

Materialized View Selection as Constrained Evolutionary Optimization

Jeffrey Xu Yu, Xin Yao, *Fellow, IEEE*, Chi-Hon Choi, and Gang Gou

Abstract—One of the important issues in data warehouse development is the selection of a set of views to materialize in order to accelerate a large number of on-line analytical processing (OLAP) queries. The maintenance-cost view-selection problem is to select a set of materialized views under certain resource constraints for the purpose of minimizing the total query processing cost. However, the search space for possible materialized views may be exponentially large. A heuristic algorithm often has to be used to find a near optimal solution. In this paper, for the maintenance-cost view-selection problem, we propose a new constrained evolutionary algorithm. Constraints are incorporated into the algorithm through a stochastic ranking procedure. No penalty functions are used. Our experimental results show that the constraint handling technique, i.e., stochastic ranking, can deal with constraints effectively. Our algorithm is able to find a near-optimal feasible solution and scales with the problem size well.

I. INTRODUCTION

TODAY'S markets are much more competitive and dynamic than ever. Business enterprises prosper or fail according to the sophistication and speed of their information systems, and their ability to analyze and synthesize information using those systems. A data warehouse is a subject-oriented, integrated, time-varying, nonvolatile collection of data that is used primarily in organization decision making [1]. As an emerging network service, a data warehouse system collects data from many data sources through communication networks locally and internationally by adopting a update-driven approach. A data warehouse system provides a solid platform of consolidated historical data for analysis, and disseminates such analysis to users locally and remotely.

In addition to large volumes of data being transferred to a data warehouse via communication networks, the amount of data maintained in a data warehouse is huge in size, in the range of hundreds of gigabytes or terabytes. Upon such enormous amount of data collected from different sources, various of business decisions need to be made in a few minutes, in order to cope with the rapid change in different sectors of the market from time to

time. Such timely manner requests the data warehouse system to be able to answer OLAP (On-Line Analytical Processing) queries efficiently, and be able to assist executives or managers to make a better and faster decision. OLAP queries can be issued by decision-makers locally or remotely. The outcome of OLAP queries are of the statistical analysis or summarization, and the query processing time for such OLAP queries is considerably long. In order to efficiently support decision-making or OLAP queries, a data warehouse system needs to precompute or materialize some of such OLAP queries. The OLAP queries being materialized are called materialized views, or simply views. The motivation is to minimize the total query processing cost for all possible OLAP queries by selection of a set of materialized views under some resource constraints. It is worth noting that it is impractical to maintain materialized views for all OLAP queries due to the huge disk-space consumption and/or large update cost.

The important issue is how to select such a set of materialized views in order to minimize the total query processing time of OLAP queries with a certain constraint. The constraint can be either disk-space constraint or maintenance-cost constraint. The disk-space constraint specifies the availability of the disk-space in a data warehouse, whereas the maintenance-cost constraint specifies how long all views must be updated, because changes to the source data result in recomputing the materialized views accordingly, which will be periodically done in a time window.

- **Disk-space Constraint Handling:** Most of the reported studies [2]–[5] studied a disk-space view-selection problem, using a disk-space constraint, as the disk consumption of OLAP queries is very large. Harinarayan *et al.* in [2] studied the disk-space view-selection problem using a linear cost model. The linear cost model states that the cost of answering a query using a view is the number of records present in the view. Their greedy algorithm can reach at least 63% of the benefit of the optimal solution, in order to identify a set of materialized views for minimizing the total query processing cost. Gupta *et al.* [3] extended the results reported in [2] to the selection of views and indices in datacubes. They studied the precomputation of indices and subcubes, and discussed a family of one-step near-optimal algorithms under a given disk-space constraint. Gupta [4] presented a theoretical formulation of the general view-selection problem in a data warehouse and generalized view selection problems as AND, OR, and AND-OR graph problems. Shukla *et al.* [5] introduced a heuristic algorithm called PBS

Manuscript received August 31, 2002; revised March 24, 2003. This work was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project CUHK4198/00E). This paper was recommended by Guest Editors W. Pedrycz and A. Vasilakos.

J. X. Yu, C.-H. Choi and G. Gou are with the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong (e-mail: yu@se.cuhk.edu.hk; chchoi@se.cuhk.edu.hk; ggou@se.cuhk.edu.hk).

X. Yao is with the School of Computer Science, The University of Birmingham, Edgbaston B15 2TT, U.K. (e-mail: x.yao@cs.bham.ac.uk).

Digital Object Identifier 10.1109/TSMCC.2003.818494

which achieved the same $(0.63 - f)^1$ bound as [2] but with faster running time. They introduced chunk-based precomputation and showed that using chunks for aggregate subset precomputation can make the benefit larger than the “optimal” benefit when picking aggregates. All the above work considered a disk-space constraint and provided greedy algorithms using the linear cost model. Most of the greedy algorithms start from an empty set and select the next view with the maximum benefit per unit space in turn. The benefit of the views which have been selected will be unchanged in the subsequent view-selection processes, it is defined as *monotonic property*. The algorithms continue to pick views until the space limit is reached. However, the disk is cheap and the disk-space constraint becomes less important nowadays.

- **Maintenance-cost Constraint Handling:** Gupta and Mumick [6] first considered a maintenance-cost view-selection problem where the constraint is maintenance cost/time. This problem is more difficult than the disk-space view-selection problem, because the total maintenance cost for a set of views may decrease when more views are added to materialized. This is defined as *nonmonotonic property*. Gupta and Mumick proposed an inverted-tree greedy algorithm and an A*-heuristic. However, the quality of their algorithms depends heavily on the initial conditions and the heuristic used. While the A*-heuristic can find the optimal solution, it is an exponential algorithm in the worst case and may take a long time to run [7].

This materialized view selection problem is proven to be NP-hard [6]. For example, Baralis *et al.* described a real store chain application that only has four dimensions, namely, Product (50 attributes), Store (20 attributes), Time (ten attributes) and Promotion (ten attributes) [1], [8]. However, the number of possible materialized views is over 2^{90} . The search space for possible materialized views is extremely large. Existing algorithms can achieve an optimal solution only when the problem size is small but cannot tackle large-scale problems.

Computational intelligence plays a significant role in supporting the design of intelligent systems [9], [10]. Hence, computational intelligence is highly desirable to assist the design of a data warehouse system as a network service that collects data from different remote data sources and disseminates high-quality data analysis to decision makers locally and remotely in an efficient way. Zhang *et al.* [11] proposed an evolutionary approach to materialized view selection, but they did not consider any constraints. Lee and Hammer [12] made the first attempt to solve the maintenance-cost view-selection problem using evolutionary algorithms. They tested nine different ways to add a penalty function to the original objective function. However, their results were less satisfactory. They did not show any results for problems larger than 20 views, even though they mentioned that they did try to tackle large problems.

In this paper, we propose a new constrained evolutionary algorithm for the maintenance-cost view-selection problem. Our algorithm does not use any penalty functions. Instead, a novel stochastic ranking procedure is used. This is the first time that

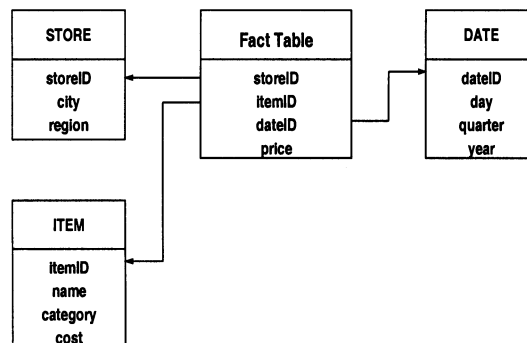


Fig. 1. Multidimensional data warehouse with three dimensions: Store, Item, and Date.

the stochastic ranking is used to solve a combinatorial problem. Through extensive experimental studies, we found that the feasible solutions can be easily found by our stochastic ranking approach. In addition, our new constrained evolutionary algorithm explores the search space better than the other existing algorithms. It can scale well with the problem size.

The rest of this paper is organized as follow. Section II discusses the maintenance-cost view-selection problem and defines a general cost model. Constraint handling and our evolutionary algorithm are discussed in Section III. We conduct extensive performance studies and report the results in Section IV. We conclude the paper in Section V.

II. THE PRELIMINARIES

A. A Multidimensional Data Warehouse

The star-schema of an m -multidimensional data warehouse consists of a *fact table* and a collection of *dimension tables* [1]. A fact table consists of m dimensions, denoted D_1, D_2, \dots, D_m , along with measures of interest, denoted M . Each value in the D_i dimension of the fact table corresponds to a unique record in the corresponding D_i dimension table, where all the details about that dimension D_i are kept. In a dimension table, attributes can be further organized in a hierarchy structure. Suppose that a multidimensional data warehouse has m dimensions and the i -th dimension has m_i attributes. There are $\prod_i^m (2^{m_i} + 1)$ possible OLAP queries (SQL group-by queries), or views.

Fig. 1 shows a star-schema for a multidimensional data warehouse of three dimensions: STORE, ITEM and DATE. There is a fact table and three dimension tables. In the fact table, it keeps the three dimension identifiers, *storeID*, *itemID* and *dateID*, for the three dimension tables, along with the measure of interest, *price*. The dimension table STORE has two attributes, *city* and *region*, in addition to its record identifier *storeID*. The dimension table ITEM has three attributes, *name*, *category* and *cost*, in addition to its record identifier *itemID*. In a similar fashion, the dimension table DATE has four attributes, *day*, *month*, *quarter* and *year*, along with the record identifier *dateID*. These dimension tables keep the detail information about the dimension. All the fact table and the three dimension tables can be joined to a large table representing a multidimensional space. The total number of OLAP queries is then $765 = (2^2 + 1) \times (2^3 + 1) \times (2^4 + 1)$.

¹ f is the fraction available space consumed by the largest aggregate.

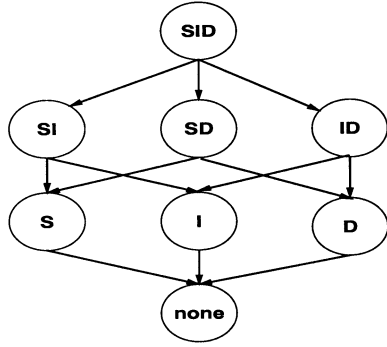


Fig. 2. Example of dependent lattice.

B. The Maintenance-Cost View Selection Problem

Harinarayan *et al.* [2] introduced a dependent lattice whose vertices are the OLAP queries or views and edges represent the dependencies among the OLAP queries. Like [2], we define a *dependent lattice*, (L, \preceq) , with a set of elements (queries or views) L and a dependence relation \preceq (derived-from, be-computed-from). Given two queries q_i and q_j . We say q_i is dependent on q_j , ($q_i \preceq q_j$), if q_i can be answered using the results of q_j . A dependent lattice can be represented as a directed acyclic graph, $G = (V, E)$. Here V represents the set of queries, as vertices. We use $V(G)$ and $E(G)$ for the set of vertices and the set of edges of a graph G . An edge, $v_i \rightarrow v_j$, exists in E , if and only if $v_j \preceq v_i$ and $\exists v_k (v_j \preceq v_k \wedge v_k \preceq v_i)$, for $v_i \neq v_k \neq v_j$.

Fig. 2 illustrates a simple dependent lattice of three dimensions, where S, I and D represent storeID, itemID and dateID, respectively, for the example shown in Fig. 1. Here, we ignore all the details in the dimension tables, and only consider the identifiers as representatives of the dimensions. A vertex in Fig. 2 represents an OLAP-query or a view. For example, S represents an OLAP query which is interested in the total income price in each store. And SI represents an OLAP query which is interested in the total income price in each store with each item. The edge from the vertex SI to the vertex S represents the fact that the OLAP query S can be processed by the OLAP query SI. Note: in Fig. 2, none represents an OLAP query which is interested in the total income price from all dimensions.

In addition, the graph representation, $G = (V, E)$, for the dependent lattice, has the following weights associated with vertices and edges.

- Three weights on a vertex $v \in V(G)$:
 - r_v : initial data scan cost'
 - f_v : query frequency;
 - g_v : update frequency.
- Two weights on an edge $(v, u) \in E(G)$:
 - $w_{q_u, v}$: query processing of u using v ;
 - $w_{m_u, v}$: updating cost of u using v .

It is important to know that $r_u \geq r_v$ if $v \preceq u$. An additional vertex, v_+ , is introduced into the directed acyclic graph as the virtual root (representing the multidimensional data warehouse) such that, for all $v_i \in V(G)$, $v_i \preceq v_+$. Note: $v_+ \notin V(G)$. The data size of the virtual root, r_+ , is the largest among all the data sizes.

In a general setting, let $q(u, v)$ denote the query processing cost of answering a query u using a selected materialized view v . $q(u, v)$ is the sum of query processing costs associated with edges on the shortest path from v to u plus the initial data scan cost of the vertex v , r_v . If view v cannot answer query u in $q(u, v)$, the raw table, the virtual vertex v_+ , will be used instead of v . Similarly, $m(u, v)$ denotes the maintenance cost which is the sum of the maintenance-costs associated with the edges on the shortest path from v to u . In [2], a linear cost model was proposed. The linear cost model states that the cost of answering a query using a view is the number of rows present in the view. We attempt to adopt a more general cost model than this linear cost model. Here, as shown by the two functions $q()$ and $m()$, we assume a general query processing cost and maintenance cost model. First, a query processing cost can be different from a maintenance cost for a pair of vertices. Second, we also assume that the query processing cost may involve other query processing costs (associated with edges) in addition to the initial table scan costs (associated with vertices). Third, there are multiple paths from a view to a query. In our setting, we consider selection of the shortest path.

Let $M (\subseteq V(G))$ be a set of vertices to be selected as materialized views. Furthermore, let $q(v, M)$ denote the minimum cost of answering a query $v (\in V(G))$ in the presence of the set of materialized views (M) , and $m(v, M)$ be the minimum cost of maintaining a materialized view $v (\in V(G))$ in presence of the set of materialized views (M) . The maintenance-cost view-selection problem is to select a set of views M that minimizes $\tau(G, M)$, where

$$\tau(G, M) = \sum_{v \in V(G)} f_v \cdot q(v, M)$$

under the constraint that $U(M) \leq S$, where, $U(M)$, the total maintenance cost is defined as

$$U(M) = \sum_{v \in M} g_v \cdot m(v, M).$$

C. The Difficulty of Maintenance-Cost View-Selection Problem

In many real applications, maintenance-cost is more likely to be the real constraint to keep the materialized views consistent with the data in data warehouse, rather than disk-space constraints. The maintenance-cost view-selection problem seems to be very similar to the disk-space view-selection problem. However, the maintenance-cost view-selection problem is more difficult. For the maintenance-cost view-selection problem, the maintenance cost of the views relies on each other. Selection of a view will affect the prior materialized views. The total maintenance cost for a set of views may decrease when more views are added to materialize while the space occupied by a set of views always increases when a new view is selected under the disk-space constraint. Fig. 3 illustrates the difference between the disk-space view-selection problem and maintenance-cost view-selection problem. Here, for a vertex, v_i , T, and u are table size (r_{v_i}) (for the query processing cost) and maintenance cost ($w_{q_u, v}$), respectively. For simplicity, we assume that the query frequency (f_v) and update frequency (g_v) are the same for every

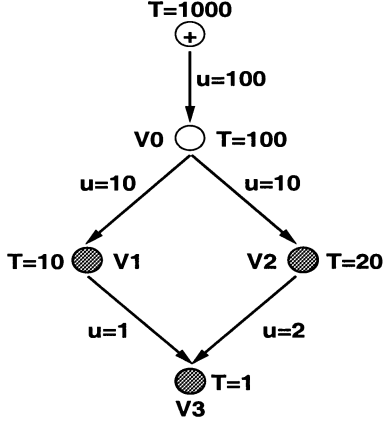


Fig. 3. Example of view maintenance.

vertex in this example. Suppose $M = \{v_3, v_1, v_2\}$ are materialized in an order of v_3 and v_1 followed by v_2 . The total disk-space used is $1 + 10 + 20 = 31$ and the total maintenance-cost is $1 + 100 + 100 = 201$, because v_1 and v_2 need to be computed from the virtual root and v_3 is answered by v_1 . Now consider materializing v_0 . The total disk-space used is increased to $1 + 10 + 20 + 100 = 131$, and the total maintenance cost is decreased to $1 + 10 + 10 + 100 = 121$, because v_1 and v_2 now can be updated by v_0 . This nonmonotonic property makes maintenance-cost view-selection very difficult.

D. An A*-Heuristic Algorithm

Gupta and Mumick [6] proposed an A*-heuristic algorithm to solve the maintenance-cost view-selection problem and claimed that the A*-heuristic can guarantee to reach an optimal solution. The A*-heuristic is shown by Algorithm 1. The A*-heuristic uses an *inverse topological order* to find a set of materialized views. It defines a binary tree T_G whose leaf vertices are the candidate solutions of this problem. At each stage of searching, A*-heuristic evaluates the benefit of remaining downward branches, and selects the branch of the greatest benefit to go down. Each vertex in the binary search tree has a label $\langle N_x, M_x \rangle$ ($M_x \subseteq N_x$), where M_x is the set of views which have been chosen to materialize and considered to answer the set of queries N_x . The search space is $2^{|V(G)|}$, where $V(G)$ is the set of vertices of the graph G . They estimated the benefit of the downward branches by summing up two functions $g(x)$ and $h(x)$. $g(x)$ is the total query processing cost of the queries on N_x using the selected views in M_x . $h(x)$ is an estimated lower bound on $h^*(x)$ which is defined as the remaining query cost of an optimal solution corresponding to some descendant of x in T_G [6].

Although the A*-heuristic can guarantee to find an optimal solution, it is an exponential algorithm in the worst case and may take a prohibitively long time to run. In this paper, we will examine the quality and scalability of our algorithm in comparison with the A*-heuristic, and report our findings in Section IV.

Algorithm 1 A*-Heuristic [6]

Input: A graph $G(V, E)$ and a maintenance-cost constraint S .

Output: a set of materialized views.

- 1: **begin**
- 2: Create a tree T_G having just the root
 - A. The label associated with A is $\langle \phi, \phi \rangle$.
- 3: Create a priority queue (heap) $L = \langle A \rangle$
- 4: **repeat**
- 5: Remove x from L , where x has the lowest $g(x) + h(x)$ value in L
- 6: Let the label of x be $\langle N_x, M_x \rangle$, where $N_x = \{v_1, v_2, \dots, v_d\}$ for some $d \leq n$.
- 7: **if** $d = n$ **then**
- 8: **return** M_x
- 9: **end if**
- 10: Add a successor of x , $l(x)$, with a label $\langle N_x \cup \{v_{d+1}\}, M_x \rangle$ to the list L .
11. **if** $(U(M_x) < S)$ **then**
12. Add to L a successor of x , $r(x)$, with a label $\langle N_x \cup \{v_{d+1}\}, M_x \cup v_{d+1} \rangle$
13. **end if**
14. **until** (L is empty);
15. **return** \emptyset ;
16. **end**

III. EVOLUTIONARY ALGORITHMS

Evolutionary computation techniques have received a great attention [13]. Some evolutionary algorithms were proposed to solve the maintenance-cost view-selection problem, because of its robustness. [11] first proposed an evolutionary approach to materialized view selection problem without considering any constraints. [12] made the first attempt to solve the maintenance-cost view-selection problem by evolutionary algorithms, but did not show any experiments for problems larger than 20 views.

In this paper, we propose a new evolutionary algorithm which fits the maintenance-cost view-selection problem well. First, a pool of bit string genomes are generated randomly. This is the initial population. Each genome represents a candidate solution to the problem to be solved. The length of this genome is the total number of vertices in the lattice; 1 and 0 mean that the vertices need to be materialized or not respectively. A genome can be formalized as $\text{genome} = (x_1 x_2 x_3 \dots x_N)$, where N is the total number of vertices in the lattice. Here, $x_i = 1$ if view v_i is selected for materialization and $x_i = 0$ if view v_i is not selected for materialization. For example, in Fig. 3, N is 4. $\text{genome} = (1 0 1 0)$ means that two views, v_0 and v_2 , are materialized. During the crossover and mutation processes, good candidates will survive and poor candidates will die. In the following, we will introduce penalty methods and stochastic ranking, and give our new evolutionary algorithm.

A. Constraint Handling: Penalty versus Stochastic Ranking

Lee and Hammer in [12] used a genetic algorithm with the penalty method to set a static penalty coefficient, r_g , to find

a near-optimal solution to the maintenance-cost view-selection problem. (Hereafter we called it as LEE algorithm.) In brief, we introduce their penalty-based approaches, and express our concerns.

Let $x = (x_1, x_2, \dots, x_N)$, for $x_i = 0$ or 1 , and $M_x = \{v_i | x_i = 1, i = 1, 2, \dots, N\}$, the original maintenance-cost view-selection problem can be formulated as follows:

$$\begin{aligned} & \text{Maximize } f(x) = B(G, M_x) = \tau(G, \phi) - \tau(G, M_x) \\ & \text{subject to : } U(M_x) \leq S. \end{aligned}$$

This is a constrained combinatorial optimization problem. The common method for dealing with constrained optimization problems is to introduce a penalty function to the objective function to penalize the solutions violating the constraint. Usually, the penalty function can be defined as

$$\phi(x) = \max \{U(M_x) - S, 0\}.$$

Then, the original optimization problem with constraints can be transformed into an unconstrained one:

$$\text{Maximize } f(x) = B(G, M_x) - r_g \cdot \phi(x)$$

where r_g is the penalty coefficient. The choice of the penalty coefficient r_g is very important [14]. A too small r_g will result in under-penalization, namely, infeasible solutions not being penalized enough. So many infeasible solutions may be found. A too large r_g will result in over-penalization, namely, some “beneficial” infeasible solutions being penalized too much during the course of evolutionary optimization. The reason why some infeasible solutions may be beneficial during the course of evolution is that, when feasible regions in the whole search space are disjoint, some infeasible regions may act as bridges among feasible regions. If a too large r_g makes such infeasible solutions inaccessible, then it is difficult for an evolutionary algorithm to jump from one feasible region to another one which may have better fitness values. Thus, an overly large r_g may prevent a good feasible solution from being found. Because of the importance of r_g , there has been much research work done on it. However, the setting of r_g has been a difficult problem. It is difficult to find a precise value to realize the right balance between the original objective function and the penalty function. Even most dynamic setting methods, which start with a low r_g value and end with a high r_g , are not likely to work well for problems for which the unconstrained global optimum is far away from its constrained one [14]. In [12], the fitness function $f(x)$ has three forms:

Subtract mode(S)

$$\begin{aligned} f(x) &= B(G, M_x) - \text{Pen}(x), \quad \text{if } B(G, M_x) - \text{Pen}(x) \geq 0, \\ &= 0, \quad \text{otherwise.} \end{aligned}$$

Divide mode(D)

$$\begin{aligned} f(x) &= \frac{B(G, M_x)}{\text{Pen}(x)}, \quad \text{if } \text{Pen}(x) > 1, \\ &= B(G, M_x), \quad \text{if } \text{Pen}(x) \leq 1. \end{aligned}$$

Subtract and Divide mode(SD)

$$\begin{aligned} f(x) &= B(G, M_x) - \text{Pen}(x), \quad \text{if } B(G, M_x) > \text{Pen}(x), \\ &= \frac{B(G, M_x)}{\text{Pen}(x)}, \quad \text{if } B(G, M_x) \leq \text{Pen}(x) \\ & \quad \text{and } \text{Pen}(x) > 1, \\ &= B(G, M_x), \quad \text{if } B(G, M_x) \leq \text{Pen}(x) \\ & \quad \text{and } \text{Pen}(x) \leq 1. \end{aligned}$$

Their penalty functions² also have three forms:

- Logarithmic penalty (LG):

$$\text{Pen}(x) = \log_2 (1 + \rho \cdot (U(M_x) - S)).$$

- Linear penalty (LN):

$$\text{Pen}(x) = 1 + \rho \cdot (U(M_x) - S).$$

- Exponential penalty (EX):

$$\text{Pen}(x) = (1 + \rho \cdot (U(M_x) - S))^2.$$

Whichever of the three fitness function forms above is used in practice, this can be considered as a penalty method of a static r_g value. We note that in the *subtract mode*, in fact, $r_g = 1$. In the other two modes, *D* and *SD*, although no explicit r_g values are shown, they are fixed. The unconstrained fitness function $f(x)$ is composed of $B(G, M_x)$ and $\text{Pen}(x)$, and this relation does not change in the whole evolutionary process. As it does in numerical function optimization problems, such a penalty method does not work very well in combinatorial optimization problems either. We will compare its experiment results with ours in Section IV.

Since finding an optimal r_g value is difficult and the penalty methods setting a static or dynamic r_g value do not work well for the optimization problem with constraints, [14] put forward a new constraint handling technique, named stochastic ranking, to balance the dominance of the objective and penalty functions for constrained *numerical* optimization. The novel idea of this technique is the introduction of a probability P_f for rank-based selection. During the course of ranking, We need to compare pairs of two adjacent individuals. If they are both feasible solutions, naturally, we will compare them according to the objective function. However, when either of them is infeasible, the probability of comparing them according to the objective function is P_f , while the probability of comparing them according to the penalty function will be $1 - P_f$. Since P_f is a probability, it gives an opportunity for both the objective and penalty functions to rank a pair. When $P_f > 1/2$, the ranking is biased toward the objective function. When $P_f < 1/2$, the ranking is biased toward the penalty function. So P_f can balance the objective and penalty functions more directly, explicitly and conveniently. By adjusting P_f , we can adjust the balance between the objective function and the penalty function easily. Moreover, we do not have any extra computing cost for setting r_g values since we do not use any penalty terms. In practice, we usually set $P_f < 1/2$ to reduce the ratio of infeasible solutions to the whole in the final generation. For different optimization problems, we will experiment with setting different P_f values.

²EX should really be called “polynomial”, but let us use what the authors used.

B. Our New Stochastic Ranking Evolutionary Algorithm

Based on our analysis in the last section, we observe that the stochastic ranking approach will have better performance for this problem than the LEE method. Although stochastic ranking has been used for constrained numerical optimization problems and shown good performance using (μ, λ) evolution strategy [14], [15], it is unclear whether it is effective for combinatorial optimization problems. Our paper presents the first attempt toward generalizing this approach to combinatorial optimization problems, using a operator sequence, crossover-mutation-selection, as used in generic algorithms.

The basic framework of our evolutionary algorithm is shown in Algorithm 2. Similar to most evolutionary algorithms, both crossover and mutation are used. The crossover operator we use is *uniform crossover*, as shown in Algorithm 3. It exchanges the information of two chromosomes to generate two new chromosomes. The mutation operator we use is similar to the most usually used one, as shown in Algorithm 4. The probability that every bit of every gene will be flipped is P_m . The key difference from most evolutionary algorithms is the stochastic rank procedure we used. The stochastic ranking algorithm is based on [14], but modified in some places for the specific problem of materialized views selection, as shown in Algorithm 5. It is used for ranking the union of new and old individuals. The ranking procedure is similar to bubble-sort. In every sweep of N , every two adjacent individuals are compared. If there is no any change of individual's rank after a sweep, then this bubble-sort-like procedure can be terminated. Note: N is the number of vertices in the lattice.

It is worth noting that, for dealing with numerical optimization problems, [14] uses a (μ, λ) evolution strategy, and set the truncation level as $\mu/\lambda \approx 1/7$, where μ and λ are the number of parents and the number of children, respectively. In this paper, we are dealing with the materialized view selection problem as a combinatorial optimization problem using a typical operator-sequence as used in genetic algorithms. Like most genetic algorithms, we generate λ offsprings from μ parents, where $\lambda = \mu$.

IV. EXPERIMENTAL STUDIES

In this section, we present some results of our experimental study. All the algorithms were implemented using `g++`. These experiments were done on a Sun Blade/1000 workstation with a 750 MHz UltraSPARC-III CPU running Solaris 2.8. The workstation has a total physical memory of 512 M.

A. Experimental Setup

In order to evaluate the performance of our stochastic ranking evolutionary algorithm (EA) and the best result of the penalty-based algorithm (LEE), we also implemented an algorithm for finding the optimal solution. To find the optimal

Algorithm 2 The Basic Framework of Our Evolutionary Algorithm (denoted EA)

Parameter: population size P

```

1: begin
2: Generate the initial population  $G(0)$ ;
3: repeat

```

```

4:  $t = t + 1$ ;
5:  $G_1(t) \leftarrow \text{UniformCrossover}(G(t - 1))$ ;
   {refer to Algorithm 3.}
6:  $G_2(t) \leftarrow \text{Mutation}(G_1(t))$ ; {refer to
   Algorithm 4.}
7:  $S \leftarrow \text{StochasticRanking}(G(t - 1) \cup G_2(t))$ ,
   which sorts  $G(t-1) \cup G_2(t)$  to an ordered
   individuals sequence  $S$  of
   size  $2 \times P$ ; {refer to Algorithm 5.}
8:  $G(t) \leftarrow$  the anterior  $P$  individuals of  $S$ ;
9: until (termination condition is
   satisfied)
10: end

```

Algorithm 3 UniformCrossover

Input: Generation G

Parameter: crossover probability P_c

```

1: begin
2: Select a pair of individuals of  $G$ 
   randomly:
    $g_1 = (b_1, b_2, \dots, b_N)$ ,  $g_2 = (c_1, c_2, \dots, c_N)$ ;
3: sample  $u \in U(0, 1)$ ;
4: if ( $u < P_c$ ) then
5:   for every bit  $i$  of individual do
6:     sample  $r$ , either 0 or 1;
7:     if ( $r = 1$ ) then
8:       the bit  $i$  of  $g_1' = b_i$ ;
9:       the bit  $i$  of  $g_2' = c_i$ ;
10:    else
11:      the bit  $i$  of  $g_1' = c_i$ ;
12:      the bit  $i$  of  $g_2' = b_i$ ;
13:    end if
14:  end for
15: else
16:   $g_1' = g_1$ ;
17:   $g_2' = g_2$ ;
18: end if
19: Repeat the above procedure  $P/2$ 
   times, which will generate  $P$  new
   individuals;
20: end

```

Algorithm 4 Mutation

Input: Generation G

Parameter: mutation probability P_m

```

1: begin
2: for every individual in  $G$  do
3:   for every bit in the individual do
4:     Mutate the bit with the probability
       of  $P_m$ ;
5:   end for
6: end for
7: end

```

Algorithm 5 Stochastic Ranking

Input: $\lambda = 2 \times P$ individuals $\{I_j | j = 1, \dots, \lambda\}$

Parameter: balance parameter P_f
 Note: the fitness function: $f(x) = B(G, M_x)$, the penalty function: $\phi(x) = \max\{U(M_x) - S, 0\}$. The N is set to be λ as analyzed in [14].

```

1: for  $i = 1$  to  $N$  do
2:   for  $j = 1$  to  $\lambda - 1$  do
3:     sample  $u \in U(0, 1)$ ;
4:     if  $(\phi(I_j) = \phi(I_{j+1}) = 0)$  or  $(u < P_f)$ 
       then
5:       if  $f(I_j) < f(I_{j+1})$  then
6:         swap( $I_j, I_{j+1}$ );
7:       end if
8:     else
9:       if  $\phi(I_j) > \phi(I_{j+1})$  then
10:        swap( $I_j, I_{j+1}$ );
11:      end if
12:    end if
13:  end for
14:  if no swap done then
15:    break;
16:  end if
17: end for

```

Set of materialized views to precompute, we enumerated all possible combinations of views, and find a set of views by which the query processing cost is minimized. Its complexity is $O(2^N)$, where N is the number of vertices. For a small number of lattice (16 vertices), we compare among EA, LEE, A*-heuristic and the optimal algorithm. We also report the scalability of the EA algorithm using a large number of lattice up to 256 vertices. In the following, LEE is the best result given in [12].

Table I summarizes all the parameters together with the default values used in our experiments.

Given a dependent lattice (L, \prec) of size N , we construct a directed acyclic graph $G(V, E)$. A vertex, v , has three weights: table size R_v , update frequency θ_u and query frequency θ_q . An edge, from v to u , has two weights: $Q_{(u,v)}$ and $U_{(u,v)}$. We assign these weights to the graph $G(V, E)$ as follows. First, we randomly generate N distinctive table sizes (R_v). The N table sizes are randomly picked up and assigned to the vertices on a condition that the table sizes of ancestors of a vertex are greater than that of the vertex. We assume that query frequencies follow a Zipf distribution, the high query frequencies are most likely to be assigned at the high level (close to top) by default. We also assume that, when the raw table is updated, all views need to be recomputed. Thus, all vertices are assumed to have the same update frequency. Given an edge from v to u , (v, u) , we assume that the maintenance-cost of u using v is smaller than the query processing cost of u using v . We also assume that the maintenance-cost is more related to the table size of u . In the set of tests we reported in this paper, $Q_{(v,u)}$ is a number smaller than the table size of v , (R_v). $U_{(v,u)}$ is about one tenth of the table size u , (R_u). The maintenance-cost constraint is a crucial condition for the maintenance-cost view-selection problem. In our experiments, we assume that the minimum maintenance-cost constraint, C_{min} , is the minimum value which allows all views to be selected as materialized views.

TABLE I
 NOTATIONS AND DEFINITIONS OF THE SYSTEM PARAMETERS
 USED IN EXPERIMENTS

Notation	Definition (Default Values)
N	the number of vertices (16)
θ_q	Zipf distribution factor for query frequency (0.2)
θ_u	Zipf distribution factor for update frequency (0.0625)
R_v	table size for a vertex v
$Q_{(v,u)}$	query processing cost for a vertex u using v
$U_{(v,u)}$	maintenance-cost for a vertex u using v
C_{min}	the minimum maintenance-cost constraint that allows all vertices to be selected as materialized views
P_f	probability for stochastic ranking function (0.4)
P_m	mutation probability (0.001)

B. Experimental Results

1) *Feasibility of the Solutions:* First, we investigate the feasibility of the solutions of our EA by varying the P_f value. In Fig. 4(a)–(d), the number of vertices is 32. The results were averaged over 30 independent runs of our EA algorithm. In Fig. 4(a), the y-axis indicates the percentage of feasible solutions in the final generation. Recall that maintenance-cost constraint has a big effect on the result. In this testing, we try to use different maintenance-cost constraint to see how P_f deals with the maintenance-cost constraint. In Fig. 4(a), $C_{min}(1)$ and $C_{min}(0.8)$ represent the maintenance-cost constraint $1 \times C_{min}$ and $0.8 \times C_{min}$, respectively. When the maintenance-cost constraint is $C_{min}(1)$, the percentage of feasible solution is always one hundred. It shows that EA can always find a feasible solution if the maintenance-cost constraint is large enough. When the maintenance-cost constraint is $C_{min}(0.8)$, it shows that P_f can alter the percentage of feasible solutions very easily. When $P_f = 0.5$, the percentage of feasible solutions drops sharply from 100% to 0%. If maintenance-cost constraint is reduced to $0.5 \times C_{min}$, the EA gets all 0s solutions in the final generation since the maintenance-cost constraint is too low to select any vertices.

In Fig. 4(b)–(d), the optimal solution produced by A*-heuristic is chosen as the denominator to evaluate our EA. In these three figures, the maintenance-cost constraint is $0.8 \times C_{min}$. Fig. 4(b) shows the quality of the feasible solutions. The y-axis represents a ratio of the average query processing cost of the feasible solutions over the optimal query processing cost. As expected, when P_f is less than or equal to 0.4, the average query processing cost of feasible solutions is greater than 1, because the query processing cost of the optimal solution is the lowest among all the feasible solutions. In contrast, when $P_f > 0.4$, the average query processing cost of feasible solutions is equal to 0 as there are no feasible solutions found. (Fig. 4(a) shows that the percentage of feasible solution is equal to 0 when $P_f > 0.4$.)

Fig. 4(c) shows the quality of the infeasible solutions in the final generation. Since the infeasible solutions trade off the maintenance-cost with a lower and better overall query processing cost, the average query processing cost of infeasible solutions is less than 1. Fig. 4(d) shows the maintenance-cost of the infeasible solutions from the optimal maintenance cost. It shows that in the worst case, the average maintenance-cost of infeasible solutions is no greater than 1.3 times of the maintenance-cost of the optimal solution.

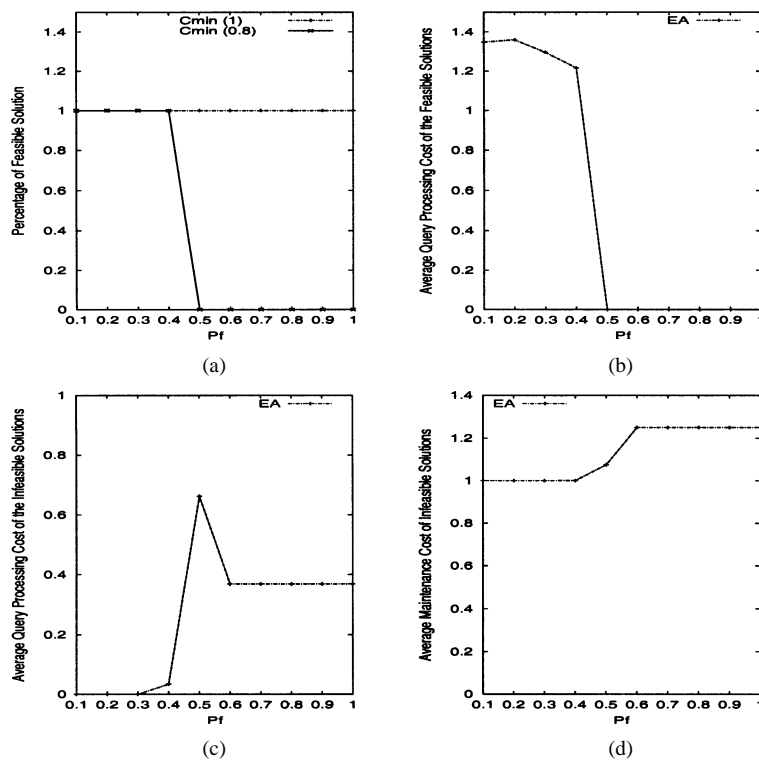


Fig. 4. Feasibility of the solutions by varying the P_f value. (a) P_f versus percentage of feasible solutions; (b) P_f versus average query processing cost of feasible solutions; (c) P_f versus average query processing cost of infeasible solutions; (d) P_f versus average maintenance cost of infeasible solutions.

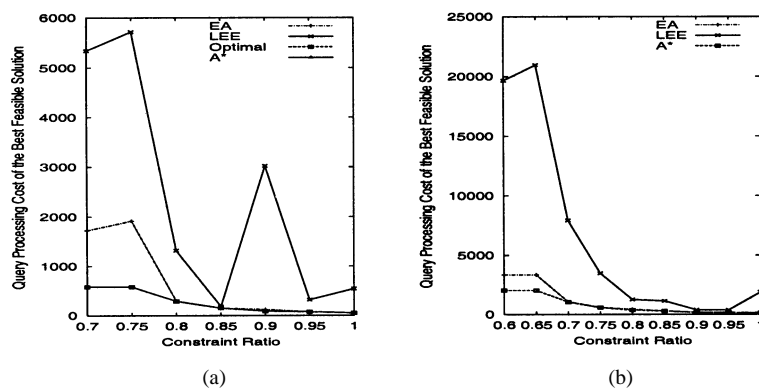


Fig. 5. Optimality of solutions with different maintenance-cost constraint. (a) Query processing cost versus maintenance-cost constraint (16 vertices); (b) query processing cost versus maintenance-cost constraint (32 vertices).

The above testings demonstrate that P_f gives a convenient way to fine-tune the algorithm. By varying the P_f value, EA can deal with the maintenance cost constraint well. As a result, we will choose $P_f = 0.4$ as the default P_f value in the subsequent experiments.

2) *Optimality of Solutions:* In this experimental study, we investigate the performance of our EA, LEE, A*-heuristic and the optimal algorithm under different maintenance-cost constraints. Let the maintenance-cost constraint be $R \times C_{min}$. In Fig. 5(a) and (b), R varies from 0.7 to 1. (Note that when $R < 0.7$, none of the algorithms can select any views.) A larger R value implies that it is likely to select more views. When $R = 1$, it means that all vertices may be selected. The number of vertices is 16 and 32 respectively in Fig. 5(a) and (b). We took the average query processing costs of our EA and LEE over 30 independent runs.

In Fig. 5(a), we use exhaustive search to compute the optimal solution. It shows that A*-heuristic performs in the same way as the optimal. Our EA always gives a near optimal feasible solution that is very close to the optimal. On the other hand, the query processing cost of LEE is much higher than the optimal solution.

In Fig. 5(b), we compare our EA and LEE with A*-heuristic. It shows that our EA can find near optimal feasible solutions that are very closed to A*-heuristic. Our EA outperforms the LEE algorithm significantly.

3) *Scalability of the Algorithms:* There are several existing algorithms for solving the maintenance-cost view-selection problem. Fig. 6 shows four algorithms, namely, LEE, EA and A*-heuristic, in addition to a greedy algorithm, called inverted-tree [6], when the maintenance-cost constraint is $0.8 \times C_{min}$. The inverted-tree greedy uses a concept

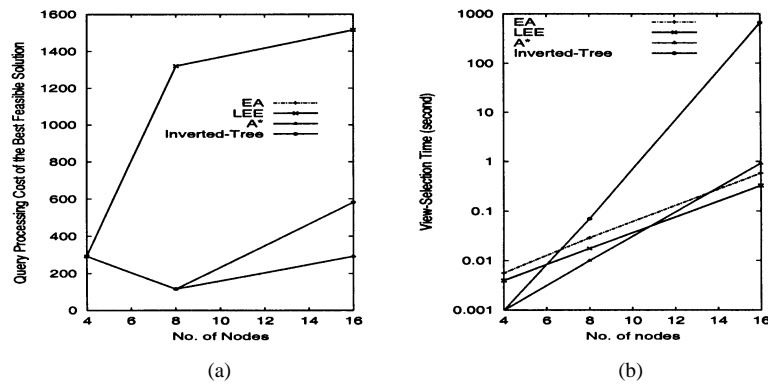


Fig. 6. Four algorithms. (a) Query processing cost versus number of vertices; (b) view selection time versus number of vertices.

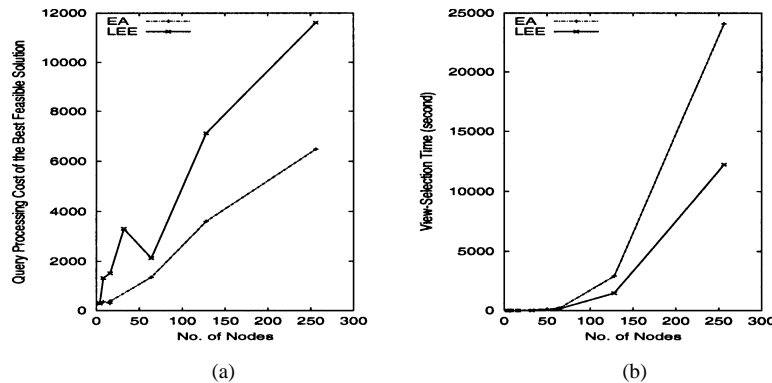


Fig. 7. Scalability of algorithm by varying the number of vertices. (a) Query processing cost versus number of vertices; (b) view selection time versus number of vertices.

called an inverted tree set. Given a vertex v in a directed graph, an inverted tree set contains the vertex v and any subset of vertices reachable from v . At each stage, the inverted-tree greedy algorithm considers all inverted tree sets of views in the given graph, and selects the inverted tree set that has the most query-benefit per unit effective maintenance-cost. Fig. 6 shows a small-scale problem with the number of vertices from four to 16. As shown in Fig. 6(a), EA and A*-heuristic performs the best (the same as the optimal). The inverted-tree greedy is inferior to EA and A*-heuristic but is superior to LEE. Fig. 6(b) shows the view-selection time. The inverted-tree greedy cannot deal with large-scale problems, due to its view selection time.

The existing algorithms do not perform well when computing a large dependent lattice. Evolutionary algorithms can explore this search space better. Since A*-heuristic and inverted-tree greedy cannot deal with the lattice up to 256, we compare our EA and LEE by varying the number of vertices, N , from 4 to 256. The maintenance-cost constraint is $0.8 \times C_{min}$. For the number of vertices from four to 64, we took the average query processing cost for both algorithms over 30 independent runs. When the number of vertices is greater than 128, we ran it once due to the longer execution time. In Fig. 7(a), we can see that EA significantly outperforms the LEE algorithm in terms of minimization of query processing cost. However, our EA took a longer time than LEE to find better solutions, according to Fig. 7(b). It is worth noting that our EA is much more likely to find feasible solutions as well while LEE tends to get stuck at a poor solution fairly early.

V. CONCLUSIONS

As a network service, a data warehouse system collects data from different remote data sources and disseminates high-quality data analysis to decision makers locally and remotely. In this paper, we showed that computational intelligence plays a significant role in design of a data warehouse system, and presented a new constrained evolutionary algorithm for the maintenance-cost view-selection problem.

The algorithm is based on a novel constraint handling technique—stochastic ranking. Although stochastic ranking has been used in numerical constrained optimization, its suitability for combinatorial optimization was unclear. This paper demonstrates that a revised stochastic ranking scheme can be applied to constrained combinatorial optimization problems successfully.

We have evaluated our new evolutionary algorithm against both heuristic and other evolutionary algorithms. Our experiments results show that our algorithm can provide significantly better solutions than previous algorithms in terms of minimization of query processing cost and feasibility. In comparison with the latest evolutionary algorithm, i.e., the LEE algorithm [12], our algorithm can avoid premature convergence and keep improving the solution, while the LEE algorithm tends to get stuck at a poor local optimum fairly early.

ACKNOWLEDGMENT

The authors appreciate the editors and anonymous referees for their invaluable suggestions and comments which help us improve the paper's quality and presentation.

REFERENCES

- [1] R. Kimball, *The Data Warehouse Toolkit*. New York: Wiley, 1996.
- [2] V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Implementing data cubes efficiently," in *Proc. 1996 ACM SIGMOD Int. Conf. Management of Data*, 1996, pp. 205–216.
- [3] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Index selection for OLAP," in *Proc. Thirteenth Int. Conf. Data Engineering*, 1997, pp. 208–219.
- [4] H. Gupta, "Selection of views to materialize in a data warehouse," in *Proc. 6th Int. Conf. Database Theory*, 1997, pp. 98–112.
- [5] A. Shukla, P. Deshpande, and J. F. Naughton, "Materialized view selection for multidimensional datasets," in *Proc. 24th Int. Conf. Very Large Data Bases*, 1998, pp. 488–499.
- [6] H. Gupta and I. S. Mumick, "Selection of views to materialize under a maintenance cost constraint," in *Proc. 7th Int. Conf. Database Theory*, 1999, pp. 453–470.
- [7] C.-H. Choi, J. X. Yu, and G. Gou, "What difference heuristics make: Maintenance-cost view-selection revisited," in *Proc. Third Int. Conf. Web-Age Information Management*, 2002.
- [8] E. Baralis, S. Paraboschi, and E. Teniente, "Materialized views selection in a multidimensional database," in *Proc. 23rd Int. Conf. Very Large Data Bases*, 1997, pp. 156–165.
- [9] J. C. Bezdek, "What is computational intelligence?," in *Computational Intelligence Imitating Life*. New York: IEEE Press, 1994, pp. 1–12.
- [10] J. M. Zurada, R. J. M II, and C. J. Robinson, *Computational Intelligence Imitating Life*. New York: IEEE Press, 1994.
- [11] C. Zhang, X. Yao, and J. Yang, "An evolutionary approach to materialized view selection in a data warehouse environment," *IEEE Trans. Syst., Man, Cybern. C*, vol. 31, pp. 282–294, Aug. 2001.
- [12] M. Lee and J. Hammer, "Speeding up materialized view selection in data warehouses using a randomized algorithm," *Int. J. Cooperative Inform. Syst.*, vol. 10, no. 3, pp. 327–353, 2001.
- [13] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evol. Comput.*, vol. 4, no. 1, pp. 1–32, 1996.
- [14] T. P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 4, pp. 284–294, Sept. 2000.
- [15] H.-P. Schwefel, *Evolution and Optimum Seeking*. New York: Wiley, 1995.

Jeffrey Xu Yu received the B.E., M.E., and Ph.D. in computer science from the University of Tsukuba, Japan, in 1985, 1987, and 1990, respectively.

He was a Research Fellow (April 1990–March 1991) and an Assistant Professor (April 1991–July 1992) with the Institute of Information Sciences and Electronics, University of Tsukuba, and a Lecturer in the Department of Computer Science, Australian National University, Canberra (July 1992–June 2000). Currently, he is an Associate Professor in the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong. His major research interests include wireless information retrieval, data warehouse, on-line analytical processing, query processing and optimization, and design and implementation of database management systems.

Dr. Yu is a member of ACM and a society affiliate of IEEE Computer Society.



Xin Yao (SM'96–F'02) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, in 1982, the M.Sc. degree from the North China Institute of Computing Technology, Beijing, in 1985, and the Ph.D. degree from USTC in 1990.

He was an Associate Lecturer and Lecturer between 1985 and 1990 at USTC while pursuing the Ph.D. degree. He took up a postdoctoral fellowship in the Computer Sciences Laboratory, Australian National University (ANU), Canberra, in 1990, and continued his work on simulated annealing and evolutionary algorithms. He joined the Knowledge-Based Systems Group at CSIRO Division of Building, Construction and Engineering, Melbourne, Australia, in 1991, working primarily on an industrial project on automatic inspection of sewage pipes. He returned to Canberra in 1992 to take up a lectureship in the School of Computer Science, University College, the University of New South Wales (UNSW), the Australian Defence Force Academy (ADFA), where he was later promoted to Senior Lecturer and Associate Professor. He moved to the University of Birmingham, England, as a Professor of computer science in 1999. Currently, he is the Director of the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA). His research interests include evolutionary artificial neural networks, automatic modularization of machine learning systems, evolutionary optimization, constraint handling techniques, computational time complexity of evolutionary algorithms, iterated prisoner's dilemma, and data mining.

Dr. Yao won the 2001 IEEE Donald G. Fink Prize Paper Award for his work on evolutionary artificial neural networks. He is the editor-in-chief of IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the Associate Editor of several other journals. He chairs the IEEE NNS Technical Committee on Evolutionary Computation and has chaired/co-chaired more than 25 international conferences and workshops. He has given more than 20 invited keynote/plenary speeches at conferences and workshops world-wide. His Ph.D. work on simulated annealing and evolutionary algorithms was awarded the President's Award for Outstanding Thesis by the Chinese Academy of Sciences.

Chi-Hon Choi received the B.Eng degree in systems engineering and engineering management from the Chinese University of Hong Kong (CUHK), where she is currently pursuing the M.Phil degree, also in systems engineering and engineering management.

Her current research interests include design and analysis of data warehousing and online analytical processing, design and implementation of database management systems, query processing and query optimization.

Gang Gou received the B.S. degree from the Department of Computer Science and Technology, NanKai University, China, in 2000.

He is currently pursuing the M.Phil. degree in the Department of Systems Engineering and Engineering Management at the Chinese University of Hong Kong. His recent research focuses on data warehouse, OLAP queries, and materialized view selection. He has also interests in approximate query processing, data streams processing, and data mining.