 2</td>
<td>OS2</td>
<td>Basic</td>
</tr>
<tr>
<td>Case 3</td>
<td>Linux</td>
<td>Java</td>
</tr>
<tr>
<td>Case 4</td>
<td>Mac</td>
<td>Java</td>
</tr>
<tr>
<td>Case 5</td>
<td>Solaris</td>
<td>Pascal</td>
</tr>
</tbody>
</table>

Table 1. Five cases in Example 1

The bit-wise indexes for the above cases are shown in Table 2.

<table>
<thead>
<tr>
<th>$BWI_1$</th>
<th>10000</th>
<th>1000</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>$BWI_2$</td>
<td>01000</td>
<td>0100</td>
<td>010</td>
</tr>
<tr>
<td>$BWI_3$</td>
<td>00100</td>
<td>0010</td>
<td>001</td>
</tr>
<tr>
<td>$BWI_4$</td>
<td>00010</td>
<td>0010</td>
<td>010</td>
</tr>
<tr>
<td>$BWI_5$</td>
<td>00001</td>
<td>0001</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 2. The bit-wise indexes of the cases in Table 1

5. Indexing phase in Bit-Wise Indexing CBR
The bit-wise indexes for prior cases are generated by the following two algorithms:

**Algorithm 5.1 (Bit-wise indexes creation algorithm):**

**Input**: a case $C_i$.

**Output**: The bit-wise index $BWI_i$ of $C_i$.

**Step 1**: Create a bit-wise 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 $BWI_i$.

**Algorithm 5.2 (Matrix of bit-wise indexes creation algorithm):**

**Input**: A set of cases.

**Output**: The bit-wise index matrix $T_{BWI}$ of the cases.

**Step 1**: Create an empty matrix $T_{BWI}$ and set the counter $i$ to 1.

**Step 2**: Repeat the following sub-steps for each case $C_i$ until all cases are processed.

- **Sub-step 2.1**: Use *bit-wise indexes creation algorithm* to get the index $BWI_i$ of $C_i$.

- **Sub-step 2.2**: Add $BWI_i$ into $T_{BWI}$.

**Step 3**: Return $T_{BWI}$.

After a bit-wise indexing matrix is built, bit-wise operations can easily be used to retrieve similar cases in CBR.

**6. Matching Phase in Bit-Wise Indexing CBR**

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

**Algorithm 6.1 (Similar-Cases-Seeking Algorithm):**

**Input**: The bit-wise index matrix $T_{BWI}$ and a new case $C_N$.

**Output**: A set of similar cases $Rc$ with their corresponding similarity degrees.

**Step 1**: Use *Bit-wise indexes creation algorithm* to get the index $BWI_N$ of new case $C_N$.

**Step 2**: Initialize the counter $j$ to 1 and let $Rc$ be empty.

**Step 3**: For each $BWI_j$ in $T_{BWI}$, do the following sub-steps (where $1<j\leq|C|$):

- **Sub-step 3.1**: Call the *Search-relevant Algorithm* (described below) to compare the relevance degree $rdi_j$ between $BWI_j$ and $BWI_N$.

- **Sub-step 3.2**: If $rdi_j=0$, ignore the case $C_j$ and go to Step 3.4.

- **Sub-step 3.3**: Call the *Similarity-Computing Algorithm* to compute the similarity $sim_j$ between the $C_N$ and $C_j$, and then add case $C_j$ with similarity $sim_j$ into $Rc$.

**Step 3.4**: Add 1 to $j$.

**Step 4**: Sort the cases in $Rc$ in descending order of similarities.

**Step 5**: Output $Rc$.

**6.1 Retrieving relevant cases**
A prior case, if it 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 be easily found by using the ‘AND’ bit-wise operation to compare the two bit vectors. The following Search-relevant-case algorithm is thus proposed to achieve this purpose:

**Algorithm 6.2 (Search-relevant-case algorithm):**


Output: the relevant degree $rdi_j$.

Step 1: Use the ‘AND’ bit-wise operation on $BWI_N$ and $BWI_j$ and store the result as $rdi_j$, which is also a bit string.

Step 2: Return $rdi_j$.

Since the ‘AND’ bit-wise operation is fast, the Search-relevant-case algorithm can be used to select relevant prior cases quickly. In real implementation, integers are used to represent indexes and $rdi$. If $rdi$ is zero, then the bit vector of the prior case will be filtered. By this way, all irrelevant prior cases can be filtered out efficiently and precisely.

### 6.2 Computing similarity

After all relevant prior cases have been retrieved the similarities between the new case and these relevant prior cases need to be computed in order to select the useful prior cases. As mentioned above, a matching function based on a weighted sum of matched attributes is defined to retrieve similar cases. Each attribute has its own weight. The similarities between a new case and relevant prior cases can not be computed without knowing which attributes have the same values. Since a case has only one value for an attribute, at most $r$ bit of an attribute in $rdi$ is set after the Search-relevant-case algorithm is executed. Accordingly, a special bit-wise vector, called a Mask Vector, is proposed to help compute similarities. Let $<1>$ be the string of length $\alpha$ with all 1’s and be the string of length $\alpha$ with all 0’s. The definition of the Mask Vector is shown below.

**DEFINITION 7 (Mask Vector):**
A bit-wise 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 $Mask_k=S_1S_2...S_r$, where for $S_k=<1>$ and $S_i=<0>$ for $i\neq k$.

By applying the 'AND' operation on $Mask_k$ and the bit-wise vector $rdi$ generated from the search-relevant-cases algorithm, the similarities between a new cases and prior cases for attribute $A_k$ can be easily found. A similarity-measuring function for BWI-CBR is the defined as:

$$SIM(Case_i) = \frac{\sum_{j=1}^{\alpha} (PC_{ij} \times W_j)}{\sum_{j=1}^{\alpha} W_j},$$

where $SIM(Case_i)$ is the similarity between the $i$-th prior case and then new case, $W_j$ is the weight of the $j$-th attribute $j$. $PC_{ij} = 0$ if the result of performing AND bit-wise
operation \( rdi_i \) and \( Mask_j \) is 0 and \( PC_j = 1 \), otherwise.

Several prior cases may have the same similarity with a new case as long as they have the same feature sets matched. This is especially commonly seen when the number of features is small. For this situation, the cost for calculating the similarities of all the prior cases can be reduced if all possible similarities can be pre-computed and stored into the Similarity Mapping List. Each element in the Similarity Mapping List is a similarity value for some feature matched. Thus, the similarity of a prior case with a new case for known features matched can be easily found from the list, instead of from calculation by the formula. The Similarity Mapping List is formally defined as follows.

**DEFINITION 8 (Similarity Mapping List):**

Let \( L \) be a Similarity Mapping List. \( L_i = \sum_{j=1}^{\partial} b_{ij} \times W_j \) be an element in \( L \) with an index value \( i \), determined from the features matched, \( L_i \) can thus be represented as a binary code \( b_{i1}b_{i2}...b_{ir} \), with \( b_{ir} = 1 \) if the \( j \)-th feature is matched, where \( 1 \leq i \leq 2^{\partial} - 1 \).

**Algorithm 6.3 (Similarity Mapping List Creation Algorithm):**

- **Input:** Weight for indexes of CBR.
- **Output:** The similarity mapping list \( L \).
- **Step 1:** Initialize the counter \( k \) to 1 and the list \( L \) to be empty.
- **Step 2:** For each \( k \), do the following sub-steps:
  - Sub-step 2.1: Encode \( k \) into a binary string \( <b_{i1}b_{i2}...b_{ir}> \).
  - Sub-step 2.2: Calculate the similarity degree \( L_k \) by the formula in Definition 8.
  - Sub-step 2.3: Put \( L_k \) into the list \( L \).
  - Sub-step 2.5: If \( k = 2^{\partial} - 1 \), exit the processing of Step 2; Otherwise, set \( k = k + 1 \) and repeat Step 2.
- **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 \( j \), the Mask Vector, and the Similarity Mapping List \( L \).
- **Output:** The similarity of case \( j \).
- **Step 1:** Initialize an empty binary string of length \( r \).
- **Step 2:** For each \( i \), set the \( i \)-th position in the string to 1 if the result of using the ‘AND’ bit-wise operation on \( Mask_i \) and \( rdi_j \) is not all 0; set it to 0 otherwise.
- **Step 3:** Transform the binary string into an integer \( j \).
- **Step 4:** Get \( L_j \) from the Similarity Mapping List.
- **Step 5:** Return \( L_j \).

During, we had constructed Since the Similarity Mapping List and the Mask Vector are constructed in the pre-processing step, and only the ‘AND’ bit-wise operation needs to be done on Mask Vector and bit-wise vectors of relevant cases in the Similarity computing.
algorithm. The computational time for finding the similarity of each relevant prior case with a new case can thus be significantly reduced.

**Example 2:** An example is given here to show the Similar-Cases-Seeking Algorithm in details. Continuing from Example 1, the BWI\(_N\) of new case \(C_N\), \{OS=Solaris, PL=Java, DB=ORACLE\} is \{00001 0010 010\}. Each BWI\(_i\) in \(T_{BWI}\) is processed as follows:

- For \(BWI_1\): The relevant degree \(rdi_1\) between \(BWI_1\) and \(BWI_N\) are found to be \{00000 0000 000\} by the Search-relevant-case 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 bit in \(rdi_2\) is "1", the Similarity Computing Algorithm is called to compute the similarly \(sim_2\) of case 2 with the new case as 0.333. Case 2 is them 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 calculated as 0.333.

- For \(BWI_4\): \(rdi_4 = \{00000 0010 010\}\). Since more than one of the bit in \(rdi_4\) is "1", case 4 is a relevant case. Its similarity is calculated 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 calculated as 0.333.

After sorting the relevant cases in decreasing order of similarities, the results are shown as follows:

<table>
<thead>
<tr>
<th>Relevant case</th>
<th>Case 4</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similarity</td>
<td>0.667</td>
<td>0.333</td>
<td>0.333</td>
<td>0.333</td>
</tr>
</tbody>
</table>

### 6.3 Discussion of filtering and storage

The new indexing method and the new matching function to speed up retrieving similar prior cases in CBR have been proposed. To use the matching function to compute the similarity is a time-consuming task. If we can filter irrelevant cases first, we can only compute the similarity between relevant cases and the new case. Then we can reduce the time of retrieving useful prior cases. Therefore, we propose the two-phases algorithm to retrieve useful prior cases. At relevant cases retrieving phase, we filter the irrelevant prior cases. And we can only compute the similarity between relevant prior cases and the new case at similarity computing phase. Let us assume that number of the case base is \(N\) and filtering percentage is \(M\).

- **Without filtering**
  
  Time of retrieving useful prior cases\(= ((r + 1) \times t_{\text{and}} + t_{c}) \times N\), where \( t_{\text{and}} \) is ‘AND’ bit-wise operation time and \( t_{c} \) is the Similarity Mapping List seek time.

- **With filtering**
  
  Time of retrieving useful prior cases\(= (M \times ((r + 1) \times t_{\text{and}} + t_{c})) \times N\)

- **Filter percentage vs. Retrieval time:**
  
  Retrieval time with filtering \(= \frac{(M \times ((r + 1) \times t_{\text{and}} + t_{c})) \times N}{((r) \times t_{\text{and}} + t_{c}) \times N}\)

  Retrieval time without filtering \(= (r \times t_{\text{and}} + t_{c}) \times N\)

  The efficiency is increased as

  \[
  \frac{\text{Retrieval time with filtering}}{\text{Retrieval time without filtering}} \cong M
  \]

  Since the proposed method is designed for performance issue, some extra storage spaces are required. In following, the size of extra storage spaces required in our method is...
analyzed.

- The storage space of Matrix of bit-wise indexes $T_{BW} = |C| \times \sum_{i=1}^{r} \alpha(i)$.

  Assume that there are 100000 cases in case base and 16 attributes for each case. For each attribute, there are 4 attribute values. Therefore, the storage space of Matrix of bit-wise indexes $T_{BW} = (100000) \times \sum_{i=1}^{16} 4 \text{bits} = 6400000/8000000 = 0.8 \text{ M bytes}$.

- The storage space of Mask Vector $= r \times \sum_{i=1}^{r} \alpha(i)$.

  Assume that there are 16 attributes for each case. For each attribute, there are 4 attribute values. Therefore, the storage space of the storage space of Mask Vector $= 16 \times \sum_{i=1}^{16} 4 = 1024/8 = 128 \text{ bytes}$.

- The storage space of Similarity Mapping List $L = f \times (2^r - 1)$, where $f$ is the storage space required for storing a real number.

  Assume there are 16 attributes for each cases. Therefore, the storage space of the storage space of Similarity Mapping List $L = f \times 2^{16} - 1 = 4194304/8 = 0.524 \text{ M bytes}$.

  As we can see, the size of extra storage for Similarity Mapping List is an exponential function of $r$. Therefore Similarity Mapping List may not be suitable for the domain with large number of attributes.

7. Parallelized Bit-Wise Indexing CBR

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

7.1 The architecture of parallel bit-wise indexing CBR

The Bit-wise Indexing method of CBR treats each case as a bit vector and the similarities between each prior case and a new case can be computed independently. Therefore, our parallel BWI-CBR Indexing method straightly uses a set of cases as a vector matrix. 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 are used to do the matching. Each processor thus handles about \( m/k \) cases.

### 7.2 Indexing phase in Parallel Bit-Wise Indexing CBR

The indexes (BWI) for the cases are thus equally distributed in the processors given. Each processor thus has sub-matrix of indexes to process. The *bit-wise indexes sub-matrix creation algorithm* is described below:

**Algorithm 7.1 (bit-wise indexes sub-matrix creation algorithm):**

- **Input**: The \( T_{BWI} \) and the number \( k \) of processors.
- **Output**: The sub-matrix of bit-wise indexing for each processor \( ST_{BWI}^i \).

**Step 1:** Set \( l=\left\lfloor \frac{|C|}{k} \right\rfloor \), 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 matrix \( ST_{BWI}^i \).

  **Sub-step 2.1:** Copy the indexes from \( BWI_{1+(i-1)} \) to \( BWI_l \) in \( T_{BWI} \) to \( ST_{BWI}^i \) if \( i<k \). Copy the indexes from \( BWI_{1+(i-1)} \) to \( BWI_{|C|} \) in \( T_{BWI} \) to \( ST_{BWI}^i \), otherwise.

**Step 3:** Return the set of \( ST_{BWI}^i \), \( 0<i\leq k \).

**Example 3:** Assume there are two processors to handle the task in Example 2. The \( ST_{BWI}^i \) for each processor \( i \) is shown as follows:

<table>
<thead>
<tr>
<th>( ST_{BWI}^1 )</th>
<th>( ST_{BWI}^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>10000</td>
</tr>
<tr>
<td>Case 2</td>
<td>01000</td>
</tr>
<tr>
<td>Case 3</td>
<td>00100</td>
</tr>
<tr>
<td>Case 4</td>
<td>00010</td>
</tr>
<tr>
<td>Case 5</td>
<td>00001</td>
</tr>
</tbody>
</table>

### 7.3 Matching phase in parallel bit-wise indexing CBR

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

**Algorithm 6.2 (Parallel similar cases seeking algorithm):**

- **Input**: The set of \( ST_{BWI}^i \)'s for \( k \) processors and a new case \( C_N \).
- **Output**: A set of similar cases \( R_c \) with their corresponding similarity degrees.

**Step 1:** Execute *similar-cases-seeking algorithm* for each processor \( i \) on \( ST_{BWI}^i \) to get similar cases \( R_c_i \) with their similarities.

**Step 2:** Merge all the \( R_c_i \) into \( R_c \) in decreasing order of similarities.

**Step 3:** Output \( R_c \) with their similarities.

**Example 4**

Continuing from Example 2, the relevant cases in both processors are shown as below:

<table>
<thead>
<tr>
<th>( R_c_1 )</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.333</td>
<td>0.333</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( R_c_2 )</th>
<th>Case 4</th>
<th>Case 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.667</td>
<td>0.333</td>
<td></td>
</tr>
</tbody>
</table>

These relevant cases are the merged and sorted in in decreasing order of similarities as shown as follows:

<table>
<thead>
<tr>
<th>Relevant Case</th>
<th>Case 4</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similarity</td>
<td>0.667</td>
<td>0.333</td>
<td>0.333</td>
<td>0.333</td>
</tr>
</tbody>
</table>

Finally, the relevant cases with their similarities are then returned.
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 processors system, running the Microsoft Windows NT multithreaded OS. The system includes 512K L2 cache and 128MB shared-memory. The experiment results along with different number 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.

![Fig. 3. Speed-up of parallel BWI-CBR on two processors.](image)

Next, the performance of the BWI-CBR and the two-processor parallel BWI-CBR is compared. A Pentium-Pro 200 quadruple processors system, running the Microsoft Windows NT multithreaded OS. The system includes 1M L2 cache and 512MB shared-memory. The experiment results along with different number of cases are show in Fig. 5. It can easily be seen that the speed-up increases along with the increase of cases, finally converting to 3.5.

![Fig. 4. Speed-up of parallel BWI-CBR on four processors.](image)

It is obvious that BWI-CBR is quite suitable for parallelized since the bit-wise indexing matrix of the proposed method can easily be separated into several independent, nearly equal-sized sub-matrixes. Therefore, when the BWI-CBR is implemented in a multiple CPU machine, the workload can be easily shared on processors and assure that the workloads of all processors are almost balanced.

7. Conclusion and future work

In addition to accuracy, performance should also be taken into consideration in retrieving similar cases 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 indexing method based on bit-wise indexing has been proposed. Several corresponding algorithms,
including the index creation algorithm and the case retrieving algorithm, are described. Some experiments are also made for comparing the performance on multiple processors, with the results showing the proposed parallel method is quite efficient. In the future, we will attempt to modify the indexing method and the corresponding retrieving algorithms to CBR in the data warehousing continually. Also, since the queries of OLAP/OLTP in a data warehouse may be complex and time consuming, a good indexing strategy that can reduce the query time and the computational overhead is our future study.

Reference


