A Novel Vertical Partition Approach: Group-Oriented Restricted Growth String Based Genetic Algorithm

Jun Du, Reda Alhajj and Ken Barker
Advanced Database Systems and Applications Lab
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
{jundu,barker,alhajj}@cpsc.ucalgary.ca

Abstract –**Vertical partitioning is an optimization problem that can resort to genetic algorithms (GA). However, the performance of the classical GA application to vertical partitioning as well as to similar problems such as clustering and grouping suffers from two major drawbacks – redundant encoding and non-group oriented genetic operations. This paper applies the restricted growth (RG) string [14] constraint to manipulate the chromosomes so that redundant chromosomes are excluded during the GA process. On RG string compliant chromosomes, the group oriented crossover and mutation becomes realizable. We thus propose a novel approach called Group oriented Restricted Growth String GA (GRGS-GA) which incorporates the two above features. Finally, we compare the proposed approach with a rudimental RG string based approach and a classical GA approach. The conducted experiments demonstrate a significant improvement of GRGS-GA on partitioning speed and result.**

Keywords: Data design, RG string, Group oriented genetic operators, Vertical Partition Problem

I. INTRODUCTION

Vertical partition is the problem of clustering attributes of a relation into fragments for subsequent allocation. The technique is used to minimize the execution time of user applications that run on these fragments. Vertical partition provides an important technique for designing distributed database systems.

Vertical partition algorithms contain two essential parts: the optimization method and the objective function. Özsu and Valduriez [11] argue that finding the best partition scheme for a relation with m attributes by exhaustive search must compare at least the m^{th} Bell number of possible fragments, which means that such an algorithm has a complexity of $O(m^m)$. Thus, it is more feasible to look for heuristic methods to seek optimal solutions. On the other hand, database partition aims at enhancing the transactional processing in database. The objective function evaluates whether such a goal is achieved. Almost all previous algorithms have these two discernable elements. However, the graphical approach developed by Navathe and Ra [13] is an exception; it does not have an explicit objective function.

Most previous algorithms employ multiple iterations of binary partition to approximate m-way partition. Navathe *et al* [9] propose the Recursive Binary Partition Algorithm (RBPA), which extends Hoffer and Severance's work [6] by automating the selection process of vertical fragments; they propose some empirical objective functions. Cornell and Yu [3] adopt the same approach but replace the empirical objective functions with one constructed on a model database. Chu and Ieong [18] adopt transactions as units in their algorithm; however, it is still a binary partition approach.

Efforts have also been made to use other optimization techniques to benefit vertical partition. Hammer and Niamir [5] propose a hill-climbing method that alternatively groups and *regroups* attributes and fragments to reach a suboptimal solution. Song and Gorla [16] solve the problem with GA. However, each run of their GA only gets a binary partition. Therefore, the GA only provides the intermediate results in a recursive process. Our aim in this paper is to propose a *pure* GA solution to vertical partition, i.e., the direct result from the GA execution is already an m-way partition.

Falkenaur [4] stated that most problems can be solved with either the classic Holland-style GA [6] or the ordering GA [15]. However, these two types of GAs do not work as efficiently with the grouping problem.

According to Falkenaur's definition of the grouping problem, the vertical partition problem is a similar one. Both kinds aim at "partition a set U of objects into a collection of mutually disjoint subsets u_i of U..." and "optimizing a cost function defined over the set of all valid groupings" [4]. However, most grouping problem inherited some "hard constraints", thus facing reduced search space, while vertical partition usually treats all grouping combinations feasible solutions. Also, the objective function of the vertical partition problem is more complicated than the grouping problem examples illustrated by Falkenaur. Another similar problem is the high dimensional clustering problem whose objective is to minimize inter-cluster dissimilarities and maximize intra-cluster dissimilarities.

The first major difficulty previous GAs had in solving the grouping problem is that one solution can be encoded into several different chromosomes. The mapping between the real solution space and the chromosomes representing those solutions is not a one-to-one relationship. Such a highly redundant encoding system "is a major blow to the efficiency of a GA" [4].

The second serious drawback of most GAs designed for grouping problems is that the GA operators are conducted

on objects rather than groups. GGA proposed by Falkenaur [4] succeeds in utilizing group oriented GA operators. However, it requires an encoding completely different from the conventional encoding scheme. This encoding scheme represents the solution in length variable strings, including two parts – the fixed length object part and the variable length group part. The crossover and mutation operations are only applied on the group part, but after each crossover or mutation operation, the content of the object part needs to adjust accordingly.

The work described in this paper considers the vertical partition problem and reports a GA that can eliminate the encoding redundancy by using a *restricted growth* (RG) string [14] constraint in constructing chromosomes and performing group oriented GA operations. This proposal is novel GA based approach called Group oriented Restricted Growth String GA (GRGS-GA). To evaluate its effectiveness and demonstrate its superiority, we compare the result of using GRGS-GA with that of using traditional GA as well as RG string encoding with traditional object based GA operators.

The balance of the paper is structured as follows. Section II introduce the partition evaluator (PE) developed by Chakravarthy *et al* [2]; this evaluator will be used as the fitness function for the GAs described in Section III. Three GAs are covered in Section III; in particular, we concentrate on RG string encoding and the new group oriented GA operators. Section IV reports experimental results, and it is demonstrated that GRGS-GA can effectively find optimal solution for vertical partition problems. Section V is summary and conclusions.

II. OBJECTIVE FUNCTIONS FOR VERTICAL PARTITION

The two kinds of objective functions used for partition algorithms are model cost functions based on the transaction access analysis on a model DBMS and those based on an empirical assumption. The former form of objective function is specific to the underlying DBMS while the latter is more general and intuitive.

In addition to the AUM used as input for both types of objective functions, the model cost function takes into account the specific access plan chosen by the query optimizer, *e.g.*, the join method and the type of scan on the relation by each transaction type. Without this additional information, the empirical cost objective function only shows the trends in the cost that are affected by the partition process. However, it is useful for the logical design of a database when information about physical parameters may not be available. Although less precise than the model cost functions, they can be very effective in comparing different optimization techniques used by algorithms.

In this paper, we use an empirical objective function, a modified version from the partition evaluator proposed by Carkravarthy *et al.* [2]. This partition evaluator uses the square-error criterion commonly applied to clustering strategies. We thus name it the Square-Error Partition Evaluator (SEPE).

The SEPE consists of two major cost factors: the *irrelevant local attribute access cost* and the *relevant remote attribute access cost.* They represent the additional cost required, other than the ideal minimum cost. Further, the *ideal* cost is the cost when transactions only access the attributes in a single fragment and have no instances of irrelevant attributes in that fragment. Both costs are calculated using the square-error result; they are denoted E_M^2 and E_R^2, respectively. More details about the SEPE, including the formula, can be found in [2].

III. GENETIC ALGORITHMS FOR VERTICAL PARTITION

A typical GA application to the vertical partition problem is presented in Section A; its encoding system is straightforward. Each step involved in the GA process is described in detail. Restricted strings (RG) based GA is covered in Section B. The proposed GRGS GA is described in Section C.

A. RD-GA

The first GA we call RanDom integer string GA (RD-GA); it is the most straight forward GA and the easiest to implement.

The partition scheme is encoded as an array of integers. Every attribute matches an element in the array, and the value of the element is the attribute membership, the number of the fragment to which the attribute belongs. For example, if the maximum fragment number is five, then an array (2, 3, 1, 0, 0, 4) defined for a collection of attributes $\{a_0, a_1, a_2, a_3, a_4, a_5\}$ represents the partition scheme of $\{\{a_0\}, \{a_1\}, \{a_2\}, \{a_3, a_4\}, \{a_5\}\}$. The integer array does not need to include all the five numbers. Thus (2, 2, 0, 0, 0, 4) is also a valid representation, which describes a partition scheme of $(\{a_0, a_1\}, \{a_2, a_3, a_4\}, \{a_5\})$. This feature facilitates flexibility to represent arbitrary schemes as long as the number of fragments is not greater than a predefined number (i.e., 4, in this case). However, it is noteworthy that the actual codification is orthogonal to the fragmentation; for example, (1, 1, 4, 4, 4, 3) is different from (2, 2, 0, 0, 0, 4) as integer arrays, but both produce the same partition scheme. The potential impact is that more redundant evaluations and comparisons will occur.

We assign a random number for each attribute. Each random number ranges from 1 to the maximum order of the attributes.

The selection can be implemented using either Roulette or Tournament selection combined with Elitism. Tournament selection randomly selects n chromosomes in the base population and chooses the fittest one for the next generation, thereby eliminating the weakest few chromosomes in each generation. In our experiment, we always set n to two.

The uniform crossover method is selected in our experiment since later we will find that the RGRS GA crossover is a variation of uniform crossover. To fairly compare the three GAs, we also implement the RGS-GA

(See Section III.A) using uniform crossover. However, in the crossover procedure a special check is added to ensure that no two identical chromosomes would conduct crossover operations. If two identical chromosomes are chosen to crossover, one of them gets mutated and skips the crossover procedure. Our experiments show that this strategy is very useful. The method offers significant performance improvement in every case.

We implemented the one point mutation method, i.e., only one gene in a chromosome is altered in each mutation. In the context of vertical partition, a mutation means the membership of an attribute is changed.

The GA is terminated when there is no improvement in the PE values over a number of generations. We define a parameter in our GAs, called termination number n. Whenever n generations have passed, the GA will test if there is any improvement over the previous n generations.

B. RGS-GA

The main difference between RGS-GA and RD-GA lies in the encoding schemes they use. From the new encoding scheme, we derive a novel way to randomly initialize the chromosome population.

The encoding system has a drawback: it allows numerous chromosomes to represent the same partition scheme. Recall that for a relation with m attributes, the number of distinct solutions is the m^{th} Bell number B(m), while the encoding system can produce m^m different chromosomes; $m^m > B(m)$. The redundancy rate $m^m/B(m)$ grows as m increases. This highly redundant encoding deviates from the principle of minimal redundancy, which is one of the several design principles for constructing useful representations defined by Radcliffe [12].

Rusk [14] describes the set partition problem and presents a way to travel through all of the possible partition schemes without redundancy. Every partition can be represented with a RG string (see Definition 1) and two different RG strings must correspond to two different partitions. For example, the set [4] is a collection of numbers: {1, 2, 3, 4}. One of its partitions is {{1}, {2, 3}, {4}}, which can be represented by the RG string (0112).

Our new encoding system regulates each chromosome so it must be a RG string thereby eliminating redundant representations of the partition solutions. To differentiate it from the traditional encoding scheme used in Section III.A, the latter is called the random *string encoding* and this is the *RG string encoding*.

As in the random string encoding GA, the RG string encoding represents a grouping solution as an array of integers, denoted $a[n]$, where n is the number of attributes in the relation. The elements in the array may be integer values ranging from 1 to n. Meanwhile, they must satisfy Definition 1, given next. In addition to the formal definition of RG string, other supporting extended definitions and theorems required for this section are presented next:

Definition 1: An RG *string* r is a sequence of integers represented as an array, which satisfies the following inequality:

$r[i] \le 1 + \max\{r[0], r[1], ..., r[i-1]\}$, $0 < i < n$, $r[0] = 1$

For example, {1 1 2 3 1 1 2 4} is a RG string, but {4 4 2 3 4 4 2 1} is not, although they map to the same solution in random string *encoding* scheme.

Definition 2: The *degree of a RG string* r is the largest value in r, denoted $d(r)$. ■

For example, consider $r = \{11123221\}$, then $d(r) = 3$.

Definition 3: The i^{th} prefix *of a RG string* r, denoted p_r^i, is the substring that includes the first i values of r. ■

For example, consider $r = \{11123221\}$, then $p_r^4 = \{1112\}$.

Definition 4: *High potential degree* $hd(p_r^i)$ for *position* i in a RG string r is the highest possible degree for the i^{th} prefix, inclusive. ■

For example, consider $r = \{11123221\}$, then $hd(p_r^5 = 3)$.

Three operations in GAs can change the constitution of a chromosome in the population: initialization, crossover, and mutation. Any unrestricted changes in a chromosome could result in a violation of the RG string constraint. Example 1 illustrates the violation.

Example 1: Consider two RG strings $r1 = \{11123213\}$ and $r2 = \{12134231\}$. Assume a one point crossover occurs after position 4. The two chromosomes are changed to {11124231} and {12133213}; but the first chromosome is not a RG string. Similarly, a one point mutation on *r1* at position 4 may produce a non RG string compliant chromosome {11133213}. ■

To avoid breaking the constraint, one might add rules so that in Example 1 the mutation of *r1* at position 4 is disallowed because changing the element to any number other than 2 will cause a violation of the RG string constraint. However, overly restricting crossover and mutation will limit offspring variety. The simplest solution turns out to be the most powerful: after each operation we use a rectifier to ensure that every string is a RG string

We have designed two different chromosome constructors for the initialization process in the RG string type GAs. They are the Sparse Constructor (SC) and the Dense Constructor (DC). The SC is based on the constructor we used in the RD-GA. After a random integer string is generated, it is converted to the RG string with the rectifier. When the DC generates chromosomes, the RG constraint is enforced from the beginning. No rectification process is needed after all genes in a chromosome are randomly created because each gene is created as a random integer between 1 and the high potential degree for its position, complying with the RG string constraint.

These two constructors get their names because the SC should typically be able to create a partition that has more fragments than one created by the DC. Both constructors use a random function to create an integer. However, in SC, each element can range from 1 to n, where n is fixed and equal to the maximum number of fragments anticipated by the user. In DC, the first element is set to 1 and the upper bound for each element increases gradually, which rarely reaches n-1.

The mutation should have the ability to move one attribute from its current fragment to another fragment or

to a new fragment containing just the attribute. In order to achieve this, we could define the mutation operation as altering one attribute's membership a[*i*] based on a random number. Usually it is a random function of the number of fragments in the current partition (i.e. its degree). The simplest mutation operator generates a random membership ranging from 1 to the degree of the string plus a constant ε, denoted random (degree $+\varepsilon$), where ε is a number greater than or equal to 1, depending on how much the mutation should increase the number of fragments.

For example, the RG string {0 0 1 2 0 1 3} will undergo a mutation in its 4th position. The original gene is 2 and the degree of the RG string is 3. The new gene could be a value from 0 to 4 if the statement mutation operator random (*degree* + 1) is used. The chance of having the value 4, namely, the probability of increasing the number of fragments, is 0.2. If mutation operator random (*degree* + 6) [1] is used, the new gene could be a value from 0 to 9, which increases the probability to 0.5.

In some experiments, we found the following mutation operators are also effective for certain cases: $\text{random}(\varepsilon * \text{degree})$ *or* $\text{random}(\text{degree} * \text{degree}/\varepsilon + 1)$.

C. GRGS-GA

Group oriented Restricted Growth String GA, GRGS-GA, brings an improvement over RGS-GA by using group oriented GA operators.

Falkenauer [19] first proposed GGA to solve the NP-hard grouping problem bin packing problem. He recognized that traditional GA operators take wrongly the objects as the building blocks in solving the grouping problem. He thus suggested using crossover, mutation and inversion operations that are more context sensitive and less schemata disruptive. Although he identifies the redundant problem in the classical encoding scheme, his length variable encoding scheme still allows redundancy. So, our contribution is to implement group oriented operators in a fix length encoding scheme that totally deprive encoding redundancy.

The group oriented crossover on two parent chromosomes generally goes through the following steps:

1. Unify group ids by adjusting group ids of one of the parents so that the two parents use unique group ids.
2. Select at random a number of group ids for each parent.
3. Inject into the first parent the attributes of group ids selected in the second parent. And record groups in the first parent that has been modified; we call them chopped groups.
4. Apply special strategies to regroup attributes in the chopped groups to get the first offspring.
5. Repeat steps 3 and 4 on the two parents with inversed roles to get the second offspring.

[1] Although during initialization, the number for each array element is confined to be less than the number of attributes, during mutation, an exception is allowed because the rectification corrects it.

In step 4, the attributes in the chopped groups need to be re-assigned according to certain strategies. Unlike the grouping problem analyzed by Falkenauer, the vertical partition problem usually has no hard constraints. It is almost impossible to use heuristics to reduce the search space. For the vertical partition problem, we only designed the following two simple strategies to deal with the attributes in chopped groups.

- *Union:* This strategy simply includes all attributes in the chopped groups in a new fragment. It must be complemented with a high crossover rate (i.e., the portion of attributes in the parents that will be affected by crossover) – so that only few attributes need to be reassigned, and a mutation operator called splitting mutation, which will be introduced soon, is applied. The potential benefit of this strategy is revealed when the optimal partition has a small number of fragments.
- *Binary Merge:* This strategy maintains a list of chopped group ids. Then two ids are selected at random from the list. Attributes within these two ids are merged into one group. The advantage of such method is that the partition size would not be affected too much between parent chromosomes and offspring chromosomes. Compared to the union method, it is less sensitive to crossover rate or the mutation operators to be used. Example 2 illustrates how two 10 attributes partition schemes undergo the group oriented crossover operation.

Example 2 Let $r1=\{1221341324\}$ and $r2=\{1122341343\}$ represent the following two partition: {{1, 4, 7}, {2, 3, 9}, {5, 8}, {6, 10}} and {{1, 2, 7}, {3, 4}, {5, 8, 10}, {6, 9}}

1. Add 4 to each group id in r2 to get $r2'=\{5566785787\}$
2. Select at random group ids 4, 1 for r1 and 8 for r2'.
3. Inject group 8 of r2' to r1 to get $r1'=\{1221381384\}$ and record group 2 and 4 in r1 as chopped groups.
4. Change the group id 2 and 4 in r1 to 9: $r1''=\{1991381389\}$, and rectified it into the RG string $r1''=\{1221341342\}$ representing {{1, 4, 7},{2, 3, 10},{5, 8},{6, 9}}. Thus, one offspring is obtained.
5. Inject group 4 and 1 of r1 to r2". $r2''=\{1561741784\}$. The chopped group ids are 5, 6, 8, and 7. By binary merging the attributes as described above, we get $r2''=\{19a19419a4\}$, where a stands for group id 10. After rectifying $r2=\{1231241234\}$, we have another offspring partition: {{1, 4, 7}, {2, 5, 8}, {3, 9}, {6, 10}} ■

	Partition 1	Partition 2
Parents	{{1,4,7}, {2,3,9}, {5,8}, {6,10}}	{{1,2,7}, {3,4}, {5,8,1}, {6,9}}
GRGS-GA offspring	{{1,4,7}, {2,5,8}, {3,9},{6,10}}	{{1,4,7}, {2,3,10}, {5}, {6,9}}
RGS-GA offspring	{{1,7}, {2,3,4,9}, {5,7,10},{6}}	{{1,2,4,7}, {3}, {5,8}, {6,9,10}}

Fig. 1. Crossover comparison of RGS and GRGS

Apply uniform crossover adopted by RGS-GA to the same two RG strings r1 and r2, given the uniform mask as {0001010111}, where 1s indicate that genes at those positions are exchanged between parents. So the offspring would be {{1, 2, 4, 7}, {3}, {5, 7}, {6, 9, 10}} and {{1,

7}, {2, 3, 4, 8}, {5, 8, 10}, {6}}. From Figure 1, it can be easily recognized that GRGS-GA offspring inherits more characteristics from its parents while RGS-GA crossover could be very destructive.

In GGA, the mutation strategies follow three directions: creating a new group, eliminating an existing group, and shuffling a small number of objects between groups.

In our GRGS, we constructed four candidate mutation operators.

- *Merging*: Two group ids are first selected at random, and the attributes in the two groups are merged into one group.
- *Jumping*: The jumping operator is in fact the mutation operator we used for RGS-GA. It allows one attribute to change its membership in order to belong to a different group. Also at a certain probability, it can form a new group on its own. This feature is desirable when most partition schemes in a population are fine-grained grouping, thus the resulting chromosome with a single-attribute fragment has higher chance to survive.
- *Splitting*: This operator first decides on a group id at random, then, splits this group into two smaller groups. It is mostly useful when combined with the union type crossover.
- *Swapping*: The swapping operation works on two randomly selected groups. The swapping rate decides on the proportion of attributes in groups to be exchanged in each mutation. In our experiments, the swapping mutation is found not as effective as the jumping mutation.

The inversion operation used in GGA is not applicable in our encoding scheme. In GGA, the inversion is used to convert between "redundant" chromosomes. These chromosomes although represent the same solution, they are considered to bear order sensitive information. For the vertical partition problem, the order of the groups in a partition is irrelevant. We thus do not need the inversion operation at all.

IV. EXPERIMENTS

In this section, the three different GAs covered in Section III are used and compared for a 20-attribute usage matrix widely discussed in the literature on the vertical partition problem.

A. GA Configuration

A Key to GA success is an optimal configuration. Good GA search performance is only possible if care is taken to configure the algorithm appropriately. GA theory provides little guidance for proper selection of the configuration parameter settings. Although several papers describe optimization for GA configuration, the refined tuning of GA is a time consuming trial-and-error one that is unavoidable if the algorithm is to be successful.

The configuration used in the case studies is based on our experiments and the recommendations commonly found in the literature. These parameters are not necessarily the "best" for all the examples discussed below but facilitate the comparison among the three GAs. The following presents the parameters used for the trail runs conducted in the experiments: Population size: 200; Mutation rate: 0.01; Selection mechanism: 2-competitor tournament; and Elite copies: 8.

Since in GRGS-GA, uniform crossover is easier to implement than one-point crossover or two-point crossover, we choose to implement uniform crossover in RD-GA and RGS-GA, as well in order to be fair in the comparison process. The mutation operator is the simple one point mutation in RD-GA. In RGS-GA, we mutate a chromosome by assigning a randomly selected gene to a random membership other than its old one. The random number ranges from 1 to the number of genes in this chromosome plus 1. Among the four types of mutation operators designed for GRGS-GA, we alternatively use the merge and the jump mutation at a ratio of 1:2.

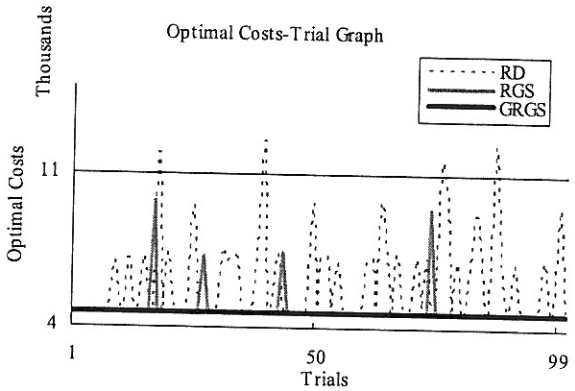

Fig. 2. Optimal costs-trial graph for 20-attribute example

B. Case 1: 20-Attribute Example

The 20-attribute example has already been utilized by other researchers as described in [9], [13] and [2]. The usage matrix used in this example describes the reference pattern of 15 transactions accessing 20 attributes. The methods proposed in [9] and [2] both find the same optimal partition that groups 20 attributes into 4 fragments. In particular, Chakravarthy et al. [2] decide based on the PE value that this 4-fragment partition is better than the 5-fragment partition found in [13].

The termination numbers in all GAs set for this example are 10. Each GA is executed 100 times. **Fig. 2** shows the optimal costs found in each trial. The above mentioned 4-fragment partition is evaluated to have a PE value of 4627. So we argue that if the final partition of each trial has a PE value less or equal to 4627, then such trial is considered a success. The success rates of RD, RGS and GRGS GAs are 65.0%, 96.0% and 100%, respectively. Another interesting statistic is the average number of generations needed to reach the optimal solution in each GA. They are 56.6, 32.6, and 36.2 for RD, RGS and GRGS, respectively. Apparently, RD-GA performs worst among the three in terms of fitness and convergence speed.

V. SUMMARY AND CONCLUSIONS

Vertical database partition is a significant problem for database transaction performance. Some have used optimization techniques to improve vertical partition [5] [16]. In this article, we proposed a GA-based solution. Particularly, this solution features two new attempts: first, a RG string constraint is applied to overcome the redundant encoding of previous GAs for the partition problem or similar ones; second, group-oriented crossover and mutation operators are used to improve GA's convergence performance.

We discuss a new integer-encoding system for GA – the *RG string encoding*. Traditionally, the encoding scheme for clustering or grouping problems uses a fixed length integer string to represent a solution; each attribute corresponds to a value less than a predefined number to describe its membership. In the RG encoding system, the chromosome representing a partition scheme must comply with the RG string definition to ensure a one-to-one mapping between the solution coding and the partition result. Our new GRSG-GA uses RG string encoding and executes group oriented genetic operators. Compared to the variable length encoding adapted by GGA variants [4] [17], its length is fixed and can easily adapt conventional GA operators. At the same time, the number of each element uniquely identifies a group. The characteristic enables GA using this encoding scheme to utilize group oriented operators.

This paper compared three different GAs: one using traditional encoding scheme; one using RG string encoding scheme with traditional GA operators and the last one using RG string encoding with group oriented GA operators. These three GAs are employed to solve the vertical partition problem. The advantage of GRGS increases as the sizes of the problem grows.

The success of using GRGS to solve vertical partition problem suggests it may be used to solve other clustering or grouping problems. To solve this particular vertical partition problem, we designed two crossover operators and 4 mutation operators. In our experiments, we found out that the binary merge crossover operator and merge and jump mutation operators are more effective than the others. However, if the process is to be applied for other domains, different strategies may be incorporated into these operators.

VI. REFERENCES

[1] R. Bellman, Adaptive Control Processes: A Guided Tour, Princeton University Press, Princeton, New Jersey, 1961.

[2] S. Chakravarthy, J. Muthuraj, R. Varadarjan, and S.B. Navathe, "A Formal Approach to the Vertical Partition Problem in Distributed Database Design," *Technical Report, CIS Department*, University of Florida, Gainesville, FL, 1992

[3] D. W. Cornell and P. S. Yu, "An Effective Approach to Vertical Partition for Physical Design of Relational Database," *IEEE Transactions on Software Engineering*, Vol.16, No.2, pp. 248-258, 1990,

[4] E. Falkenauer. Genetic Algorithms and Grouping Problems. John Wiley & Sons, England, 1998.

[5] M. Hammer and B. Niamir, "A Heuristic Approach to Attribute Partition," *Proceedings of ACM-SIGMOD*, pp.93-100, 1979.

[6] J. H. Holland, Adaptation in Natural and Artificial System, University of Michigan Press, 1975.

[7] J. A. Hoffer and D. G. Severance, "The Use of Cluster Analysis in Physical Database Design," *Proceedings of the International Conference on Very Large Databases*, pp.69-86, 1975.

[8] S.B. Navathe, K. Karlapalem, and M. Ra, "A Mixed Fragmentation Approach for Initial Distributed Database Design," *Journal of Computers and Software Engineering*, Vol.3, No.4, 1995.

[9] S.B. Navathe, S. Ceri, G. Wiederhold, and J. Dou, "Vertical Partition Algorithms for Database Design," *ACM Transactions on Database Systems*, Vol.9, No.4, 1984.

[10] B. Niamir, "Attribute Partition in a Self-Adaptive Relational Database System," *Technical Report 192*, Massachusetts Institute of Technology, Laboratory for Computer Science Cambridge, Mass., Jan. 1978.

[11] M.T. Özsu and P. Valduriez, Principles of Distributed Database Systems, Prentice Hall, 1999.

[12] N.J. Radcliffe, Forma Analysis and Random Respectful Recombination, Belew and Booker, pp.222-229, 1991

[13] M. Ra. And S.B. Navathe, "Vertical Partition for Database Design: A Graphical Algorithm," *ACM SIGMOD Record*, Vol.18 No.2, pp.440-450, 1989.

[14] F. Ruskey, "Simple Combinatorial Gray Codes Constructed by Reversing Sublists," *Algorithms and Computation*, Lecture Notes in Computer Science 762, pp.201-208, Springer-Verlag, 1993.

[15] D. Smith: Bin Packing with Adaptive Search. Proceedings of the 1st International Conference on Genetic Algorithms (1985) J.J. Grefenstette (ed) [ICGA85]. pp. 202-207, 1985.

[16] S. Song and N. Gorla, "A Genetic Algorithm for Vertical Fragmentation and Access Path Selection," *The Computer Journal,* Vol.43, No.1, pp.81-93, 2000.

[17] C. Y. Hung, R. T. Sumichrast and E. C. Brown, "A Grouping Genetic Algorithm for Material Cutting Plan Generation," *Computers & Industrial Engineering*, Vol.44, 2003

[18] W. W. Chu and I. T. Ieong, "A Transaction-Based Approach to Vertical Partition for Relational Database Systems," *IEEE Transactions on Software Engineering*, Vol.19, No.8, pp.804-812, 1992.

[19] E. Falkenauer, "A Hybrid Grouping Genetic Algorithm for Bin Packing," *Journal of Heuristics*, Vol.2, pp.5-30, 1996.