An improved branch-and-bound clustering approach for data partitioning

Chun-Hung Cheng, Kam-Fai Wong and Kwan-Ho Woo

Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Shatin, NT, Hong Kong SAR

E-mail: chcheng@se.cuhk.edu.hk [Cheng]; kfwong@se.cuhk.edu.hk [Wong]; khwoo@se.cuhk.edu.hk [Woo]

Received 1 May 2008; received in revised form 20 January 2009; accepted 10 May 2010

Abstract

In this paper, we are concerned with clustering algorithms for vertical partitioning. In particular, we examine the use of a branch-and-bound scheme. An existing algorithm using such a scheme may produce infeasible solutions to some problems. We adopt the same branch-and-bound scheme and develop a new branching strategy to avoid infeasibility. Illustrative examples are used to demonstrate the effectiveness of our new approach. In addition, we also show how to formulate the horizontal partitioning problem such that the same algorithm can be applied.

Keywords: vertical partitioning; clustering; branch-and-bound

1. Introduction

Distributed technology is an important development in database systems. Data and programs of a database are distributed over a computer network in such a way that they may be accessible from every node in the network. In this environment, response time of the system naturally depends on the distribution of the database programs and data over the network (Abdelguerfi and Wong, 1998; Özsu and Valduriez, 1991). In most of today’s systems, the database programs are typically stored in every node. This is possible as the storage cost has been significantly reduced. The remaining issue is concerned with the handling of data. Specifically, data, which are frequently accessed together, should be placed on near-by nodes (preferably on the same nodes). Nevertheless, the identification of these data groups is not straightforward and simple. In this paper, we deal with the identification of the desirable data groups and examine the use of a branch-and-bound approach.

Given the work-profile of a database application showing the transactions and the data they access, the objective of our algorithm is to cluster data into groups. Individual groups, also known
as database fragments, may then be placed on the nodes where they are frequently accessed. Our approach is applicable only to a relational database model. However, this is not a restrictive assumption as relational database models are by far the most popular. Further, the system’s log is available for the construction of the work-profile. Based on this information, access patterns can be estimated, and from it, important database transactions can be located. Finally, we also assume that the primary key of a relation is duplicated in every vertical fragment produced. In this way, the reconstruction of the whole relation from its vertical fragments is possible through a join operation.

A relation is essentially a table (Date, 1995). PROJECT in Fig. 1 is an example of a relation. The relation PROJECT contains all ongoing projects of a company. This relation may be vertically divided into two smaller units shown in Fig. 2. The primary key ProjNo is duplicated in both relations so that the original relation can be reconstructed.

Note that relation fragments PROJECT2 and PROJECT3 are the tables defined by [ProjNo, ProjName, Location] and [ProjNo, Budget], respectively. These fragments are not randomly formed. It is the role of a database designer to determine how best to partition the original relation in order to achieve the highest level of performance and/or reliability.

<table>
<thead>
<tr>
<th>ProjNo</th>
<th>ProjName</th>
<th>Budget</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>Database development</td>
<td>130000</td>
<td>Hong Kong</td>
</tr>
<tr>
<td>J2</td>
<td>Group technology</td>
<td>115000</td>
<td>Guangzhou</td>
</tr>
<tr>
<td>J3</td>
<td>CAD/CAM</td>
<td>240000</td>
<td>Shanghai</td>
</tr>
<tr>
<td>J4</td>
<td>Maintenance</td>
<td>330000</td>
<td>Beijing</td>
</tr>
</tbody>
</table>

Fig. 1. A PROJECT relation.

<table>
<thead>
<tr>
<th>ProjNo</th>
<th>ProjName</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>Database development</td>
<td>Hong Kong</td>
</tr>
<tr>
<td>J2</td>
<td>Group technology</td>
<td>Guangzhou</td>
</tr>
<tr>
<td>J3</td>
<td>CAD/CAM</td>
<td>Shanghai</td>
</tr>
<tr>
<td>J4</td>
<td>Maintenance</td>
<td>Beijing</td>
</tr>
</tbody>
</table>

Fig. 2. Examples of vertical partitions.

<table>
<thead>
<tr>
<th>ProjNo</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>130000</td>
</tr>
<tr>
<td>J2</td>
<td>115000</td>
</tr>
<tr>
<td>J3</td>
<td>240000</td>
</tr>
<tr>
<td>J4</td>
<td>330000</td>
</tr>
</tbody>
</table>

© 2010 The Authors.
International Transactions in Operational Research © 2010 International Federation of Operational Research Societies
Cheng (1995) applied a branch-and-bound clustering approach for vertical partitioning. Although Cheng’s algorithm works well for many problems, it may produce infeasible (or undesirable) solutions for some problem situations. In this paper, we adopt Cheng’s framework but propose a branching strategy to avoid infeasibility. Although we are only concerned with vertical partitioning, we may easily modify our algorithm to deal with horizontal partitioning. Solving the horizontal partitioning using our proposed algorithm will be discussed later in this paper.

The rest of the paper is organized as follows. In the following section, we discuss the nature of vertical partitioning and review some related work in this area. Section 3 highlights the limitations of the existing branch-and-bound approach. In Section 4, we propose a new formulation of vertical partitioning. Section 5 presents how our algorithm works, and Section 6 shows the effectiveness of the algorithm. In Section 7, we also show how to set up the horizontal partitioning problem such that our algorithm can be used to produce horizontal partitions. Section 8 concludes this paper.

2. Vertical partitioning

Navathe et al. (1984) show that if a relation has \( n \) non-primary key attributes, the number of possible vertical fragments to consider would equal the \( n \)th Bell number, \( B(n) \). For a large value of \( n \), \( B(n) \approx n^n \). Hence, many solution techniques are heuristic based. Hoffer (1976) uses a 0–1 non-linear integer-programming model. The model minimizes storage, retrieval, plus update costs subject to capacity constraints on database subfiles. An approximate solution based on the bond energy (BE) algorithm (McCormick and Schwietzer, 1972) is used. Instead of the BE algorithm, Slagle et al.’s (1975) algorithm may be used. Slagle et al.’s algorithm is a modification of the BE algorithm and may be used to generate solutions of better quality.

Eisner and Severance (1976) propose to identify the most frequently accessed data fragments and place them in high-speed primary memory. This problem is isomorphic to the minicut–maxflow network problem, which can be solved with the Ford/Fulkerson algorithm. However, their method is inefficient for large problems. Hammer and Niamir (1979) propose an approach using a hill-climbing technique that can find a near optimal vertical partition.

Navathe et al. (1984) extend the work of Hoffer. Affinity among attributes is defined in the affinity matrix to express the extent to which they are simultaneously processed. The BE algorithm (McCormick and Schwietzer, 1972) is introduced to partition attributes according to their affinity. Because the BE algorithm does not produce a solution in a diagonal structure, a heuristic algorithm is required to divide attributes into fragments. Cornell and Yu (1987, 1990) develop an integer-programming formulation. At each cycle, the formulation finds an optimal partitioning that splits a relation into two fragments. The formulation can be applied recursively until no profitable split can be found.

Navathe and Ra (1989) propose an algorithm for vertical partitioning that uses a graphical technique. The major feature of this algorithm is that all fragments are generated by one iteration in a time of \( O(n^2) \). However, their algorithm has some undesirable features as we note below. Lin and Zhang (1993) propose a graphical vertical partitioning algorithm to overcome some deficiencies found in the Navathe–Ra algorithm.
In both, Navathe and Ra (1989) and Lin and Zhang (1993), the associated attribute-affinity graph is constructed based on an affinity matrix. The edges with affinity value 0 are removed from the attribute-affinity graph to form an affinity graph. A cycle of an affinity graph is distinguished if the affinity values of the edges on the cycle are larger than the affinity values of the edges going out from the cycle. A distinguished subgraph $G'$ is cycle-atomic if there is no distinguished cycle whose vertex set is a proper subset of the vertex set of $G'$. A graph $G$ is two-connected if the number of vertices is more than two, and if an arbitrary vertex is removed; the resulting graph is still connected. Similar to a cycle-atomic distinguished cycle, we may define a two-connected atomic distinguished subgraph. A completely distinguished subgraph is a complete subgraph, which is distinguished.

Navathe and Ra (1989) observe that a fragment can be produced from a distinguished cycle. A heuristic algorithm is used to find a cycle-atomic distinguished cycle in the affinity graph. However, their algorithm cannot always find a cycle-atomic completely distinguished cycle even if it exists. In Lin and Zhang (1993), their algorithm first finds a two-connected atomic distinguished subgraph instead of a cycle-atomic distinguished cycle. Their algorithm is more efficient than the one in Navathe et al. (1984) and more effective than that in Navathe and Ra (1989).

Chu and Ieong (1993) develop a vertical partitioning algorithm to partition attributes according to a set of transactions. Cheng (1995) proposes a branch-and-bound algorithm for a binary access matrix. Cheng and Motwani (2009) examine different removing strategies to decompose a binary access matrix. Son and Kim (2004) develop an adaptable vertical partitioning method for distributed systems. The unique feature of their approach allows the designers to specify a specific number of data fragments in their designs. Huang and Van (1995) propose an heuristic search algorithm to search the large solution space of partitions and to choose one partition that yields the minimum number of disk accesses by using the A$^*$ technique. Cheng et al. (2002) explore the use of a genetic search clustering algorithm for data partitioning. By formulating the underlying problem as a traveling salesman problem (TSP), they take advantage of this particular structure. Three new crossover operators are also proposed and experimental results indicate that they outperform Slagle et al.’s (1975) algorithm in solving the partitioning problem.

3. Limitations of existing approaches

As we will base our study on Cheng’s (1995) framework to develop our algorithm, we provide a more detailed discussion of Cheng’s algorithm here.

3.1. Background

A work-profile describes the access patterns of a set of transactions, \( \{1, 2, \ldots \} \) say, over the attributes of a database relation, i.e. \( \{1, 2, \ldots \} \). For design purposes, it is commonly modeled by a transaction–attribute access matrix. Consider the transaction–attribute access matrix in Fig. 3. It contains five non-primary key attributes, i.e. \( \{1, 2, 3, 4, 5\} \), and four transactions, \( \{1, 2, 3, 4\} \), accessing the relation. A “1” (or “0”) entry in the matrix indicates that the corresponding
transaction uses (or does not use) the attribute(s) concerned. Note that in the matrix in Fig. 3, the
distribution of “1” in the matrix is completely random.

Consider another transaction–attribute access matrix in Fig. 4. It is formed by rearranging
certain transactions and attributes of the matrix in Fig. 3. The matrix in Fig. 4 comprises a
diagonal cluster structure. A diagonal cluster structure refers to the structure with most of the
“1”s arranged along the diagonal. TC-1 is a transaction group which accesses AC-1, an attribute
group; similarly, TC-2 accesses AC-2.

In Fig. 4, TC-1/AC-1 and TC-2/AC-2 form two perfectly separable smaller matrices (i.e.,
clusters). These smaller matrices formed are called submatrices. All the attributes required by a
transaction are placed in the same cluster. Attributes in a cluster make up a fragment. This lays
down the objective of vertical partitioning. Clearly, such an ideal diagonal cluster structure is
uncommon; especially in real life situations. For instance, consider the transaction–attribute

Fig. 3. A transaction–attribute access matrix.

Fig. 4. A rearranged transaction–attribute access matrix.

Fig. 5. A realistic transaction–attribute access matrix.
access matrix in Fig. 5. An ideal diagonal cluster structure does not exist in this matrix because attributes 4 and 5 are accessed by transactions from different clusters. Attributes 4 and 5 are known as inter-cluster attributes.

The treatment of inter-cluster attributes in an actual implementation depends on the access pattern of a system. In a system where retrieval transactions are more than update transactions, a duplication strategy as discussed in this paper is practical. However, in a system where the contrary is true, keeping inter-cluster attributes in a few servers in the computer network makes sense.

Chu and Ieong (1993) consider vertically splitting a relation by forcing inter-cluster attributes to a cluster. They show that the worst-case complexity of an optimal binary vertical partitioning algorithm based on transactions is $O(2^m)$ while that of an optimal algorithm based on attributes is $O(2^n)$, where $m$ and $n$ are the number of transactions and attributes of a relation, respectively. In this work, we consider the duplication of inter-cluster attributes.

3.2. Formulation pitfalls

Cheng (1995) formulates the vertical partitioning problem. The objective is to remove attributes to make the decomposition possible. The formulation is shown as follows:

P1: Remove attributes to decompose a transaction–attribute access matrix into submatrices with the maximum number of “1” entries retained in submatrices (i.e., clusters) subject to the following constraints:

C1: Transactions with all “0” entries in a submatrix are not allowed.
C2: The number of transactions in a submatrix cannot exceed an upper limit, $b$.
C3: Attributes with all “0” entries in a submatrix are not allowed.

Constraints C1 and C3 ensure that an attribute is accessed by at least one transaction in the same submatrix (i.e., the same cluster). An upper limit on the number of transactions in constraint C2 is included to prevent the formation of a single submatrix with all attributes and transactions. A designer may specify and vary the upper limit to examine the effect on the composition of a submatrix (i.e., cluster). The objective function is to maximize the number of “1” entries retained in submatrices. This objective function ensures that attributes in a submatrix will satisfy transactions in the same submatrix as much as they can. The goal is to produce submatrices that cannot be further decomposed without violating the constraints.

Constraint C2 forces a transaction–attribute access matrix to form clusters. Although it is important in preventing the formation of huge clusters with a large number of transactions and attributes, it does not ensure the quality of clusters formed. In addition, this constraint assumes that a single value is used as the upper limit on all clusters formed. However, it is possible that different upper limits for different clusters may be needed. Certainly, we may try to formulate the problem in such a way that different upper limits are used for different clusters. This effort turns out to be difficult. Because the formulation does not specify the number of clusters formed, it is impossible to determine how many upper limits are needed in advance. Even if this had been possible, the number of possible values would have been significantly increased.

© 2010 The Authors.
International Transactions in Operational Research © 2010 International Federation of Operational Research Societies
We will modify this formulation by dropping constraint C2 but including a constraint to ensure that the quality of a cluster (i.e., a submatrix) is acceptable. In our formulation, we make use of the cohesion measure to measure the quality of a cluster formed. The cohesion measure of a cluster is designed to assume a value between 0 and 1. The use of a measure to ensure the quality of clusters formed is a common practice in cluster analysis (Anderberg, 1973). As it is shown later in this paper, the cohesion measure is effective in forming good quality clusters.

3.3. Algorithm pitfalls

Cheng’s algorithm is based on a branch-and-bound approach that iteratively examines each unfathomed node in the enumeration tree using cluster identification algorithm (Kusiak and Chow, 1987). The root node of the tree contains only one matrix (e.g., an initial matrix). Other nodes in the tree, representing feasible or infeasible solutions, may contain several submatrices. When a feasible solution is found and its objective value is better than the best solution found so far (i.e., the incumbent solution), this feasible solution becomes the incumbent solution. When the branch-and-bound procedure is completed, the incumbent solution is the final solution.

When a submatrix is not separable but contains more transactions than the upper limit, a branching scheme is used to make the submatrix separable. The branching scheme uses a void measure to estimate the likelihood of an attribute being the factor that prevents the submatrix from being decomposed (i.e., the likelihood that an attribute is an inter-cluster attribute). Then the branching scheme removes the attribute from the submatrix and decomposes the submatrix without the removed attribute. However, removing attributes may generate an infeasible solution. Consider the matrix in Fig. 6. The removal of attributes 4 and 5 makes transactions 3 and 4 dangling (i.e., they do not have any attributes to access).

Our new strategy requires the duplication of attributes for decomposing the matrix in concern. The duplication strategy does not create a situation with dangling transactions. Hence, it can avoid the infeasibility and dangling problem that may occur in Cheng (1995). The problem we have to deal with is which attributes to duplicate. Obviously, duplicating every attribute is inefficient and hence, impractical.

![Fig. 6. The solution by Cheng (1995) to the matrix in Fig. 5.](image-url)
4. Problem formulation

Our formulation is different from Cheng’s formulation in two ways. First, instead of removing strategies in Cheng (1995) and Cheng and Motwani (2009), we duplicate attributes to decompose a matrix. Second, unlike Cheng (1995), we require a constraint to ensure the quality of a cluster formed. A measure is developed to gauge the quality of a cluster. The developed measure assumes a value between 0 and 1. A designer may specify the value of this parameter.

To ensure the quality of a cluster (i.e., a submatrix) formed, we develop a cohesion measure for all transactions in the same cluster. Suppose we have a transaction–attribute submatrix \( S = [a_{ij}] \), where \( i = 1, \ldots, m \) and \( j = 1, \ldots, n \). Further we let \( R \) be the set containing all row indices of \( S \), and \( C \) be the set containing all column indices of \( S \). Then the cohesion measure of submatrix \( S \), \( \delta_S = |\{a_{ij} = 1, i \in R \text{ and } j \in C\}|/(|R|*|C|) \), where \(|R|\) and \(|C|\) are the cardinality of sets \( R \) and \( C \), respectively.

Having defined a cohesion measure of a submatrix (i.e., a cluster), we formulate our problem as follows:

P2: Duplicate attributes for the minimal number of times to decompose a transaction–attribute access matrix into separable submatrices subject to the following constraints:

C4: Transactions with all “0” entries in a submatrix are not allowed.
C5: Attributes with all “0” entries in a submatrix are not allowed.
C6: The cohesion measure of a submatrix is more than or equal to a threshold, \( \delta \).

Formulation P2 is a minimization problem. Our strategy for clustering is to duplicate attributes. Constraint C6 explicitly imposes a threshold to ensure the acceptable level of the quality of a cluster. Constraints C4 and C5 must be satisfied throughout the branch-and-bound process in our new strategy. As mentioned before, instead of removing attributes, we duplicate the attributes in the problem matrix. For each transaction, the attributes are duplicated such that the access pattern remains unchanged. As a result, the problem of dangling transactions never happens in our approach and constraints C4 and C5 may become irrelevant in the actual implementation.

5. Solution method

Our algorithm uses a branch-and-bound framework by Cheng (1995). It transforms an inseparable problem matrix (obtainable from a node in the branch-and-bound enumeration tree) into another matrix by duplicating attributes. The transformed matrix will first go through the process of clustering and will subsequently be placed back in a node.

In the tree, the root node contains only the original problem matrix. Other nodes contain the associated transformed matrices. The algorithm uses the cluster identification algorithm at each unfathomed node to identify the set of possible clusters. The set of clusters identified will form part of a feasible solution only if these clusters have the cohesion measures larger than or equal to the threshold specified. The overall framework of the clustering approach is given in Fig. 7. The overall objective of the approach is to duplicate attributes for the minimal number of times to
decompose a transaction–attribute access matrix into separable submatrices subject to various constraints (C4–C6).

We use a stack data structure to implement the branch-and-bound enumeration tree. The push-and-pop mechanism of a stack ensures a depth-first-search strategy used in our implementation. A node contains one or more matrices.

As shown in Fig. 7, the problem matrix forms the initial node and the upper bound is set to infinity (Lines 4–5). In the while loop (Lines 7–35), we pop the node from the top of the stack and perform cluster identification (the details of which will be discussed later in this section). If the solution of the node is feasible and is better than the current incumbent solution, the set of the

Fig. 7. The clustering approach.
associated submatrices will become the new incumbent solution and the upper bound will be updated (Lines 12–15). On the other hand, if the lower bound of the current node is not lower than the upper bound, the node will be pruned (i.e., no new child nodes will be derived from the current node) (Lines 16–20). If both cases are not true, a branching rule (Lines 21–33) (which is further explained in this section) is applied. A set of new nodes are derived by duplicating the attributes in the matrix of the current node. All new child nodes in the set are then pushed back to the stack for enumeration.

To avoid duplicating the same set of attributes more than once, we use a set of flag BF_redundant in the algorithm to indicate that the set of attributes has been previously duplicated. The algorithm continues until the stack is empty. The incumbent solution at the termination of the while loop is the best solution of the branch-and-bound approach.

The best solution found is a set of submatrices with the minimum number of attribute duplications. Our branch-and-bound approach identifies the possible clusters, but it may produce very small clusters. Hence, we try to merge clusters into bigger clusters without increasing the value of the objective function and violating any constraints (Lines 38–39). We will discuss the actual implementation of the merge procedure later in this section.

In summary, our proposed clustering algorithm involves two phases: the branch-and-bound phase and the clusters merging phase. The former phase identifies the inter-cluster attributes which hinder the decomposition of the problem matrix, and the latter phase groups the fragmented clusters together if it is possible to do so.

5.1. Branching rule

Cheng (1995) introduces the use of void measure to estimate the likelihood of an attribute being an inter-cluster attribute. The algorithm then removes the inter-cluster attribute to make the decomposition of a problem matrix possible. In our approach, we have modified the rule by Cheng (1995) for our use. To illustrate our branching rule, we define

- \(a_{ij}\) = entries in the \(i\)th row and \(j\)th column of an \((m \times n)\) matrix \(M\),
- \(Q_j = \{j' \neq j: a_{ij'} = 1\}\) for some \(i = 1, 2, \ldots, m\) such that \(a_{ij} = 1\),
- \(C_j = Q_j \cup \{j\},\)
- \(R_j = \{i: a_{ij} = 1\}\) and (for some \(j' \in C_j: a_{ij'} = 1\)}.

The branching rule is composed of the following two steps:

Step 1 Evaluation of void measure:
Calculate the void measure of attribute \(j\), \(V_j\), for every attribute \(j\) in \(M\). The void measure for attribute \(j\) is defined as
\[V_j = |\{a_{ij} = 0, i \in R_j\} \text{ and } j' \in C_j\}|.

Step 2 Branching:
Sort the attributes by the value of void measures in a descending order. The attributes with the highest void measure will be considered first in the enumeration tree. For each attribute, split it into a set of attributes in which each has a unique single entry “1” derived from the original attribute being considered.
Take the simple matrix in Fig. 5 as an example. The matrix is not decomposable originally. The branching rule is applied. Row has all 1s and hence $C_j$ contains \{1, 2, 3, 4, 5\} for all $j$. Also, $R_1 = R_2 = \{1, 2\}$, $R_2 = \{2\}$ and $R_4 = R_5 = \{2, 3, 4\}$. It follows that $V_1 = V_2 = 3$, $V_3 = 0$ and $V_4 = V_5 = 6$. Ties are broken randomly and we assume that attribute 5 is duplicated first. The resultant matrix is still not decomposable. As a result, the branching rule is applied again. Attribute 4 is chosen for duplication and at this time, the matrix can be decomposed. Figure 8 illustrates how the attributes are duplicated for matrix decomposition.

5.2. Cluster identification

In Cheng’s (1995) approach, Kusiak and Chow’s (1987) cluster identification is proposed for examining each node in the enumeration tree. However, the cluster identification algorithm is not efficient as it requires $O(\min(n,m) \times n \times m)$ for a regular $m \times n$ matrix. In our implementation, we consider a matrix as a set of row bitsets. Because a bitset can be manipulated by bitwise operations which can be done in a linear time, the complexity of cluster identification in this implementation depends on the number of the rows (i.e., the transactions) only, irrespective of how large the number of columns (i.e., the attributes) is. In the actual implementation, the ideal complexity, however, may not be achievable, because a computational machine can only handle a limited length of a bitset. When a bitset is longer than a machine can handle, the bitset has to be chopped up into a set of chunks for the bitwise operations.

The CLUSTER_IDENTIFICATION algorithm (shown in Fig. 9) aims at decomposing the input matrix $M$ into a set of submatrices (i.e., clusters) with the cohesion measure threshold $\delta$ being satisfied.

In the CLUSTER_IDENTIFICATION algorithm (shown in Fig. 9), we initialize the matrix being processed ($M_p$) as the input problem matrix. In the while loop (Lines 4–23), we apply the BINARY_SPLIT algorithm to split the matrix $M_p$ (Line 5). If $M_p$ is decomposable, it will be split into two parts, say $M_1$ and $M_2$, in which $M_1$ is the submatrix candidate for forming part of a cluster solution while $M_2$ is the remaining submatrix (see details later). If the cohesion measure of $M_1$ is larger than or equal to the threshold $\delta$, $M_1$ will be part of a cluster solution and $M_2$ will be treated as the revised $M_p$ for further decomposition (Lines 7–9). Otherwise (i.e., the cohesion measure of $M_1$ is smaller than the threshold $\delta$), the algorithm will terminate the while loop and return the infeasibility status to the branch-and-bound algorithm (Lines 10–13). The same case applies to a non-decomposable $M_p$ (Lines 15–20).
In order to understand how $M_1$ and $M_2$ are identified in each iteration, we consider the BINARY_SPLIT algorithm in Fig. 10. To identify the submatrices, two sets of flags, $MR_1$ and $MC_1$, are used to indicate which rows and columns belong to $M_1$ while the others belong to $M_2$. The algorithm first initializes the first row of input matrix $M$ and its associated columns to be the row and column elements of $M_1$, respectively. This can be done by marking the flags of the row (i.e., 1) and its possessed columns in $MR_1$ and $MC_1$, respectively (Line 1). The algorithm then scans all the rows of the input matrix $M$ repeatedly and groups the rows and their associated columns to $M_1$ if they share common columns (i.e., attributes). Grouping these rows and columns to $M_1$ can also be done by marking the corresponding flags of rows and columns in $MR_1$ and $MC_1$, respectively (Lines 3–11). If no cluster can be obtained from the input matrix $M$ (i.e., matrix $M$ is identical to matrix $M_1$), the algorithm will return the non-separable status back to CLUSTER_IDENTIFICATION (Lines 14–15). Otherwise, it will return two submatrices, say $M_1$ and $M_2$, to CLUSTER_IDENTIFICATION. In the CLUSTER_IDENTIFICATION algorithm, $M_1$ is the submatrix candidate for forming part of a cluster solution. All the columns should be unique in matrix $M_1$ (i.e., the attributes duplicated from the same attribute cannot coexist in the same cluster). As a result, $M_1$ has to be formed by grouping all the rows identified in $MR_1$ and their associated columns in the

![Fig. 9. Algorithm for cluster identification.](image-url)
original problem matrix. This can guarantee the uniqueness in the set of columns in matrix $M_1$. On the other hand, $M_2$ will remain as a submatrix for further division. $M_2$ has to be formed by grouping the complement set of the rows and columns identified in $MR_1$ and $MC_1$.

Figure 11 illustrates how cluster identification identifies submatrices (clusters) from the transaction–attribute access matrix (shown in Fig. 8) when attributes 4 and 5 of the matrix are duplicated.

5.3. Clusters merging process

As mentioned above, a scheme is necessary for merging clusters into a larger group without increasing the value of the objective function and violating any constraints. A useful solution
would be the one with the least number of clusters. The most direct implementation of the merging scheme is to try to merge all the possible combinations of the clusters in the solution. The feasible clustering solution with the least number of clusters will be the best solution. However, this brute-force method is very time consuming when there are many submatrices. As a result, a fast method is proposed for getting a good solution.

In the proposed algorithm, a pair of clusters can be merged together in the solution set only if the resulting cohesion measure of the merged cluster is larger than or equal to the threshold \( d \) and it has the highest value among all the pairs. The algorithm terminates when no pair is suitable for merging. The solution set will then be the cluster solution for our proposed clustering algorithm.

The implementation of the algorithm MERGE is shown in Fig. 12. The set of clusters resulted from the branch-and-bound algorithm is used in the MERGE algorithm. In each iteration of the do-while loop (Lines 2–15), the algorithm evaluates the resulting cohesion measures of merging all the possible pairs of clusters (Lines 4–7). The pair of clusters is merged together if its resultant cohesion measure value is larger than or equal to the threshold \( \hat{d} \) and it is the largest value among all the pairs considered (Lines 8–10). The algorithm will terminate when there is no possible merger in the cluster set (Line 15). Finally, Fig. 13 illustrates how the fragmented set of clusters formed by cluster identification are merged together to form a larger cluster.

6. Effectiveness of the algorithm

In this section, we examine the effectiveness of the algorithm. First, we discuss its computational complexity and memory requirement. Then, we use an illustrative example to show the mechanics of the algorithm. Finally, we carry out a computational study to understand the computational aspect of the algorithm.

© 2010 The Authors.
International Transactions in Operational Research © 2010 International Federation of Operational Research Societies
6.1. Computational complexity and memory requirement

Our proposed partitioning approach is comprised of the branch-and-bound phase and the cluster merging phase. In the former phase, the branch-and-bound approach is in fact an adaptive strategy for exhaustive searching the set of the columns to be duplicated in a matrix. An exhaustive search results in $O(n^2)$ in complexity, where $n$ is the number of the columns in a problem matrix.

For each node examined, we perform (1) cluster identification, (2) evaluation of the cohesion measure, (3) evaluation of the void measure for each attribute as well as (4) the formation of a transformed matrix (matrix with a particular column being duplicated). Assume that a problem matrix is of size $m \times n$. The time complexities of the above-mentioned processes are $O(m^2)$, $O(m)$, $O(n \times m)$ and $O(m)$, respectively. As discussed previously, we represent the matrix using a set of row bitsets. Therefore, the time complexities of most of the processes are independent of the

![Fig. 12. Algorithm for merging submatrices.](image)

![Fig. 13. An example of merging the clusters for eliminating the fragmentation problem.](image)
number of columns and the bitwise operation can be manipulated in constant time. Overall, the examination of a node requires $O((n+m) \times m)$.

Now we consider the cluster merging phase. The time complexity would be as bad as $O(n^3)$ if the brute-force merging algorithm is applied. This is certainly not acceptable. Hence, we have proposed a fast algorithm whose time complexity is $O(n^3)$.

Compared with the branch-and-bound phase, the time required for the merging phase is negligible. As a result, the clustering algorithm requires $O(n^a \times (n+m) \times m)$ in terms of time complexity for clustering an $m \times n$ matrix. In fact, with the help of applying the branching strategy, the complexity may be reduced to $O(k \times (n+m) \times m)$, where $k$ is a value representing the number of nodes to be examined which is much smaller than $n^3$ in practice.

Next, we consider the space complexity of our algorithm. In the branch-and-bound phase, we have to conduct a search on the enumeration tree to find the best solution. For an $m \times n$ problem matrix, it would require $O(n!)$ nodes to build the whole enumeration tree. This is impractical for a realistic problem. In our algorithm, we generate a node on an as needed basis. This can be done by performing the push-and-pop mechanism of a stack data structure. With the use of stack data structure, the number of nodes in memory is reduced to $\sum_{i=0}^{n-1} (n-i) = \frac{1}{2} n(n+1)$ and, therefore, the memory requirement is as bad as $O(n^2)$. $O(m \times n)$ is required to represent the structure of an $m \times n$ matrix, and hence the space complexity for the branch-and-bound phase is $O(m \times n^3)$. As only one node is considered in each iteration of the merging phase, the space complexity is then $O(m \times n)$. As a result, the overall space complexity of our algorithm is $O(m \times n^3)$. In the actual implementation, because the matrices are represented by bitset, the memory space occupied is estimated to be $O(m \times n^3) \times 1$ bit. The memory requirement in fact is very small and should not pose any problem to an average computational machine.

6.2. Illustrative example

Consider the enumeration tree in Fig. 14 as an example. The problem matrix is shown as the root node of the search tree. Assume that the lowest threshold for a cluster, $\delta$, is 0.7. The problem matrix cannot be decomposed and hence transformation is required. By evaluating the void measure for each attribute in the problem matrix, we find that attribute 1 has the highest value. Attribute 1 is being duplicated first in the search tree. Cluster identification is then performed on the transformed matrix and it is found that one of the clusters does not fulfill the cohesion measure threshold. Further branching is necessary and at this time attribute 2 is found with the highest void measure. Hence, it is being duplicated in the matrix. Cluster identification is again performed on the latest transformed matrix. Because all the clusters formed satisfy the criterion, the set of clusters found is set as the incumbent solution and the upper bound is updated. The search then backtracks to other sibling nodes at Level 2. Because these nodes have the lower bound $Z_L = 2$, they are excluded from consideration. The search backtracks to Level 1 of the search tree and attribute 2 is being duplicated. With the duplication of attribute 2, the matrix is still not decomposable. A further duplication is required. However, the further duplication will only result in a matrix with its $Z_L = Z_U$. Hence, further processing along this path is not necessary. The search continues until there is no unfathomed node in the search tree.
Although the inter-cluster attributes are identified as reflected in the solution, some clusters in the solution formed are in fragmentation. Clusters with attribute \{1\} and with attributes \{100,2\} could be grouped together to form a larger cluster (see Fig. 15).

To illustrate the effectiveness of our algorithm, let us consider another larger problem with the threshold being 0.6. In Fig. 16, the transaction–attribute access matrix contains 11 transactions and 22 attributes. The matrix does not have any revealable structure.

The solution is shown in Fig. 17. Attributes 4, 5, 6, 8, 9, 11, 12, 14, 17, 18 and 19 are inter-cluster attributes. The addition of extra copies of these attributes allows the matrix to be decomposed into four clusters.

Fig. 14. Search tree for illustrating the branch-and-bound approach.
6.3. Experimental result

The branch-and-bound clustering algorithm was implemented in C++ on Solaris SUN-BLADE 1000 with ultra-sparc III+CPU running at 900 MHz. We have 10 matrices with different sizes and structures for the experiment. These matrices are artificially created to contain a pattern. Then an entry in a matrix is randomly picked and swapped from “0” to “1” or “1” to “0”. The number of times of swapping is also randomly determined. For each problem, the branch-and-bound

algorithm is run with nine different cohesion measure thresholds for experiments. We have limited the running time for each experiment to be within 24 h.

Tables 1 and 2 report the computational time and the final upper bound found (i.e., the minimum number of inter-cluster attributes) for different matrices with different thresholds, respectively. From these tables, we can see that setting cohesion measure appropriately is very important. If the threshold is set too high (e.g., 0.8 or 0.9), the algorithm will incur a long computational time. Also, the upper bound is close to the number of the columns. This tells us that nearly all the attributes have to be duplicated (i.e., nearly all of them are considered as inter-cluster attributes). Only the attributes with almost the same access pattern can be grouped together. As a result, many small clusters are formed and heavy fragmentation occurs in a clustering solution. In fact, there is no practical meaning for this solution. On the other hand, the

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Cohesion measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>1 7 × 11</td>
<td>0</td>
</tr>
<tr>
<td>2 20 × 20</td>
<td>0</td>
</tr>
<tr>
<td>3 23 × 20</td>
<td>0</td>
</tr>
<tr>
<td>4 10 × 22</td>
<td>0</td>
</tr>
<tr>
<td>5 11 × 22</td>
<td>0</td>
</tr>
<tr>
<td>6 20 × 25</td>
<td>0</td>
</tr>
<tr>
<td>7 12 × 30</td>
<td>0</td>
</tr>
<tr>
<td>8 20 × 50</td>
<td>0</td>
</tr>
<tr>
<td>9 25 × 35</td>
<td>0</td>
</tr>
<tr>
<td>10 30 × 40</td>
<td>0</td>
</tr>
</tbody>
</table>

The algorithm cannot terminate within 24 h.

Table 2

The upper bound (Z_U) found for the problem matrices with different thresholds

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Cohesion measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>1 7 × 11</td>
<td>0</td>
</tr>
<tr>
<td>2 20 × 20</td>
<td>0</td>
</tr>
<tr>
<td>3 23 × 20</td>
<td>0</td>
</tr>
<tr>
<td>4 10 × 22</td>
<td>0</td>
</tr>
<tr>
<td>5 11 × 22</td>
<td>0</td>
</tr>
<tr>
<td>6 20 × 25</td>
<td>0</td>
</tr>
<tr>
<td>7 12 × 30</td>
<td>0</td>
</tr>
<tr>
<td>8 20 × 50</td>
<td>0</td>
</tr>
<tr>
<td>9 25 × 35</td>
<td>0</td>
</tr>
<tr>
<td>10 30 × 40</td>
<td>0</td>
</tr>
</tbody>
</table>

The branch-and-bound process cannot be finished within 24 h, and only an intermediate result is available.
The algorithm does not identify any cluster at all when the threshold set is too low (e.g., 0.1 or 0.2). With the low threshold, even highly dissimilar rows and columns can be grouped together. It defeats the purpose of clustering. Hence, such a solution is not practical for database partitioning. In general, experiments with different thresholds are necessary for getting the appropriate partitioning solution.

Tables 3 and 4 show the total number of nodes and the number of nodes required for getting the best solution, respectively. The large difference between the number of nodes generated and the number of nodes traversed to find the best solution shows the effectiveness of our branch-and-bound scheme. As we observe in Tables 3 and 4, only a small number of nodes as opposed to the

### Table 3
The total number of nodes in the search tree during the branch-and-bound process

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Cohesion measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>1 7 × 11</td>
<td>1</td>
</tr>
<tr>
<td>2 20 × 20</td>
<td>1</td>
</tr>
<tr>
<td>3 23 × 20</td>
<td>1</td>
</tr>
<tr>
<td>4 10 × 22</td>
<td>1</td>
</tr>
<tr>
<td>5 11 × 22</td>
<td>1</td>
</tr>
<tr>
<td>6 20 × 25</td>
<td>1</td>
</tr>
<tr>
<td>7 12 × 30</td>
<td>1</td>
</tr>
<tr>
<td>8 20 × 50</td>
<td>1</td>
</tr>
<tr>
<td>9 25 × 35</td>
<td>1</td>
</tr>
<tr>
<td>10 30 × 40</td>
<td>1</td>
</tr>
</tbody>
</table>

m stands for million times.

### Table 4
The number of nodes traversed when the best clustering solution is found

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Cohesion measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>1 7 × 11</td>
<td>1</td>
</tr>
<tr>
<td>2 20 × 20</td>
<td>1</td>
</tr>
<tr>
<td>3 23 × 20</td>
<td>1</td>
</tr>
<tr>
<td>4 10 × 22</td>
<td>1</td>
</tr>
<tr>
<td>5 11 × 22</td>
<td>1</td>
</tr>
<tr>
<td>6 20 × 25</td>
<td>1</td>
</tr>
<tr>
<td>7 12 × 30</td>
<td>1</td>
</tr>
<tr>
<td>8 20 × 50</td>
<td>1</td>
</tr>
<tr>
<td>9 25 × 35</td>
<td>1</td>
</tr>
<tr>
<td>10 30 × 40</td>
<td>1</td>
</tr>
</tbody>
</table>

m stands for million times.

---

© 2010 The Authors.
International Transactions in Operational Research © 2010 International Federation of Operational Research Societies
number of total nodes have been examined to find the best solution. In addition, the total number of nodes is much smaller than $n^k$, where $n$ is the number of columns in the problem matrix. This provides additional evidence that the branch-and-bound algorithm works efficiently in pruning. Finally, Table 5 reports the computational time for merging the fragmented set of clusters. We can see that the process is very effective.

For problems 8–10, the problem is so complicated that the best solution cannot be found within 24 h. Only the intermediate result (i.e., the last found incumbent solution in the branch-and-bound enumeration tree) is shown as the final solution. It is possible that an intermediate result could be a good solution. As mentioned before, the introduction of void measure enables the algorithm to estimate the likelihood of an attribute being an inter-cluster attribute. These inter-cluster attributes are most likely to be identified at the earlier stage of a search. As a result, the intermediate results found for problems 8–10 are all good solutions.

7. Horizontal partitions

This section shows how our algorithm is used to produce horizontal fragments. As shown in Zhang and Orlowska (1994), horizontal partitioning decomposes a relational table along its tuples. Horizontal partitioning of a relation is performed using predicates that are defined on that relation. Given a relation PROJECT in Fig. 1, the SQL query “SELECT * FROM PROJECT WHERE Budget > 140000” will extract information of all projects with budgets over US$140,000. At this point, we are only interested in the simple predicate (i.e., Budget > 140,000). The predicate forms the basis of one of the horizontal partitions shown in Fig. 18.

In general, given a relation $R(A_1, A_2, \ldots, A_n)$, where $A_i$ is an attribute defined over domain $D_i$, a simple predicate $p_j$ defined on $R$ has the form $p_j: A_i \theta Value$ where $\theta \in \{ =, <, \neq, \leq, >, \geq \}$ and Value $\in D_i$. SQL queries containing simple predicates are common. But on some occasions, e.g. in decision support system applications, ad hoc queries are also frequently used. These queries
inevitably involve complicated predicates. A complicated predicate consisting of multiple simple predicates\(^1\) could be decomposed into an equivalent collection of SQL queries with simple predicates. For this reason, irrespective of the complexity of the predicates, horizontal partitioning could be achieved by analyzing them.

Given an SQL query containing a set of simple predicates accessing a relation \(R\), we define a set of all simple predicates and build a set of complicated predicates (involving logical OR and/or AND operations) to decompose the relation \(R\) into horizontally partitioned fragments.

Similar to the transaction–attribute matrix for vertical partitioning, we use a transaction–predicate matrix to represent the predicate access pattern of a set of given transactions. This matrix is used to define the primary horizontal partitioning model. Based on the transaction–predicate matrix, we identify and duplicate inter-cluster predicates to form clusters of transactions using the same set of predicates. Further, using the clusters and inter-cluster predicates, we can define the database partitions (i.e., fragments). We use an example in Zhang and Orlowska (1994) to illustrate our partitioning strategy. In this example, \(R = (\text{Eno, Ename, Sal, Degree})\) is a relation schema and there are seven transactions (T1–T7) using nine predicates over \(R\). These predicates are the following:

1: \(\text{Eno} < 10\)
2: \(\text{Eno} < 20\)
3: \(\text{Sal} \geq 50 \text{ K}\)
4: \(\text{Eno} \geq 20\)
5: \(30 < \text{Eno} < 60\)
6: \(\text{Degree} = \text{Ph.D.}\)
7: \(\text{Eno} < 15\)
8: \(\text{Eno} > 50\)
9: \(\text{Sal} < 50 \text{ K}\)

Their corresponding transactions are shown as follows:

1: \(\text{Eno} < 10, \text{Sal} \geq 50 \text{ K}, \text{Degree} = \text{Ph.D.}\)
2: \(\text{Eno} < 20, \text{Sal} \geq 50 \text{ K}\)
3: \(\text{Eno} \geq 20, \text{Sal} \geq 50 \text{ K}\)

\(^1\)In disjunctive normal form, multiple conjunctive predicates are connected with the disjunction operator. Similarly, in conjunctive normal form, multiple disjunctive predicates are connected with the conjunctive operator.

© 2010 The Authors.
International Transactions in Operational Research © 2010 International Federation of Operational Research Societies
4: $30 < \text{Eno} < 60$, $\text{Sal} < 50\, \text{K}$, Degree = Ph.D.
5: $\text{Eno} < 15$, $\text{Sal} < 50\, \text{K}$, Degree = Ph.D.
6: $\text{Eno} > 50$, $\text{Sal} < 50\, \text{K}$, Degree = Ph.D.
7: $\text{Eno} < 15$, $\text{Sal} \geq 50\, \text{K}$

The transaction–predicate access matrix shown in Fig. 19 is produced using this information. Note that transaction 1 uses predicates 1, 3 and 6. Hence, the corresponding matrix elements take on the value of “1”. With a transaction–predicate access matrix, the horizontal partitioning problem can be solved using the same algorithm. Two predicates are duplicated to decompose the matrix. The solution is shown in Fig. 20.

8. Conclusion

We consider vertical partitioning in the design of a distributed database. Vertical partitioning involves divided non-key attributes of a relational table into subtables. This division may be carefully designed to ensure performance. Cheng’s (1995) approach makes use of a branch-and-bound clustering approach. Although this approach is useful, it suffers from two pitfalls. First, the formulation of the vertical partitioning problem does not explicitly consider the quality of clusters formed. Second, the algorithm may generate infeasible solutions.

In this paper, we improve Cheng’s approach to solve the vertical partitioning problem. We reformulate the vertical partitioning problem. In our formulation, we develop the cohesion measure to measure the quality of clusters formed. The measure is defined to assume a value between 0 and 1, and hence, we can significantly reduce the number of parameters to test. In the design of the solution algorithm, we duplicate attributes instead of deleting attributes to cluster a
matrix. This strategy is effective in avoiding generating infeasible solutions with dangling transactions suffered by Cheng’s approach.

We also show how to use a transaction–predicate access matrix to represent the horizontal partitioning problem. The application of the access matrix makes the utilization of our algorithm in the horizontal partitioning problem possible.

In this work, we followed the literature and addressed the limitations of Cheng’s approach. Further, we provided implementation details of the algorithm for the problem. Another promising solution methodology may require formulating vertical partitioning as an integer-programming problem and make use of linear relaxation to trim down the size of a search tree. However, we feel that our approach is conceptually simpler. Practitioners without optimization background will find it easier to understand.

Acknowledgement

We would like to thank two anonymous reviewers for their insightful comments and suggestions. Their careful reviews have significantly improved the quality of this work. This research has been substantially supported by a grant (Project No. 414608) from Research Grants Council of Hong Kong SAR, China.

References


