Simultaneous Vertical Fragmentation and Segment Assignment in Distributed Data Bases using Genetic Algorithms

Angel Fernando Kuri-Morales¹, Edmundo Taddei-Zavala²

¹ Instituto Tecnológico Autónomo de México
Río Hondo No. 1, México D.F.
akuri@rhon.itam.mx
² Centro de Investigación en Computación
Instituto Politécnico Nacional, México D.F.

Abstract. Present technology favors the use and application of distributed data bases (DDB): those DBs in which information may be found in geographically distributed sites. The design of the distribution (the fragmentation of the relations and their assignment to the corresponding sites in the network) is one of the most complex tasks involved in the design of a DDB. Typically, it is not done automatically. In this work we discuss a genetic algorithm (GA) as applied to such design. The GA tackles, simultaneously, the problem of vertical segmentation and the assignment of the partitions to the sites in an optimal way. The evolutionary nature of the GA allows us to solve in parallel the fragmentation and assignment problems. These two problems display a highly non-linear relation between one another.

1 Introduction

The design of DDBs includes, besides the classical design problems of centralized DBs, the problem of the design of the distribution, whose aim is to improve the overall performance of the system. The design of the distribution is composed, in turn, of the fragmentation (also referred to as segmentation) and assignment of the segments to the diverse sites in the network. The segmentation is the process by which a global relationship is decomposed in horizontal and/or vertical segments. A vertical segment has to do with the grouping of data as a function of an attribute or a set of attributes, whereas the horizontal segments have to do with a grouping as a function of tuples or sets of tuples. Assignment, on the other hand, refers to the problem of placing the segments (be these vertical or horizontal) in a way such that access to the information is performed in the “best” possible way. Typically, the criteria which determine if segmentation and distribution are optimal are established independently. Hence, it is frequent that optimization is attempted in two stages. (We use the term “optimize” and its derivations to denote the search and possible determination of the best value or set of values in a finite space of possible solutions according to a measure of performance which is clearly specified.) In the first stage we search for the “best” segmentation and in the second the “best” placement of the
segments obtained in stage one [1]. The application of a GA allows, among other
things, the simultaneous optimization of segmentation and distribution attending to
the respective criteria without assuming linear independence (tacit in the method of
stages) between such criteria. In this article we report results which pertain to vertical
segmentation and assignment of such vertical fragments.

Vertical partitioning is the stage during which a relation is divided in sets of
attributes called vertical segments [2]. Vertical segmentation of the relations,
traditionally, is made under one of two perspectives. The first one of them is based on
the consideration that all attributes are initially placed in a unique segment and this is
segment is successively split in accordance with some (usually heuristic) criterion.
The other assumes that, initially, each attribute constitutes a segment and these are
grouped successively according to some other criterion (usually also a heuristic).
Formulated in this way the problem seems to be simple enough. However, the
methodology described has two important flaws. First: the number of possible
solutions grows exponentially with the number of attributes. For example, in a
relation with 10 attributes the number of disjoint sets (that is, the number of possible
vertical segments) is 115,975. In a relation with m attributes the number of ways in
which these may be grouped is defined by Bell’s number B(m). B(m) approaches mm
for a large enough m. Second: the algorithms used in practice (generically called best-
first algorithms) assume independence of the events considered. This assumption is,
in general, not valid. It is accepted for lack of better alternatives.

The purpose of the distribution is to improve the global behavior of the
system and, therefore, both segmentation and distribution have to be strongly
connected to the requirements of the system itself. That is, the decision of which
segments to form and where to place them has to be taken in accordance with the use
that different transactions induce on the attributes of the relation [3].

Here we attempt to find optimal segmentation and distribution considering
both goals to be inextricably related.

The rest of the paper is organized as follows: in section 2 we comment on
some related works; in section 3 we describe the GA used to solve the problem; in
section 4 we show the experimental results, in section 5 we make some considerations
regarding the theoretical soundness of the proposed method; finally, in the last section
(section 6) we make our concluding remarks.

2 Related Works

The problem of distribution in DDBs has been approached in several basically
independent ways. This problem has been described as the problem of file assignment
which establishes, as a premise, the prohibition of splitting the relations (in this case
treated as files) and, hence, the problem reduces to assigning full relations over the
sites in the network with the aim of minimizing the total cost of local and remote
accesses [4]. However, as established in the literature, this approach does not model
correctly the problem in DDBs since considering the relations as individual files does
not represent adequately the actual practical environment [5]. A detailed study of the problem of file assignment may be found in [6].

In other proposals it is assumed that segmentation has been performed as a step previous to the assignment of the fragments. The problem of segmentation has been described in detail in [1, 2].

In [7] it is assumed that relations may be segmented in a natural way in several blocks of a fixed number of sets of tuples and the problem resides in where to place these segments. To this effect the proposed algorithm keeps a counter of accesses in each one of the sites for each one of the blocks. The counter of accesses to a block is updated for each one of the transactions that makes an access to a tuple in the corresponding block. The block (or fragment) is assigned to the site corresponding to the largest number of registered hits. By looking to the counter periodically it is possible to dynamically reassign the segments as a response to changing patterns of access in time.

Ceri and Pelagatti [1] propose a set of general criteria for the assignment of fragments considering the difference of measuring or not the replication of data and a measure of cost and benefit of the assignment of segments is given in each case. Apers [5] proposes a similar segmentation scheme if a function of cost (such as the communication cost) is optimized. Further, he adds a random horizontal partition which allows greater parallelism if a function of performance (such as the average response time) is optimized. To minimize the total cost of transmission he proposes two ways to evaluate the function: one based on decision trees and the other in a greedy algorithm.

An algorithm of vertical partition of DDBs focused on maximizing the processing of local transactions is proposed in [8] and a graphic algorithm for the vertical partition is developed in [3]. In [9] it is considered that the attributes in a relation, when placed in a given node, will form the set of elements of the vertical fragment of the relation in such node. Besides, it is pointed out, it is possible to formulate a simple objective function which allows to assign and reassign the attributes of the relation in the nodes of the network in an optimal way by minimizing the costs of communication. An extension of the previous algorithm is formulated in [10] where to the aforementioned objective function we add the terms needed to consider the cost of access to a segment and the cost of storing a segment in a given site.

In [11] a two-phase approach is used. These consist of: 1) Forming clusters of segments and 2) Assigning the clusters to the nodes in the network. In the works of [12] and [13] an algorithm of vertical segmentation is proposed. In this case the purpose is to minimize remote accesses and maximize local accesses without considering replication. In other words, the goal is that every site is able to process transactions locally with a minimum number of hits to remote sites. The idea is to achieve a minimum cost of processing for any transaction regardless of where it is originated. [14] reports the results obtained by applying a genetic algorithm proposed by [12] and [13].
3. A Genetic Algorithm for the Simultaneous Vertical Fragmentation and Assignment in DDBs

The GA designed for the simultaneous vertical segmentation and assignment in DDB (which we shall call GASP) consists of the 5 components described in what follows.

3.1 Representation of the Solutions

When vertically segmenting we have $B(m)$ different ways to divide a relation consisting of $m$ attributes. Each one of these options has to be represented with a finite string over some finite alphabet [15], taking into consideration the characteristics of the problem at hand. In our case we must encode the attributes of a relation and the site to which they will be assigned. To this end, a vector of integers of length equal to the number of attributes of the relation will be used to represent possible solutions to the problem. This vector will hold integers between 1 and $N$, where $N$ denotes the number of sites in the network. In this way the following vector represents a possible solution if we consider a relation with 10 attributes and a network with 5 sites:

$$(5 2 4 3 1 3 3 5 2 4)$$

This proposal for a solution is to be interpreted as follows. The position in the vector indicates the attribute; the components of the vector indicate the sites. Therefore, the previous solution tells us that segmentation and assignment must obey what is shown in the following table:

<table>
<thead>
<tr>
<th>Site</th>
<th>Attributes in the Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2 9</td>
</tr>
<tr>
<td>3</td>
<td>4 6 7</td>
</tr>
<tr>
<td>4</td>
<td>3 10</td>
</tr>
<tr>
<td>5</td>
<td>1 8</td>
</tr>
</tbody>
</table>

This representation is adequate for this problem since the solution includes both the division of the relation in sub-relations as well as their assignment to a site of the network.

3.2 Initial Population

As opposed to traditional optimization techniques, which base the process of search of an optimum over points (that is, the search proceeds from one point to another) the GAs look for the solution passing from a set of points to another set. Every point is called and individual and the set of points is called the population. It is, therefore,
necessary to create an initial population to start a GA. This initial population was created randomly by selecting integer vectors as the one of section 3.1.

### 3.3 Evaluation Function

The evaluation function, also known as objective function, allows us to establish a classification of the individuals assigning to each a grade according to their merit. This figure of merit is called the fitness of the individual [16]. That is, the best behaved individuals get a better grade than those who behave poorly (as a possible solution to the problem in turn). The objective function is defined in terms of the optimization task desired. In this case the final purpose of segmentation and assignment is to minimize remote hits and maximize local ones. The fitness function proposed in [13] is

\[
EP(i) = E_M^2 + E_R^2
\]  

which evaluates the possible solutions via two terms. The first one calculates the cost of accessing local irrelevant attributes, while the second one calculates the cost of accessing relevant remote attributes. In order to do this, we use the Attribute Usage Matrix (AUM). The fitness function used in our case is

\[
\text{fitness}(i) = C - EP(i)
\]  

In this equation \( \text{fitness}(i) \) is the evaluation assigned to the proposed solution and \( C \) is a constant large enough to avoid negative fitnesses.

### 3.4 Genetic Operators

The GAs use a set of operators to modify the population from generation to generation. The present (i-th) population is altered to obtain the (i+1-st) population. Typical operators are selection, crossover and mutation although several others have been proposed [17].

In this algorithm we used proportional selection [18], 1-point crossover and uniform mutation. During selection individuals from the present population are selected randomly, according to the fitness obtained during evaluation, to form an intermediate population. Any individual may be selected more than once or not at all. From this intermediate population, also randomly, we select couples (with probability \( P_c \)) on which crossover will operate. This operation combines two parents to obtain two typically different offspring. This process is exemplified next. From section 3.1 we consider two possible solutions where (for clarity’s sake) all attributes are in one segment but in different sites:

\[
(1 1 1 1 1 1 1 1 1 1 )
\]

\[
(5 5 5 5 5 5 5 5 5 5)
\]
If the crossover point (obtained randomly) is “5”, when crossing the preceding individuals we get the following offspring:

\[
(1 \; 1 \; 1 \; 1 \; 5 \; 5 \; 5 \; 5 \; 5 \; 5)
\]

\[
(5 \; 5 \; 5 \; 5 \; 1 \; 1 \; 1 \; 1 \; 1 \; 1)
\]

During uniform mutation, on the other hand, one position in the vector is selected randomly with probability \(P_m\) and its value is altered randomly, as well. The expected number of positions to mutate during any given generation may be obtained multiplying \(P_c\) by the length of the individual and by the population’s size. That is:

\[
\text{Number of positions to mutate} = P_m \cdot l \cdot P_{size}
\] (3)

A mutation is performed selecting a random position in a vector of the population (selected randomly). For instance, taking the first individual illustrated in the preceding paragraphs and assuming the chosen position was 3 we have, after mutating this position:

\[
(1 \; 1 \; 4 \; 1 \; 1 \; 1 \; 1 \; 1 \; 1 \; 1)
\]

### 3.5 Parameters for the Genetic Algorithm

To completely specify the GA we need to assign values to several parameters. Such an assignment of the GA’s parameters values is not trivial. It cannot be affirmed that there is a unique best set of values. There are, however, some general recommendations in this sense. The interested reader may see, for instance [19].

We used the following values:
- Population size = 100
- \(P_c = 0.85\)
- \(P_m = 0.005\)
- Number of generations = 100
- \(C = 65000\)

### 3.6 Elitism

A last, but basic, issue has to do with the fact that the best individual of population \(i\) is kept as an element of population \(i+1\). It is only eliminated if all of the other individuals display a better fitness. This strategy is called “elitism”; a GA which follows it is called “elitist”. As will be discussed in section 5, its allows us to generalize our results in the sense that (as will be shown there) by making our algorithm elitist we are able to claim that global convergence regardless of the problem being tackled.
4 Experimental Results

To evaluate GASP’s performance we solved the following problem. This problem is discussed and solved by exhaustive enumeration in [13]. An exhaustive search was performed and it was determined that the minimum value for the function is 5820. Let R be a relation with 10 attributes and 8 transactions which are performed on a DDB. Considering there are 10 sites where the resulting fragments may be stored and a usage matrix as shown in Table 2 (“T” stands for “Transactions”). The problem, rather simply stated, is to determine the optimum partition scheme.

<table>
<thead>
<tr>
<th>T</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25 0 0 0 25 0 25 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 50 50 0 0 0 50 50 0</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 25 0 25 0 0 0 25</td>
</tr>
<tr>
<td>4</td>
<td>0 35 0 0 0 35 35 0 0 0</td>
</tr>
<tr>
<td>5</td>
<td>25 25 25 0 25 0 25 25 25 0</td>
</tr>
<tr>
<td>6</td>
<td>25 0 0 25 0 0 0 0 0 0</td>
</tr>
<tr>
<td>7</td>
<td>0 0 25 0 0 0 0 0 25 0</td>
</tr>
<tr>
<td>8</td>
<td>0 0 15 15 0 15 0 0 15 15</td>
</tr>
</tbody>
</table>

4.1 Experiments

The first experiment consisted in generating a random initial population from which to start GASP; then measure the results. We ran the algorithm 100 times. We found that the minimum average was 5942 and that the minimum (best value) was reached 75 times out of the 100 total. This may be seen in figures 1 and 2.
A second experiment was aimed at determining whether a directed (non-random) initial population would outperform the first method. Therefore, the initial population was created so that all attributes were contained in a single fragment, i.e., a single site keeps all the attributes. GASP reduced its performance since the minimum average passed to 5978 and the best value was only found 71 times.

A third experiment considered the generation of the initial population so that each of the attributes would be contained in a different site. The performance of GASP was worse than in the two preceding cases. The average minimum jumped to 6049. Furthermore, the best value was only obtained 59 times in 100 runs. An
interesting fact was that when the number of available sites is reduced the behavior of GASP improves markedly, as shown in figures 3 and 4.

**Fig. 3.** Behavior of average fitness for 8 available sites.

**Fig. 4.** Number of best values found per generation.

As displayed in the preceding figures, the average fitness value decreased to 5889 when the number of sites was set to 8 and the number of times it found the best value passed to 85. A similar behavior was observed when the number of sites was set to 6. In that case the average fitness passed to 5684 and the number of times the best value was found was 88.
5. Theoretical Considerations

We have chosen to test our method on a relatively simple problem because more complex ones are not amenable to an exhaustive analysis: the computational complexity is too large. As far as GASP is concerned this poses no problem. But we would lack the yardstick we are looking for: a concrete way to experimentally validate the behavior of GASP. Therefore, for this interesting but simple (and hence computationally tractable) problem we have shown that the results when using a GA are satisfactory even in the worst case. The obvious question is whether these results should be expected for more complex problems.

In what follows we briefly include a discussion that theoretically shows that this is, indeed, the case. It assumes that the representation of the solution is binary in nature and, although our algorithm uses a non-binary one, it is simple to see that any non-binary representation has a binary counterpart. Although the mechanics of the process may vary (which is, in fact, the reason why we selected to represent our vectors as integers: convergence is faster) all the conclusions from the following analysis apply to the described GA.

5.1 Markov Chain Models

The GA we described (albeit without elitism) is typically referred to as the Simple Genetic Algorithm or SGA. Here we approach an analysis of the behavior of the SGAs by modeling it as finite Markov chains. Our treatment follows the one by Rudolph [20].

Markov chains are stochastic processes in which the probability that the process will be in state $j$ at time $t$ depends only on the state $i$ at time $t-1$. A "state" of a finite-population GA is simply a particular finite population. The set of all states is the set of all possible populations of size $n$. These can be enumerated in some canonical order and indexed by $i$. We may represent the $i$-th such population as a vector of length $2l$. The $y$-th element of is the number of occurrences of string $y$ in population $P_i$. Under an SGA the current population $P_j$ depends only on the population at the previous generation. Therefore, the GA may be modeled as a Markov chain.

The set of all possible populations of size $n$ can be represented by a matrix in which the columns are all possible population vectors $i$. There are populations of size $n$.

To illustrate the case, let us say that $n = 2$ and $l = 2$. The possible populations are

$$
P_0 = \begin{pmatrix} 00 \\ 00 \end{pmatrix}, \quad P_1 = \begin{pmatrix} 00 \\ 01 \end{pmatrix}, \quad P_2 = \begin{pmatrix} 00 \\ 10 \end{pmatrix}, \quad P_3 = \begin{pmatrix} 00 \\ 11 \end{pmatrix}, \quad P_4 = \begin{pmatrix} 01 \\ 01 \end{pmatrix}, \quad P_5 = \begin{pmatrix} 01 \\ 10 \end{pmatrix}, \quad P_6 = \begin{pmatrix} 01 \\ 11 \end{pmatrix}, \quad P_7 = \begin{pmatrix} 10 \\ 10 \end{pmatrix}, \quad P_8 = \begin{pmatrix} 10 \\ 11 \end{pmatrix}, \quad P_9 = \begin{pmatrix} 11 \\ 11 \end{pmatrix}\$$
The matrix $\Xi$ is, therefore,

$$
\Xi = \begin{pmatrix}
2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 2 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 2
\end{pmatrix}
$$

A state for the Markov chain corresponds to a column of $\Xi$. Now we can set up a transition matrix $P$. $P$ is an $N \times N$ matrix, and each element $P_{ij}$ is the probability that population $P_j$ will be produced from population $P_i$ under an SGA.

A finite Markov chain describes a probabilistic trajectory over a finite space $S$ of cardinality $|S| = n$, where the states may be numbered from 1 to $n$. The probability $p_{ij}(t)$ of passing from state $i \in S$ to state $j \in S$ at step $t$ is called the transition probability from $i$ to $j$ at step $t$. If the transition probabilities are independent from $t$, i.e., $p_{ij}(t) = p_{ij}(s)$ for all $i, j \in S$ and for all $s, t \in N$, the Markov chain is said to be homogeneous.

The transition probabilities of a homogeneous finite Markov chain can be gathered in the transition matrix $P = (p_{ij})$. If for each entry, $p_{ij}, i \in \{0, 1\}$ and $\sum_{i=1}^{|S|} p_{ij} = 1$ for all $i \in S$ a matrix is called stochastic. Given an initial configuration $\rho^0$ as a row vector, the configuration of the chain after the $t$-th step is determined by $\rho^t = \rho^0 P^t$. Therefore, a homogeneous finite Markov chain is completely determined by the pair $(\rho^0, P)$.

### 5.2 Markov Chain Analysis of a Genetic Algorithm

The SGA can be described as a Markov chain. The state of the SGA depends only on the genes of the individuals so that the state space is $\{0, 1\}^n$ where $n$ denotes the population size and 1 is the number of genes. Each element of the state space can be regarded as an integer number in binary representation. The probabilistic changes of the genes within the population caused by the genetic operators are captured by the transition matrix $P$, which can be decomposed into a product of stochastic matrices $P = C M S$, where $C$, $M$ and $S$ describe the intermediate transitions caused by crossover, mutation and selection, respectively. This leads to the following theorems, which we state without proof (but the interested reader may see [20]).

**Theorem 1**

The transition matrix of the SGA with mutation probability is regular.

**Corollary**
The SGA as defined in theorem 1 is an ergodic Markov chain. That is, there exists a unique limit distribution for the states of the chain with nonzero probability to be in any state at any time regardless of the initial distribution. This implies that the initial distribution has no effect on the limit behavior of the Markov chain. The initialization of the algorithm can be done arbitrarily. This explains, in part, the results of the experiments above.

Theorem 2

The Simple Genetic Algorithm does not converge to a global optimum.

When working with GAs the logical practice is to keep track of the best individual, so that one might argue that Markov chains do not represent a practical GA. In fact, after a finite number of transitions, the global solution will be visited and copied. This is a consequence of the following:

Theorem 3

In an ergodic Markov chain the expected transition time between an initial state i and any other state j is finite regardless of the states i and j.

Therefore, if we retain the best solution found up to time t all along the process we may see that:

Theorem 4

The SGA maintaining the best solution found over time after selection converges to the global optimum.

We may also state a complementary theorem:

Theorem 5

The SGA maintaining the best solution found over time before selection converges to a global optimum.

Note that theorems 4 and 5 do not cover the case of elitist selection. When using elitist selection (as in the case reported above) the best individual is not only maintained but also utilized to generate new individuals. This algorithm has another transition matrix and, therefore, different search dynamics which may be better in some cases and worse in other cases. Clearly, however, it converges to a global optimum.

In the case of a canonical SGA global convergence is not guaranteed. The reason is quite clear: for an SGA there is a minimal probability bounded from zero to lose the global optimum solution at each generation. It follows from the Borel-Cantelli Lemma that this event will occur with probability one. On the other hand, there is a minimal probability to find again a global solution if it was lost, so that this
event will also occur with probability one. In fact, the global best solution will be lost and found infinitely often so that the sequence of best individuals found over time in the SGA is an irreducible Markov chain on the state space \{0, 1, \ldots, n\} which does not converge although the expectation does.

5.2 Remarks

The previous analysis shows that global convergence is not an inherent property of an SGA. In other words, the original SGA cannot be regarded as an optimization algorithm for the static optimization problems because it is provable that it will not converge to any subset of states containing at least one global solution, even in infinite time. The fundamental theorem of GAs (for which see [21], pp. 177) does not imply that the SGA will converge to the global optimum in static optimization problems. Moreover, due to the irreducibility of the SGA it is clear that the SGA will not converge at all.

It also shows, however, that by the simple expedient of maintaining the best observed individual we may guarantee such global convergence. This is a very strong result which gives solid foundation to the claim that the results reported herein may be considered general in nature and not just a casuistic example.

6 Conclusions

All along we have argued that the problem of optimizing segmentation and segment assignment over the network’s sites is a very difficult problem. However, we also showed that if a genetic algorithm is used we can obtain very good results with a low number of evaluations of the objective function. In the problem considered, if an exhaustive search is performed, we are required to evaluate 115,975 different partition schemes. If the number of attributes in the relation is increased such an approach quickly becomes impractical. For this problem GASP has shown that with a maximum of 10,000 evaluations (that is, 8.6% of the total) we find an optimum value in the vast majority of the cases. And from figure 1 we may see that by generation 75 we have already found a value quite close to the best. This means that with approximately 6% of the evaluations of the possible total, we find very good results. The efficiency is even better when the number of sites where segments may be stored is reduced, as was also shown.

Two further points should be mentioned. First, that although more complex problems imply exponentially more alternatives to test (from Bell’s number we know these to approach \(m^n\)) the efficiency of the GA will roughly remain as reported above because of the so-called implicit parallelism inherent to this kind of algorithms (for which the interested reader may see [15]). Second, that the number of sites in most practical applications remains small and, therefore, the indices reported above bear a significant practical importance.

On the other hand, our experiments are a confirmation of the non-intuitive fact that artificially introducing plausible heuristics to the generation of the initial population \(P_0\) yields no improvement on the convergence of the algorithm. This is
explained by simply noting that when the initial population is “hand-picked” the
search space is importantly restrained. However, the improper choice of population P₀
has bearing on the algorithm’s efficiency though not in its efficacy; as mentioned
above, the GA will still converge.

The GA has shown to be a plausible alternative to the solution of
segmentation problems in DDBs in which the problem of simultaneously looking for
the segments and their assignment is tackled. We should keep in mind that, in this
approach, it is possible to incorporate the non-linear relationships that exist between
these two goals in the objective function. No restrictions are imposed on the form of
the solution.

Finally, we point out that the method may be applied practically without
changes to the most general problem: the vertical and horizontal segmentation and the
assignment of such segments. In future works we will report results obtained from
such mixed segmentation.

References

5. Apears Peter M. G. Data Allocation in Distributed Database Systems. ACM
7. Brunstrom Anna, Scott T. Leutenegger and Rahul Simha. Experimental Evaluation of
   Dynamic Data Allocation Strategies in a Distributed Database With Changing
   Database Design. ACM Transactions on Database Systems. Vol. 9. No. 4, December
   1984.
   Vertical Fragmentation and Dynamic Reassignment in Distributed Database. 1997.
10. Pérez, Joaquin, David Romero, Rodolfo Pazos, Juan Frausto y Laura Cruz.
    Integration of Vertical Fragmentation and Assignment in Distributed Database using
    the Threshold Compliance Algorithm. 1999.
11. Shepherd J., B. Harangsri, H. Ling Chen, A. Ngu. A Two-Phase Approach to Data
    Allocation in Distributed Database.
    Function for Vertically Partitioning Relations in Distributed Database and its Analysis.


