Materialized View Selection Using Discrete Quantum Based Differential Evolution Algorithm

Raouf Mayata(✉) and Abdelmadjid Boukra

Faculty of Electronics and Computer Science Laboratory LSI, USTHB BP 32, 16111 El Alia, Bab-Ezzouar, Algiers, Algeria
mayataraouf@gmail.com, aboukra@usthb.dz

Abstract. A Data warehouse is a structure that stores big amount of data. This data is exploited in the best possible ways in order to improve the efficiency of decision-making. The huge volume of data makes answering queries complex and time-consuming. Therefore, materialized views are used in order to reduce the query processing time. Since materializing all views is not possible, due to space and maintenance constraints, materialized view selection became one of the crucial decisions in designing a data warehouse for optimal efficiency. In this paper, we propose a discrete - quantum based - version of Differential Evolution DE algorithm named QDE algorithm to solve the materialized view selection (MVS) problem with space constraint. This algorithm is a merging of the original DE with Quantum Evolutionary (QEA) algorithm. The experimental results show the efficiency of the proposed algorithm compared to well known algorithms used to solve MVS problem with space constraint such as HRUA and GEA.

Keywords: Data warehouse · Materialized view selection · Differential Evolution · Metaheuristic · Quantum Evolutionary

1 Introduction

Data warehouses is a repository of data integrated from multiple sources and delivered to decision makers supporting their complex OLAP queries and helping them to make proper strategic decisions. Due to their huge volume, data warehouses' response time to complex OLAP queries is considered as a problem. To shorten that response time, several optimization methods exist in different phases of building a data warehouse. We mention *Indexing* for the physical phase, and both *fragmentation* and *views materialization* for the logical phase. Unlike regular views, a materialized view store the result of queries physically in a table. The queries executed on these tables are significantly faster than those executed on the whole raw data. However, every materialized view will occupy additional storage space, and need to be maintained (updated) from the raw

S. Chikhi et al. (Eds.): MISC 2020, LNNS 156, pp. 203–216, 2021.
https://doi.org/10.1007/978-3-030-58861-8_15

data periodically. In a data warehouse context, one always has to deal with limited resources. Therefore, all possible views can not be materialized due storage space and maintenance constraints. Materialized View Selection (MVS) is the problem of choosing among the views' universe, the set of ones that minimizes the Total Processing Cost (TPC) when materialized with respect to space or maintenance constraints. In this paper, authors will focus on the MVS problem with storage space constraint.

The MVS problem is proven to be NP-Hard. Therefore, several approaches using heuristics and meta-heuristics were proposed so far in literature.

2 Related Work

When talking about MVS problem, one has to mention the first paper dealing with it, which is [5]. At that time, they considered only the storage space constrained MVS problem. Authors proposed a greedy algorithm named HRU, this algorithm selects views to materialize in every iteration in the following way: for every view not yet chosen for materialization, the algorithm calculates its total benefit when materialized, and the view having the maximum benefit is then materialized. This step is repeated until there is not enough space to materialize further views. An extension was proposed in [1] to the selection of both views and indices. [3] also proposed heuristics based on greedy algorithm, the difference between these heuristics and HRU is that they select the view with maximum benefit per unit space. [6] compared between different greedy based algorithms for large problem instances (>10 dimensions), after proposing a search space reduction algorithm based on the observation that some hierarchically related views may have the same size. There have been several evolutionary based approaches proposed for the storage space constrained MVS problem such as [9] in which the authors proposed a Genetic Greedy Algorithm; they used the greedy concept in the repairing function. The MVS problem with maintenance constraint was first considered in [2], after noticing that storage space causes fewer problems as a constraint. They propose an inverted tree and A-star algorithms. [14] applied a hybrid evolutionary algorithm consisting of two levels of algorithm processing. The higher-level algorithm searches for good global processing plan from local processing plans based on queries. The lower level algorithm selects the best set of materialized views with the minimal total cost for a particular global processing plan. [8] proposed a genetic algorithm with a penalty function. [13] proposed an extension of [8] by replacing the penalty function with stochastic ranking procedure. In all of the aforementioned works the authors used one of three frameworks to represent all possible views. The first one is *lattice* framework introduced in [5]. We also find the Multiple View Processing Plan(MVPP) and AND-OR graph.

3 Background

3.1 Lattice Framework

V. Harinarayan et al. [5] Introduced an oriented acyclic graph G = (V, E) called *lattice* to represent the dependencies between all possible views of a star schema, where each view is characterized by a group-by clause. The Vertices V(G) of this graph represents the set of views or queries, while the Edges E(G) represents the relation between those queries. For a pair of vertices *(x,y)* we say that x depends on y $(x \leq y)$ if the query x can be answered using the result of y. An edge (v_i, v_j) between two views v_i and v_j exists only if v_i is the "immediate proper ancestor" of v_j, which means that $(v_j \leq v_i)$ and $\nexists v_k$ $(v_j \leq v_k \wedge v_k \leq v_i)$ for every three distinct v_i, v_j and v_k. Fig. 1 is the lattice representation of the views dependencies of the star schema given in the same figure.

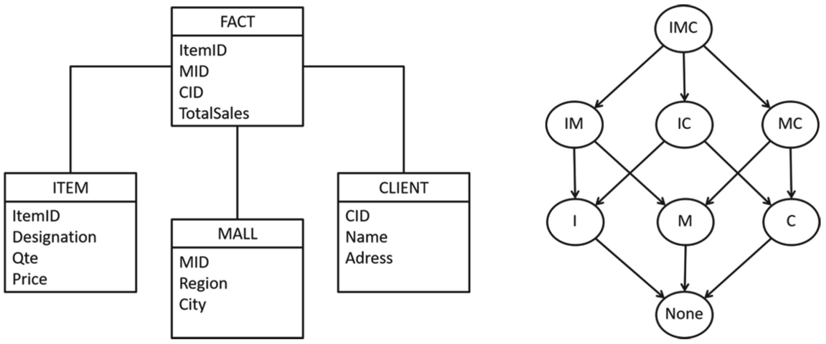

Fig. 1. Example of a lattice framework representing dependencies between views of a star schema

3.2 Space-Constrained MVS Problem

The vertices and edges of the lattice are weighted in the following way:

- Every vertex $v \in V(G)$ has three weights :
  - r_v: the initial data scan cost.
  - f_v: query frequency.
  - $|v|$: the storage space needed to materialize the view v itself.
- Every edge $(v, u) \in E(G)$ has one weight:
  - $w_q(v, u)$: processing cost of query u using v.

An additional vertex v_+ is added to the graph, this vertex represents the raw data from which every query can be calculated. For every pair of queries (u, v), we call $q(u, v)$ the query processing cost of u using v. $q(u, v)$ is the sum of the query processing costs of every edge in the shortest path between u and v, added to the initial data scan cost of $v(r_v)$. If no path links u to v, the vertex v_+ is used. Let M be the set of views to be materialized, $q(v, M)$ be the minimum query processing cost of v in presence of M. The space-constrained materialized view selection (MVS) problem can be formulated as follows :

Select a set of views M to materialize which minimizes $T(G, M)$, where:

$$T(G, M) = \sum_{v \in V(G)} f_v.q(v, M) \tag{1}$$

Under the constraint $H(G, M) \leq S$, where:

$$H(G, M) = \sum_{v \in M} |v| \tag{2}$$

S is the storage threshold not to be exceeded by the views of M.

3.3 Quantum Inspired Evolutionary Algorithm Overview

The smallest unit of information in quantum computing is called q-bit, and each q-bit is defined by two complex numbers α and β verifying: $|\alpha|^2 + |\beta|^2 = 1$. $|\alpha|^2$ represents the probability of the q-bit to be in the state '0', as for $|\beta|^2$, it represents the probability of the q-bit to be in the state '1'. In QEA, each individual q is represented by a quantum vector of length m (m being the length of solutions). Each column is represented by a single q-bit, which means: $|\alpha_i|^2 + |\beta_i|^2 = 1, i = 1, 2, 3, ...m$.

q-bit representation of solutions has the advantage of being able to represent a linear superposition of states, i.e: a single quantum vector can refer to multiple solution states with different probabilities [4], which gives the QEA a better population diversity characteristic. Like any independent evolutionary algorithm, QEA has his own operators. We mention *Measurement*, *Quantum interference* and *Mutation*. *Measurement* is the name of the operation allowing QEA to generate a solution vector using a quantum vector. As for *Quantum interference*, it intensifies the research around the best self/global solution. It consists in moving the state of each q-bit in the direction of the corresponding bit's value of the best solution in progress, and this is accomplished by making a rotation whose angle is a function of the amplitudes α_i and β_i. Like the evolutionary mutation, quantum *Mutation* is used to inspect some of the current solution's neighbors with some defined probability.

3.4 Differential Evolution

The differential Evolution (DE) algorithm was first introduced not as a separate evolutionary algorithm but as a genetic algorithm variation [12]. It is considered as a unique EA because it is not biologically motivated. DE is a population-based algorithm that is designed to optimize functions in an n-dimensional continuous domain [12]. Each individual in the population is an n-dimensional vector that represents a candidate solution to the problem. The basic idea of DE algorithm is described as follows:

Like every EA algorithm we first generate randomly a set of individuals $X = x_1, x_2, x_3, ..., x_n$. Then we proceed in the following way:

1. *Mutant vector generation:*
   For each individual x_i, a mutant vector v_i is generated using three different random individuals x_{r1}, x_{r2} and x_{r3} other than x_i, following the formula (3):

$$V_i = x_{r1} + F(x_{r2} - x_{r3}) \tag{3}$$

   Where x_{r1}, x_{r2} and x_{r3} are three distinct vectors, and F is a real number used to scale the difference between x_{r2} and x_{r3}.

2. *Trial vector generation:*
   The mutant vector v_i is then combined (crossed over) with the individual x_i itself to generate a trial vector u_i. Crossover is done as follows (formula (4)):

$$u_{ij} = \begin{cases} v_{ij}, & if(rand < c)\ or\ (j = Jr) \\ x_{ij}, & otherwise \end{cases} \tag{4}$$

   For every j in $[1, n]$, where n is the problem size and is also the size of $u_i, v_i\, and\, x_i$; u_{ij} is the j^{th} component of u_i, v_{ij} is the j^{th} component of v_i, x_{ij} is the j^{th} component of the individual x_i, $rand$ is a random number taken from the uniform distribution in $[0, 1]$; c is the constant crossover rate (in $[0, 1]$); and Jr is a random integer taken from the uniform distribution in $[1, n]$.

3. *Best fit choice:*
   After the trial vector ui is created, it is compared to the individual xi, and the fittest vector in each pair (x_i, u_i) is kept to the next generation of DE algorithm. The least fit is discarded.

$$x_{i+1} = \begin{cases} u_i, & if\ (f(u_i) < f(x_i)) \\ x_i, & otherwise \end{cases} \tag{5}$$

4 QDE Algorithm

In this paper we propose a new hybrid algorithm (QDE) using both QEA and DE algorithms. In QDE, the population is a set of quantum vectors. Each quantum vector Q_i is considered as a vector. After measurement, a set of solutions X_i is generated from every Q_i vector. Then, *mutant vectors generation* and *trial vectors generation* are applied on every vector Q_i like in the original DE algorithm. The *best fit choice* is also applied on both the original Q_i population and the new trial vectors based on the fitness of their corresponding solutions after applying measurement. After that, the *Quantum interference* operator of QEA is used on the new Q_i generation to improve it by rotating towards the global best solution. The above steps are repeated until some stop criteria are satisfied.

The main idea of the QDE is outlined in the Fig. 2 and its pseudo code is given in Algorithm 1.

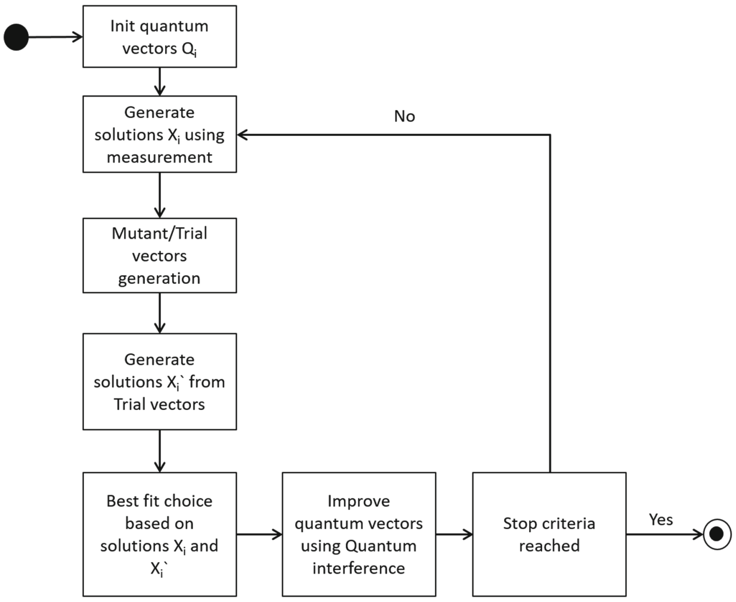
Init quantum vectors Q_i
Generate solutions X_i using measurement
No
Mutant/Trial vectors generation
Generate solutions X_i` from Trial vectors
Best fit choice based on solutions X_i and X_i`
Improve quantum vectors using Quantum interference
Stop criteria reached
Yes

Fig. 2. Main steps of QDE Algorithm

4.1 Solution Representation

In this work, the authors represent every solution by a binary vector $Xi = (x_1, x_2, x_3, ..., x_N)$, where N is the total number of views. A case x_i takes the value 1 if the view v_i is materialized, 0 otherwise. As shown in Fig. 3, the binary vector X is the representation of the solution given by the lattice L (the lattice structure of a star model with 3 dimensions), in which the materialized views are painted in grey. In this example the views v_2 and v_3 are materialized, which corresponds to the value 1 in the second and third position of the vector X. Whereas, the rest of the views are not materialized, hence the value 0 of the remaining positions.

Algorithm 1: QDE Pseudo Code

```
$N$: population size;
$n$: number of views;
$f$: stepsize parameter $\in [0.4, 0.9]$;
$c$: crossover rate $\in [0.1, 1]$;
$t$: the maintenance constraint threshold $\in [0.1, 1]$;
$C_{max}$: the max space-cost when all vertices are materialized;
$ra$: rotation angle $\in [0.01\pi, 0.1\pi]$;
begin
    Initialize a population of quantums $Q_i$ where $i \in [1, N]$;
    foreach $Q_i$ $(i \in [1, N])$ do
        $X_i$= measurement($Q_i$);
        if $(H(X_i) > C_{max} * t)$ then Repair $X_i$;
    end
    repeat
        foreach $Q_i$ $(i \in [1, N])$ do
            $r_1$ = random int $\in [1, N] : r_1 \neq i$;
            $r_2$ = random int $\in [1, N] : r_2 \notin \{i, r_1\}$;
            $r_3$ = random int $\in [1, N] : r_3 \notin \{i, r_1, r_2\}$;
            $v_i = Q_{r1} + f(Q_{r2} - Q_{r3})$;          /* $v_i$ : mutant vector */
            $J_r$ = random integer $\in [1, n]$;
            foreach $Q_i$ $(i \in [1, N])$ do
                /* $u_i$ : trial vector                          */
                if $(rand < c) or (j = J_r)$ then $u_{ij} = v_{ij}$;
                else $u_{ij} = Q_{ij}$;
            end
        end
        foreach $i \in [1, N]$ do
            $X_i'$= measurement($u_i$);
            if $H(X_i') > C_{max} * t$ then Repair $X_i'$;
            if $T(X_i') < T(X_i)$ then $Q_i = u_i$;
            apply Quantum interference on $Q_i$;
        end
        if no update on global best for 100 iterations then Apply reset
        operator;
    until stop criteria;
end
```

4.2 Q-Bit Representation

Since $|\alpha|^2 + |\beta|^2 = 1$, it basically represents the equation of a unit circle and each point on its perimeter can be represented by a single variable θ with the Cartesian co-ordinates given by cos_θ and sin_θ where θ is defined in $[0, 2\pi]$. Therefore, instead of using α and β to represent a q-bit like QEA, we use the variable θ,

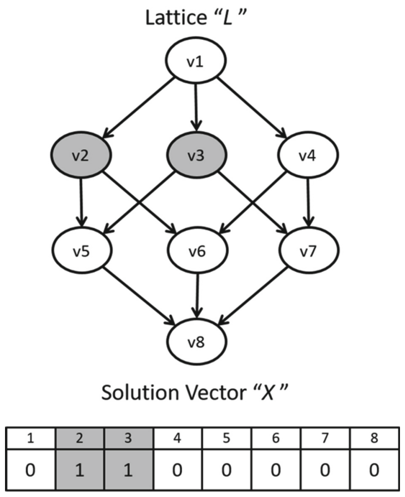

Fig. 3. Example of solution representation

from which the values α and β can be inferred. This, not only will allow us to gain space, but also will make the use of quantum interference operator easier, because the rotation angle will be added directly to the quantum itself.

4.3 Quantum Interference Application

As mentioned in the above section, quantum interference is applied by adding the rotation angle γ to the quantum directly. In order to apply this operator, we use the same look-up table used in [7] described in Table 1.

Table 1. Look up table from [7]

α	β	Best solution bit value	Angle
> 0	> 0	1	$+\gamma$
> 0	> 0	0	$-\gamma$
> 0	< 0	1	$-\gamma$
> 0	< 0	0	$+\gamma$
< 0	> 0	1	$-\gamma$
< 0	> 0	0	$+\gamma$
< 0	< 0	1	$+\gamma$
< 0	< 0	0	$-\gamma$

The choice of the rotation angle's value γ is very important. A badly chosen value can lead to premature or very late convergence or divergence.

4.4 Discrete Vs Continuous Search Space

Both QEA and DE algorithms are dedicated to solve continuous problems. In those algorithms, solution vectors move around the search space which is within a continuous real domain. However, MVS problem deals with a binary discrete search space. Some adaptation of those algorithms to solve binary discrete problems is needed. Several works like [10,11] use transfer functions. Those functions consider the continuous search space as a probability of being at the values 0 or 1. In this work the authors follow a similar approach. Instead of having solution vectors, the Vector population is a set of quantum vectors. Those vectors generate solutions using the measurement operator mentioned earlier. So, instead of manipulating the actual position of each solution, the probability of the position being at 0 or 1 is manipulated. This way one can use any continuous search space dedicated algorithm to solve a problem having a binary discrete search space.

4.5 Dealing with Unfeasible Solutions

Since the MVS is a constrained combinatorial optimization problem, several solutions during the algorithm's application will not respect the constraint forming what is called unfeasible solutions. One of the most commonly used methods to deal with this problem is to implement a penalty function penalizing those unfeasible solutions, and ensuring that even if they have good fitness, they will not be highly evaluated among other solutions. In our work, we do not use any penalty function. Instead of that, we use a repairing function. We keep dematerializing the smallest view in every iteration, until the constraint is not violated any more. Doing that will ensure that after measurement, there will be no unfeasible solutions.

4.6 The "Reset" Operator

As in most swarm algorithms, every single particle seeks the best solution. With lack of diversity and after some iterations, the particles stagnate at the same position (around the best solution). Consequently, the population prematurely converges to local optima. To deal with this issue, the authors have introduced the reset operator (known as reseeding or extinction) to ensure the escape from the local optima case. If the best global solution does not change for a certain number of iterations, *reset operator* is applied by replacing all the quantum vectors with new ones. By doing that, and keeping the best global solution found so far, one can ensure that those vectors will seek the best solution from a different starting points, covering new areas in the search space, and ensuring diversity for the whole population. In QDE the number of iteration needed to apply the reset operator is set to 100 iterations.

5 Experimental Results

In this section we will prove the effectiveness of the proposed QDE algorithm against the well-known HRU algorithm of [5]. Also, we test it against the Genetic

Algorithm which is one of the most commonly used evolutionary algorithms used to solve MVS problem both with space and maintenance constraints. The tests were conducted on an Intel based i-5 2.4 GHz PC having 8 GB RAM. Due to several tests, the parameters were set as shown in Table 2.

Table 2. Parameters used for QDE/GEA implementation

Parameters	QDE values	GEA values
Population size	100	100
Nb of generations	1000	1000
Query frequencies	following Zipf distribution	
Scale factor	0.5	-
Cross Over rate	0.3	0.5
Rotation angle γ	0.02π	-
Mutation Pb	-	0.001

We chose to compare QDE to HRU for 5 and 6 dimensions' lattices. Table 3 shows the solution's fitness (TPC) given by HRU, compared to the average of 30 independent runs of QDE for 1000 iterations. For 5-dimension lattice, QDE outperforms HRU for almost all space thresholds. They provide same TPC for the space threshold 0.7, 0.2 and 0.1. this is due to small search area for this lattice. As for 6-dimension lattice, QDE provide better solutions for all space thresholds.

Table 3. QDE vs HRU for 5-dimension and 6-dimension lattices

	5-Dimension		6-Dimension	
t	HRU	QDE	HRU	QDE
0.9	4943892	**4939772**	5091526	**5090352**
0.8	4967327	**4967200**	5114865	**5113931**
0.7	**5000283**	**5000283**	5156049	**5143236**
0.6	5052126	**5051514**	5186824	**5180342**
0.5	5106501	**5099592**	5245149	**5231102**
0.4	5228176	**5190686**	5311518	**5296674**
0.3	5337020	**5299322**	5429357	**5385871**
0.2	**5479742**	**5479742**	5554820	**5552887**
0.1	**5864309**	**5864309**	5890921	**5866430**

Table 4, Table 5 and Table 6 contain the results of 30 independent runs of QDE and GEA on a 5, 6 and 7-dimension lattices respectively. We kept track of the mean, min and max TPC found during these runs.

Table 4. QDE vs GEA for 5-dimension lattice

t	**0.1**			0.2		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5864309**	**5864309**	**5864309**	**5479742**	**5479742**	**5479742**
GEA	5895203	**5864309**	6178534	5499864	**5479742**	5580018
t	**0.3**			**0.4**		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5299322**	**5299322**	**5299322**	**5190686**	**5190686**	**5190686**
GEA	5312721	**5299322**	5354626	5203998	**5190686**	5236239
t	**0.5**			**0.6**		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5099592**	**5099592**	**5099592**	**5051514**	**5051328**	**5052126**
GEA	5106157	**5099592**	5127608	5077674	5064453	5092808
t	**0.7**			**0.8**		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5000283**	**5000283**	**5000283**	**4967200**	**4967200**	**4967200**
GEA	5064873	**5000283**	5014292	4969717	**4967200**	4975976

By observing Table 4, one can notice that QDE outperforms GEA, for all thresholds. What seems to be the best solution found by GEA (min TPC) is almost always the mean TPC of QDE. i.e QDE found that solution for every single run since mean = min = max for all thresholds except 0.6 as shown in Table 4.

As for 6-dimesion lattice, Table 5 shows that QDE finds better solutions since its mean, min and max TPC are better than GEA, except for thresholds 0.1 and 0.3 where GEA finds a similar min TPC.

Finally for 7-dimension lattice (Table 6), the supremacy of QDE over GEA becomes very clear since QDE outperforms GEA for all the statistical indicators (min,max and mean) TPC and that is for all the thresholds with no exception. This proves the scalability of QDE's solutions quality.

Table 5. QDE vs GEA for 6-dimension lattice

t	0.1			0.2		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5866430**	**5866430**	**5866430**	**5552887**	**5552444**	**5554820**
GEA	5903295	**5866430**	6005452	5584828	5555631	5662251
t	**0.3**			**0.4**		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5385871**	**5385457**	**5388109**	**5296674**	**5295998**	**5301854**
GEA	5415317	**5385457**	5463672	5309944	5301737	5327368
textbit	**0.5**			**0.6**		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5231102**	**5229696**	**5234680**	**5180342**	**5179914**	**5181307**
GEA	5241880	5233912	5251542	5186114	5181298	5194346
t	**0.7**			**0.8**		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5143236**	**5142962**	**5144289**	**5113931**	**5113650**	**5116112**
GEA	5146883	5143262	5152941	5115813	5113974	5118915

Table 6. QDE vs GEA for 7-dimension lattice

t	0.1			0.2		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5866197**	**5864225**	**5878000**	**5589373**	**5585840**	**5598541**
GEA	5920038	5873451	6009651	5619766	5591987	5692837
t	**0.3**			**0.4**		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5454278**	**5451819**	**5459491**	**5366446**	**5364286**	**5369022**
GEA	5474363	5460181	5505542	5380534	5369855	5397451
t	**0.5**			**0.6**		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5303783**	**5301315**	**5306194**	**5259941**	**5258691**	**5261597**
GEA	5310349	5303681	5319140	5265795	5261428	5274166
t	**0.7**			**0.8**		
	mean TPC	min TPC	max TPC	mean TPC	min TPC	max TPC
QDE	**5226450**	**5225031**	**5228281**	**5200795**	**5200095**	**5201447**
GEA	5229828	5226855	5233312	5202571	5200888	5205480

6 Conclusion

In this paper, we adapted a binary version of Differential Evolution (DE) Algorithm to solve MVS problem with space constraint. We used the lattice structure of [5] to represent hierarchy between different views. Also, adaptation of the DE algorithm to the discrete binary case was done through merging DE algorithm with the QEA algorithm instead of the use of transformation functions such as Sigmoid or V-shaped. The Experimental results show the efficiency of the proposed QDE algorithm in comparison with HRU and GEA algorithms. That is shown through the TPC values of QDE, HRU and GEA which is significantly better in QDE. As perspective, we want to investigate the use QEA alongside with other evolutionary algorithms. That is, to study further the efficiency of QEA when use instead of transformation functions.

References

1. Gupta, H., Harinarayan, V., Rajaraman, A., Ullman, J.D.: Index selection for OLAP. In: 13th International Conference on Data Engineering, 1997. Proceedings. pp. 208–219 (1997). https://doi.org/10.1109/ICDE.1997.581755
2. Gupta, H., Mumick, I.S.: Selection of views to materialize under a maintenance cost constraint. In: Proceedings of the 7th International Conference on Database Theory, vol. 13, pp. 453–470 (1999).https://doi.org/10.1007/3-540-49257-7-28, http://dl.acm.org/citation.cfm?id=645503.656261
3. Gupta, H., Mumick, I.S.: Selection of views to materialize in a data warehouse. IEEE Trans. Knowl. Data Eng. **17**(1), 24–43 (2005). https://doi.org/10.1109/TKDE.2005.16
4. Han, K.H., Kim, J.H.: Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Trans. Evol. Comput. **6**(6), 580–593 (2002). https://doi.org/10.1109/TEVC.2002.804320
5. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing data cubes efficiently. ACM SIGMOD Record **25**(2), 205–216 (1996). https://doi.org/10.1145/235968.233333. http://portal.acm.org/citation.cfm?doid=235968.233333
6. Łatuszko, M., Pytlak, R.: Methods for solving the mean query execution time minimization problem. Eur. J. Oper. Res. **246**(2), 582–596 (2015). https://doi.org/10.1016/j.ejor.2015.04.041
7. Layeb, A.: A novel quantum inspired cuckoo search for knapsack problems. Int. J. Bio-Inspl. Comput. **3**(5), 297 (2011)
8. Lee, M., Hammer, J.: speeding up materialized view selection in data warehouses using a randomized algorithm. Int. J. Coop. Inf. Syst. **10**(03), 327–353 (2001). https://doi.org/10.1142/S0218843001000370
9. Lin, W.Y., Kuo, I.C.: A Genetic Selection Algorithm for OLAP Data Cubes.Knowledge and Information Systems **6**(1), 83–102 (2004). https://doi.org/10.1007/s10115-003-0093-x, http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s10115-003-0093-x
10. Mirjalili, S., Mirjalili, S.M., Yang, X.S.: Binary bat algorithm. Neural Comput. Appl. **25**(3–4), 663–681 (2014). https://doi.org/10.1007/s00521-013-1525-5
11. Rashedi, E., Nezamabadi-Pour, H., Saryazdi, S.: BGSA: binary gravitational search algorithm (2010). https://doi.org/10.1007/s11047-009-9175-3

12. Simon, D.: Evolutionary Optimization Algorithms. Wiley (2013). https://books.google.dz/books?id=gwUwIEPqk30C
13. Yu, J.X., Yao, X., Choi, C.H., Gou, G.: Materialized view selection as constrained evolutionary optimization. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **33**(4), 458–467 (2003). https://doi.org/10.1109/TSMCC.2003.818494
14. Zhang, C., Yao, X., Yang, J.: An evolutionary approach to materialized views selection in a data warehouse environment. IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev. **31**(3), 282–294 (2001). https://doi.org/10.1109/5326.971656