Appling A Discrete Particle Swarm Optimization Algorithm to Database Vertical Partition

Bilal Benmessahel[1], Mohamed Touahria[1]

(1) département d'informatique, Université Ferhat ABBAS- SETIF
bilal.benmessahel@gmail.com

Abstract. Vertical partition is an important technique in database design used to enhance performance in database systems. Vertical fragmentation is a combinatorial optimization problem that is NP-hard in most cases. We propose an application and an adaptation of an improved combinatorial particle swarm optimization (ICPSO) algorithm for the vertical fragmentation problem. The original CPSO algorithm [3] suffers from major drawback—redundant encoding. This paper applies an improved version of CPSO that using the restricted growth (RG) string [5] constraint to manipulate the particles so that redundant particles are excluded during the PSO process. The effectiveness and efficiency of the improved CPSO algorithm are illustrated through several database design problems, ranging from 10 attributes/8 transactions to 50 attributes/50 transactions. In all cases, our design solutions match the global optimum solutions.

Keywords: Database vertical partition, Particle swarm optimization, RG String, Genetic algorithms, Optimization.

1 Introduction

Vertical partition (also called vertical fragmentation) 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. Compared to other types of data fragmentation, vertical partition is more complicated than horizontal partition because of the increased number of possible alternatives [1].

Vertical partition algorithms contain two essential parts: the optimization method and the objective function. Ozsu and Valduriez [1] 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.

Most previous algorithms employ multiple iterations of binary partition to approximate m-way partition. Navathe, Ceri, Wiederhold, and Dou (1984) propose the Recursive Binary Partition Algorithm (RBPA), which extends Hoffer and Severance's work by automating the selection process of vertical fragments; they propose some empirical objective functions. Cornell and Yu (1990) adopt the same approach but replace the empirical objective functions with one constructed on a model database. Chu and Ieong (1992) 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 (1979) propose a hill-climbing method that alternatively groups and regroups attributes and fragments to reach a suboptimal solution. Song and Gorla (2000) 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 PSO solution to vertical partition. By pure we mean the direct result from the PSO execution is already an m-way partition.

The work described in this paper considers the vertical partition problem and reports a Combinatorial PSO application that can eliminate the encoding redundancy by using a restricted growth (RG) string [3] constraint in constructing particles. To evaluate its effectiveness and demonstrate its superiority, we compare the result of using the improved CPSO with that of using traditional GA as well as RG string encoding with traditional object based GA operators called SGA and another GA based algorithm called Group oriented Restricted Growth String GA GRGS-GA developed to solve the vertical partition problem by Jun Du and Al (2006) [2].

The balance of the paper is structured as follows. Section 2 introduce the partition evaluator (PE) developed by Chakravarthy and al [4]; this evaluator will be used as the fitness function for the proposed approach and the two other approaches used in experiments. Section 3 presents the particle swarm optimization with a general brief overview RG strings. Section 4 we develop an improved particle swarm for vertical partition problem. Section 5 presents the application of the improved CPSO algorithm to the vertical partition problem and compares the proposed approach with the other two approaches SGA and GRGS-GA; and it is demonstrates that the improved CPSO can effectively find optimal solutions even for large vertical partition problems. Section 6 is summary and conclusions.

2 Objective functions for vertical partition

The two kinds of objective functions used for partition algorithms are 1) model cost functions based on the transaction access analysis on a model DBMS and 2) 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 [2].

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 Chakravarthy et al. (1992). 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 SEPE, including the formula, can be found in Chakravarthy et al. (1992).

3 Particle swarm optimization

PSO introduced by Kennedy and Eberhart [8] is one of the most recent metaheuristics, which is inspired by the swarming behavior of animals and human social behavior. Scientists found that the synchrony of animal's behavior was shown through maintaining optimal distances between individual members and their neighbors. Thus, velocity plays the important role of adjusting each member for the optimal distance. Furthermore, scientists simulated the scenario in which birds search for food and observed their social behavior.

They perceived that in order to find food the individual members determined their velocities according to two factors, their own best previous experience and the best experience of all other members. This is similar to the human behavior in making decision, where people consider their own best past experience and the best experience of the other people around them.

PSO algorithm

The general principles of the PSO algorithm are stated as follows. Similarly to an evolutionary computation technique, PSO maintains a population of particles, where each particle represents a potential solution to an optimization problem.

Let m be the size of the swarm. Each particle i can be represented as an object with several characteristics.

Suppose that the search space is a n-dimensional space, then the ith particle can be represented by a n-dimensional vector, $X_i = \{x_{i1}, x_{i2}, \dots x_{in}\}$, and velocity $V_i = \{v_{i1}, v_{i2}, \dots v_{in}\}$, where i = 1, 2, ..., m.

In PSO, particle i remembers the best position it visited so far, referred as $P_i = \{p_{i1}, p_{i2}, \dots p_{in}\}$, and the best position of the best particle in the swarm, referred as $G \ = \{G_1, G_2, \dots G_n\}$.
PSO is similar to an evolutionary computation algorithm and, in each generation *t*, particle i adjusts its velocity v_{ij}^t and position x_{ij}^t for each dimension *j* by referring to, with random multipliers, the personal best position p_{ij}^{t-1} and the swarm's best position G_{ij}^{t-1}, using Eqs. (1) and (2), as follows:

$$v_{ij}^t = v_{ij}^{t-1} + c1r1\left(p_{ij}^{t-1} - x_{ij}^{t-1}\right) + c2r2\left(G_{ij}^{t-1} - x_{ij}^{t-1}\right) \tag{1}$$

And

$$x_{ij}^t = x_{ij}^{t-1} + v_{ij}^t \tag{2}$$

Where c1 and c2 are the acceleration constants and r1 and r2 are random real numbers drawn from [0, 1]. Thus the particle flies through potential solutions toward P_i^t and G^t while still exploring new areas. Such stochastic mechanism may allow escaping from local optima. Since there was no actual mechanism for controlling the velocity of a particle, it was necessary to impose a maximum value V_{max} on it. If the velocity exceeded this threshold, it was set equal to V_{max}, which controls the maximum travel distance at each iteration, to avoid a particle flying past good solutions. The PSO algorithm is terminated with a maximal number of generations or the best particle position of the entire swarm cannot be improved further after a sufficiently large number of generations.

The aforementioned problem was addressed by incorporating a weight parameter in the previous velocity of the particle. Thus, in the latest versions of the PSO, Eqs. (2) and (3) are changed into the following ones:

$$v_{ij}^t = \chi(\omega v_{ij}^{t-1} + c1r1\left(p_{ij}^{t-1} - x_{ij}^{t-1}\right) + c2r2\left(G_{ij}^{t-1} - x_{ij}^{t-1}\right)) \tag{3}$$
$$x_{ij}^t = x_{ij}^{t-1} + v_{ij}^t \tag{4}$$

ω is called inertia weight and is employed to control the impact of the previous history of velocities on the current one. Accordingly, the parameter ω regulates the trade-off between the global and local exploration abilities of the swarm. A large inertia weight facilitates global exploration, while a small one tends to facilitate local exploration. A suitable value for the inertia weight ω usually provides balance between global and local exploration abilities and consequently results in a reduction of the number of iterations required to locate the optimum solution. χ is a constriction factor, which is used to limit the velocity.

The PSO algorithm has shown its robustness and efficacy for solving function value optimization problems in real number spaces. Only a few researches have been conducted for extending PSO to combinatorial optimization problems.

4 An improved CPSO for Database Vertical Partition

In this paper, we propose an improved version of the combinatorial CPSO algorithm [3] aimed at solving the Database Vertical Partition problem. The CPSO algorithm suffers from major drawback—redundant encoding. This paper applies the restricted growth (RG) string [5] constraint to manipulate the particles so that redundant particles are excluded during the PSO process. The following section presents the basics of the RG strings.

4.1 Basics of RG strings

The Restricted Growth (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, as constituents if RG string, they must satisfy Definition 1, given next. In addition to the formal definition of RG string, other supporting extended definitions are presented next:

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

$$\boldsymbol{r[i] \leq (max(r[0], r[1], \ldots, r[i-1]) + 1), 0} < i < n,\ r[\boldsymbol{0}] = \boldsymbol{1}$$

For example, {1 1 2 3 1 1 2 4} is 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 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 *RG string r*, denoted p_r^i, is the substring that includes the first *i* values of *r*.

For example, consider *r* = {11123221}, p_r^4 = {1112}.

4.2 Definition of a particle

Denote by $Y_i^t = \{y_{i1}^t, y_{i2}^t, \ldots, y_{in}^t\}$ the n-dimensional vector associated to the solution $X_i^t = \{x_{i1}^t, x_{i2}^t, \ldots, x_{in}^t\}$ taking a value in {-1, 0, 1} according to the state of solution of the *i*th particle at iteration *t*.

Y_i^t is a dummy variable used to permit the transition from the combinatorial state to the continuous state and vice versa.

$$Y_{ij}^{t} = \begin{cases} 1 & if\ x_{ij}^{t} = G_j^t \\ -1 & if\ x_{ij}^{t} = p_{ij}^{t} \\ -1\ or\ 1 & if\ (x_{ij}^{t} = G_j^t = p_{ij}^{t}) \\ 0 & otherwise \end{cases} \tag{5}$$

4.3 Velocity

Let $d1 = -1 - y_{ij}^{t-1}$ be the distance between x_{ij}^{t-1} and the best solution obtained by the *i*th particle.

Let $d2 = 1 - y_{ij}^{t-1}$ the distance between the current solution x_{ij}^{t-1} and the best solution obtained in the swarm.

The updated equation for the velocity term used in CPSO is then:

$$v_{ij}^{t} = w.v_{ij}^{t-1} + r1.c1.d1 + r2.c2.d2 \tag{6}$$

$$v_{ij}^{t} = w.v_{ij}^{t-1} + r1.c1.\left(-1 - y_{ij}^{t-1}\right) + r2.c2.(1 - y_{ij}^{t-1}) \tag{7}$$

With this function, the change of the velocity v_{ij}^{t} depends on the result of y_{ij}^{t-1}.

If $x_{ij}^{t-1} = G_j^{t-1}$, then $y_{ij}^{t-1} = 1$. Thereafter d2 turns to ''0'', and d1 takes "-2'', thus imposing to the velocity to change in the negative sense.

If $x_{ij}^{t-1} = p_j^{t-1}$, then $y_{ij}^{t-1} = -1$. Thereafter d2 turns to ''2'', and d1 takes ''0'', thus imposing to the velocity to change in the positive sense.

The case where $x_{ij}^{t-1} \neq G_j^{t-1}$ and $x_{ij}^{t-1} \neq p_j^{t-1}$, y_{ij}^{t-1} turns to ''o'', d2 is equal to ''1'' and d1 is equal to "-1'', thereafter the parameters r1; r2; c1 and c2 will determine the sense of the change of the velocity.

The case where $x_{ij}^{t-1} = G_j^{t-1}$ and $x_{ij}^{t-1} = p_j^{t-1}$, y_{ij}^{t-1} takes a value in {-1,1}, thus imposing to the velocity to change in the inverse sense of the sign of y_{ij}^{t}.

4.4 Construction of a particle solution

The update of the solution is computed within y_{ij}^{t}:

$$\lambda_{ij}^{t} = y_{ij}^{t-1} + v_{ij}^{t} \tag{8}$$

The value of y_{ij}^{t} is adjusted according to the following function:

$$y_{ij}^{t} = \begin{cases} 1 & if\ \lambda_{ij}^{t} > \alpha \\ -1 & if\ \lambda_{ij}^{t} < -\alpha \\ 0 & otherwise \end{cases} \tag{9}$$

The new solution is:

$$x_{ij}^{t} = \begin{cases} G_j^{t-1} & if\ y_{ij}^{t} = 1 \\ p_{ij}^{t-1} & if\ y_{ij}^{t} = -1 \\ a\ random\ number & otherwise \end{cases} \tag{10}$$

The choice previously achieved for the affectation of a random value in {1, -1} for y_{ij}^{t-1} in the case of equality between x_{ij}^{t-1} , p_{ij}^{t-1} and G_j^{t-1} allows to insure that the variable y_{ij}^t takes a value 0, and to permit a change in the value of variable x_{ij}^t. We define a parameter α for fitting intensification and diversification. For a small value of α, x_{ij}^t takes one of the two values p_{ij}^{t-1} or G_j^{t-1} (intensification). In the opposite case, we impose to the algorithm to assign a null value to the y_{ij}^t, thus inducing for x_{ij}^t a value different from p_{ij}^{t-1} *and* G_j^{t-1} (diversification). The parameters c1 and c2 are two parameters related to the importance of the solutions p_{ij}^{t-1} *and* $G_j^{t-1}$for the generation of the new solution X_i^t. They also have a role in the intensification of the search.

4.5 Solution representation

In this subsection, we describe the formulation of the improved CPSO algorithm for the database vertical partition problem.

One of the most important issues when designing the PSO algorithm lies on its solution representation. We setup search space of n-dimension for a relation of n-attributes. Each dimension represents an attribute and particle $X_i^t = \{x_{i1}^t, x_{i2}^t, \dots, x_{in}^t\}$ corresponds to the affectation of n attributes, such that $x_{ij}^t \in \{1, 2, 3,\dots, k\}$, where k is the number of fragments.

The scheme of Fig. 1 illustrates an example of the solution representation of particle X_i^t of the improved CPSO algorithm.

Let n = 7 attributes

k = 4 fragments.

	X_{t1}^t	X_{i2}^t	X_{i3}^t	X_{i4}^t	X_{i5}^t	X_{i6}^t	X_{i7}^t
X_i^t	1	2	1	3	4	2	2

Fig. 1. An example of solution representation with RG constraint.

4.6 Initial population

A swarm of particles is constructed based on RG string. In the particles generated, the RG constraint is enforced from the beginning. No rectification process is needed after all position in a particle are randomly created because each position is created as a random integer between 1 and the high potential degree for its position, complying with the RG string constraint. In the initial population, the first element is set to 1 and the upper bound for each element increases gradually and this rarely reaches *k*-1, where *k* is the number of fragments anticipated by the user

4.7 Creating a new solution

For creating new solution we use Eq(5). We obtain the vector value y_i^{t-1} , The vector of velocity V_i^t computed with Eq(6). The new value of λ_i^t is calculated using $\lambda_{ij}^t = \gamma_{ij}^{t-1} + v_{ij}^t$. γ_{ij}^t is determined using Eq(9) and using Eq (10) the new solution vector X_i^t is determined. But the new solution can breaking the RG constraint for this purpose we have designing *the rectifier.* The rectifier is a key issue in the proposed approach that each particle is RG string.

However, the initialization of particles does not guarantee each particle to be RG string. Also the operation of creating new solutions may change the constitution of a particle the population in a way that violates the RG string constraint. To handle such cases, we introduce a rectifying function that guarantees each particle to be RG string. For particles that violate the RG string constraint, the rectifier simply scans through a particle and converts it into RG string by adjusting the locations of its positions.

5 Implementation and experimental results

The algorithms have been implemented in java. All experiments with improved CPSO and GRGS-GA [2] and SGA were run in Windows XP on desktop PC with Intel Pentium4, 3.6 GHz processors. The GRGS-GA (Group oriented Restricted Growth String) is GA based approach proposed by Jun Du and al [2] for database vertical partition. The SGA is a Simple GA that uses random encoding schemes and classical genetics operators.

We have dividing the experimental section into two phases. The test phase and the comparison phase. In the test phase we have trying the improved CPSO on two cases, the first case is an attribute usage matrix (AUM) of 10 attributes and 8 transactions and the second case is an attribute use matrix of 20 attributes and 15 transactions.

In the comparison phase we have trying the improved CPSO with two others GA based algorithm GRGS-GA and SGA on two larges cases generated pseudo randomly with a pseudo random generator of AUM designed to generate a large size AUM.

5.1 Case 1: 10-attribute example

In this case, we use an attribute usage matrix AUM, with 10 attributes and 8 transactions. This AUM has already been utilized by other researchers as described in Cornell and Yu, 1990, Navathe and al 1984, Suk-kyu Song and Narasimhaiah Gorla, 2000, J. Muthuraj and al 1993. J. Muthuraj and al found that for this AUM the best PE value is 5820, which gives a fragmentation of 3 fragments {1 5 7} {2 3 8 9} {4 6 10}.

The improved CPSO find the best fragmentation in the 4 iteration, as illustrated in the figure 2. And the algorithm is executed 10 times. Figure 3 shows the optimal costs found in each trial. The above mentioned 3-fragment partition is evaluated to have a PE value of 5820. So we argue that if the final partition of each trial has a PE value less or equal to 5820, then such trial is considered a success. The success rate of the improved CPSO is 100%. Another interesting statistic is the average number of iterations needed to reach the optimal solution in the improved CPSO is 6.5. Apparently, the improved CPSO performs well in terms of fitness and convergence speed.

$T\backslash A$	1	2	3	4	5	6	7	8	9	10
$T1$	25	0	0	0	25	0	25	0	0	0
$T2$	0	50	50	0	0	0	0	50	50	0
$T3$	0	0	0	25	0	25	0	0	0	25
$T4$	0	35	0	0	0	0	35	35	0	0
$T5$	25	25	25	0	25	0	25	25	25	0
$T6$	25	0	0	0	25	0	0	0	0	0
$T7$	0	0	25	0	0	0	0	0	25	0
$T8$	0	0	15	15	0	15	0	0	15	15

Table 1. 10–attribute Matrix

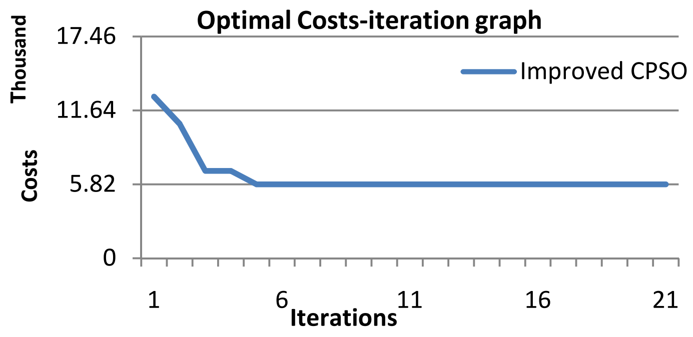

Fig. 2. PE values of best particles in each iteration

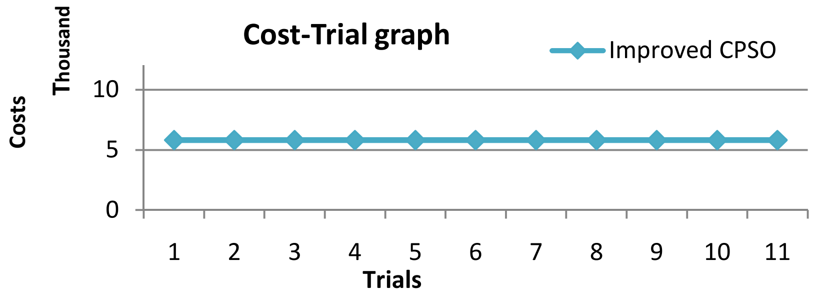

Fig. 3. Optimal solutions found in 10 trials case of 10 attributes

5.2 Case 2: 20-attribute example

The 20-attribute example has already been utilized by other researchers as described in Chakravarthy et al. (1992), Navathe et al. (1984), and Navathe and Ra (1989). The usage matrix used in this example describes the reference pattern of 15 transactions accessing 20 attributes. The methods proposed in Navathe et al. (1984) and Chakravarthy et al. (1992) both find the same optimal partition that groups 20 attributes into four fragments. In particular, Chakravarthy et al. (1992) decide based on the PE value that this four-fragment partition is better than the five-fragment partition found in Navathe and Ra (1989).

The improved CPSO Algorithm is executed 100 times. Figure 5 shows the optimal costs found in each trial. The above mentioned 4-fragment partition is evaluated to have a PE value of 4644. So we argue that if the final partition of each trial has a PE value less or equal to 4644, then such trial is considered a success. The success rate of the improved CPSO is 100%. Another interesting statistic is the average number of generations needed to reach the optimal solution in the improved CPSO is 30.4. Apparently, the improved CPSO performs well in terms of fitness and convergence speed.

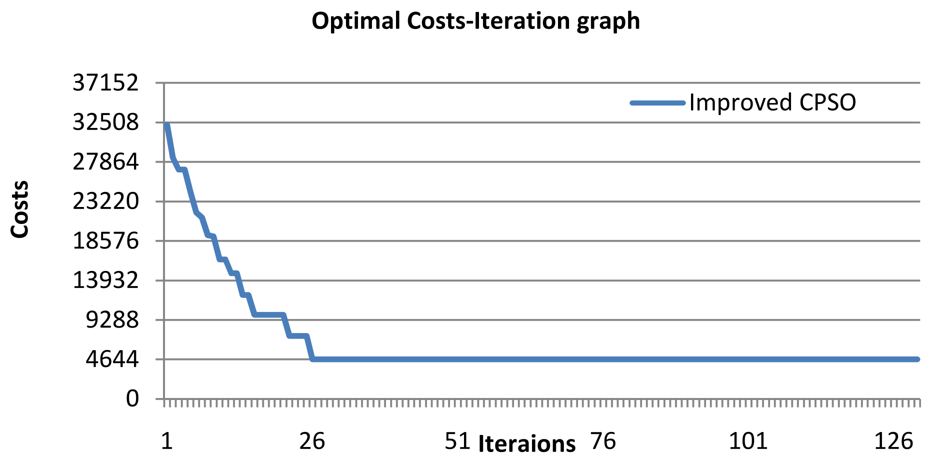

Fig. 4.The trends of PE values of best particles in each iteration. Case of 20 attributes

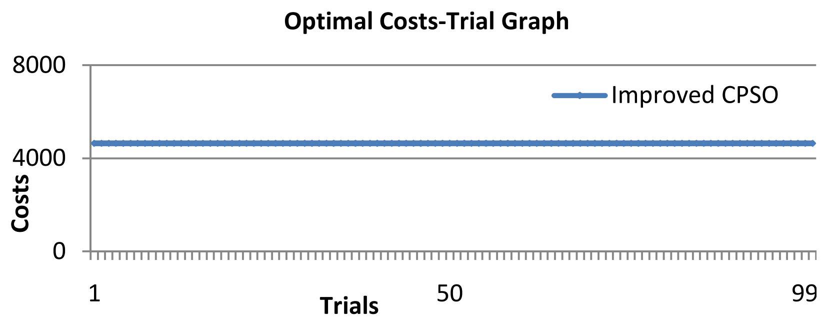

Fig. 5. Optimal costs-trial graph for 20-attribute example

5.3 Case 3: 20-attribute pseudo random AUM

In this case we compare the improved CPSO algorithm with two others GA based algorithms the GRGS-GA and SGA on pseudo random attribute usage matrix of 20 attributes and 20 transactions. This AUM generated using pseudo random AUM generator designed to generate a large size usage matrix. The number of fragments anticipated is 20 fragments. The value of the best fitness in this case is 0.

The average number of generations needed to reach the optimal solution in each algorithm. They are 30.4, 45.6, and 296.2 for improved PSO, GRGS-GA and SGA, respectively. Apparently, SGA performs worst among the three in terms of fitness and convergence speed.

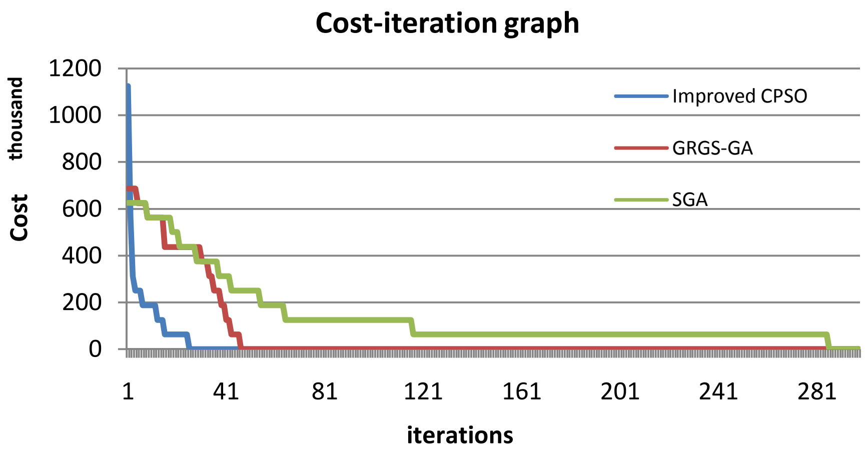

Fig. 6. The trends of PE values of best particles for Improved CPSO, GRGS-GA and SGA on the 20-Attribute pseudo AUM.

5.4 Case 4: 50-attribute example

In this case, we try to apply the improved CPSO described in Section 4 to a large size usage matrix. This AUM have 50 attributes and 50 transactions, the number of fragments anticipated is 50 fragments. The value of the best fitness is 0.

The figure 7 shows that the improved CPSO reach the best fragmentation in 193 iterations.

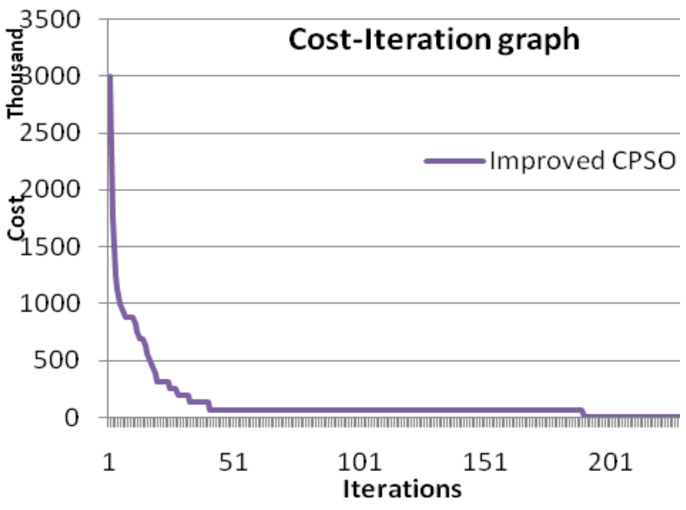

Fig. 7. The trends of PE values of best particles for Improved CPSO on the 50-Attribute pseudo AUM.

5.5 Results analysis

As the number of attributes grows, the improved CPSO becomes harder to converge because of the increased complexity of the partition problem.

Every improved CPSO trial finds the optimal partition known when the usage matrix is generated. The convergence speed of the improved CPSO is well over the two others GA based algorithms. So, the advantage of using the improved CPSO is apparent over using the other two GAs. Figure 6 shows the PE values for 20 AUM. Again from this graph, we conclude that the improved CPSO is the best among the three and GRGS-GA is better than SGA in terms of the convergence speed.

6 Conclusion

Vertical database partition is a significant problem for database transaction performance. In this article, we proposed a PSO-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, a comparison is used to evaluate the performance of the improved CPSO with to others GA based algorithms the GRGS-GA [2], and a simple GA called SGA in term of convergence speed and best fragmentation results.
The success of using the improved CPSO to solve vertical partition problem suggests it may be used to solve other clustering or grouping problems.

7 References

[1] Ozsu, M. T., & Valduriez, P. Principles of distributed database systems. Prentice Hall. (1999).

[2] Jun du, Reda Alhajj, Ken Barker « Genetic algorithms based approach to database vertical partition » Journal of Intelligent Information Systems Volume 26 , Issue 2 (March 2006) Pages: 167 - 183 Year of Publication: 2006

[3] B. Jarboui, M. Cheikh, P. Siarry, A. Rebai: Combinatorial particle swarm optimization (CPSO) for partitional clustering problem. Applied Mathematics and Computation 192(2): 337-345 (2007)

[4] Chakravarthy, S., Muthuraj, J., Varadarjan, R., & Navathe, S. B. (1992). A formal approach to the vertical partition problem in distributed database design. Technical Report, CIS Department, University of Florida, Gainesville, Florida.

[5] Ruskey, F. (1993). Simple combinatorial gray codes constructed by reversing sublists. Algorithms and Computation, Lecture Notes in Computer Science 762, pp.201–208, Berlin Heidelberg New York: Springer.

[6] Song, S., & Gorla, N. (2000). A genetic algorithm for vertical fragmentation and access path selection. The Computer Journal, 43(1), 81–93.

[7] Navathe, S. B., & Ra, M. (1989). Vertical partition for database design: A graphical algorithm. ACM SIGMOD Record, 18(2), 440–450.

[8] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, vol. IV, 1995, pp. 1942–1948.