Original paper

Soft Computing 7 (2003) 574–581 © Springer-Verlag 2003
DOI 10.1007/s00500-002-0243-1

Applying evolutionary algorithms to materialized view selection in a data warehouse

J.-T. Horng, Y.-J. Chang, B.-J. Liu

Abstract Effective analysis of genome sequences and associated functional data requires access to many different kinds of biological information. A data warehouse [14,16] plays an important role for storage and analysis for genome sequence and functional data. A data warehouse stores lots of materialized views to provide an efficient decision-support or OLAP queries. The view-selection problem addresses to select a fittest set of materialized views from a variety of MVPPs 0 forms a challenge in data warehouse research. In this paper, we present genetic algorithm to choose materialized views. We also use experiments to demonstrate the power of our approach.

Keywords Data warehouses, Genetic algorithms, Genomes, Materialized views

1 Introduction

Effective analysis of genome sequences and associated functional data requires access to many different kinds of biological information. A data warehouse plays an important role for storage and analysis for genome sequence and functional data. For example, when analyzing gene expression data, it may be useful to have access to the sequences upstream of the genes, or to the cellular location of their protein products. Such information is currently stored in different formats at different sites in a way that does not readily allow integrated analyses.

A data warehouse system is a repository of integrated information, which can be provided for query or analysis. Data warehouse systems can collect and maintain information form multiple distributed, autonomous, or heterogeneous information sources, related data retrieved from the information source can be processed and transformed into internal types available for data warehouse systems. The related data may be integrated with other information already existing in a data warehouse system, for part of results responded to user has been calculated and restored in a data warehouse system.

Published online: 14 July 2003

J.-T. Horng (✉), Y.-J. Chang, B.-J. Liu
Department of Computer Science and Information Engineering, National Central University, Taiwan
e-mail: horng@db.csie.ncu.edu.tw

We would like to thank the authors, i.e. J. Yang, K. Karlapalem, and Q. Li, of the paper [15]. In this study, we borrow their mathematical model of the work in [15].

The main functions of data warehouse systems are retrieving, filtering, integrating related information required by complex queries. In contradiction to on-demand approach (extract data only when processing queries) of traditional databases, data warehouse system provide a in-advance approach (interested data are retrieved from information source in advance). It is not necessary to re-calculate the whole query result, because some parts of the result has been calculated and restored in repository (data warehouse), and these parts can be used directly by data warehouse system. Therefore, required time can be reduced by handing complex queries to data warehouse system. Because processed information has been stored in data warehouse, there exists inconsistency between data warehouse and underlying information sources.

Figure 1 illustrates a basic architecture of a data warehouse system. The information sources are usually database systems, but may be non-traditional data such as general files, HTML and SGML documents, news wires, knowledge base, and legacy systems, etc. Each information source connects to a wrapper/monitor. The major tasks of the wrapper/monitor are the translation and change detection. The wrapper is responsible for translating the schema of the information source it concerns to the schema which is used by the data warehouse system. The monitor module is in charge of detecting any change from the information source it connects to, and reporting those changes to the component above, the integrator. Any change from information sources will be propagated to the integrator. The integrator is responsible for bringing source data into the data warehouse, propagating changes in the source relations to the data warehouse, and maintaining the data extracted in the data warehouse, which may include merging, filtering and summarizing information from different information sources. When storing integrated data, it may need to obtain further information from the same or different information sources. Then it would send requests to the appropriate wrapper modules below it.

The Genome Information Management System (GIMS) [17] is an object database that integrates genome sequence data with functional data on the transcriptome and on protein-protein interactions in a single data warehouse. The K1 system is a bioinformatics integration system [18].

As relevant information becomes available or is modified, the information is extracted from its source, translated into a common model (e.g., the relational model), and integrated with existing data at the warehouse. The information stored at the warehouse is in the form of views, referred to as materialized views, derived from the

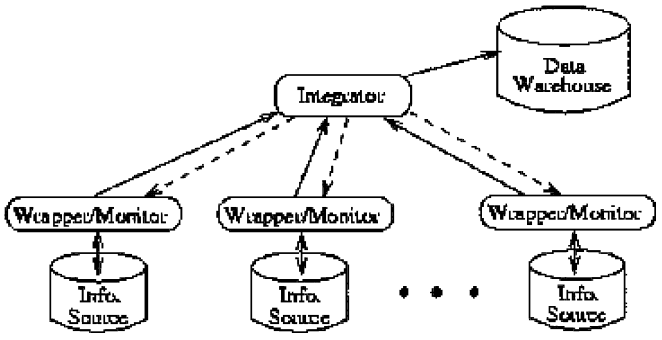

Fig. 1. Basic Architecture of a Data Warehouse

sources 000. In order to keep a materialized view consistent with the data at sources, the view has to be incrementally maintained. The maintenance of views incurs what is known as view maintenance or update costs. It is impossible to materialize all the views but rather to materialize partial shared portions of the base data. In this paper, we focus on the problem of selecting an appropriate set of materialized views to answer a query. The selection of an appropriate set of materialized views is one of the most important design decisions in designing a data warehouse system. The problem can be described as follows. Select a set of materialized views to minimize total query response time and view maintenance cost of the selected views. This problem was denoted as view-selection problem (VSP) in 0.

To solve the VSP, previous studies 000 proposed several heuristics. In 0, a greedy algorithm is proposed to solve the VSP such that the set of materialized views satisfies the space constraint. Although the approach guarantees that it can obtain solution achieved (1–1/e) times the optimal benefit, its solution quality is still not good enough. The work in 0 focuses on dynamic view updating problem.

In 0, the authors present a framework and algorithm for analyzing the issues in selecting views to be materialized so as to achieve the best combination of good query performance and low view maintenance. The algorithm they proposed to solve VSP is based on the idea of reusing temporary results from the execution of global queries with the help of Multiple View Processing Plan (MVPP). The VSP is described in 0 as follows:

- Generate the best MVPP by merging the common subexpresssions.
- Select views to be materialized from the best MVPP.

When VSP is decomposed into these two steps described above, some potential solutions maybe not to be selected. Because the best materialized views set may exists in a non-best MVPP. We will discuss the problem in next section.

It has been proven that the VSP is a NP-complete problem 0. To solve it, an obvious approach is to apply the exhaustive algorithm for materialized view selection on the set of queries. However this approach is very expensive if the search space is large. So many researchers have applied the heuristics to trim the search space, in order to get the results quickly. In order to avoid exhaustively searching the whole solution space and to obtain a better solution than that obtained by heuristic, we apply evolutionary approach to deal with this problem. We then demonstrate the power of our approach through experimental results.

This paper is organized as follows. The mathematical model of VSP is formalized in Sect. 2. Section 3, we apply genetic algorithms to solve the model. In Sect. 4, we show several experimental results. Conclusions and future works are summarized in Sect. 5.

2
Introduction of view-selection problems

In this chapter we will first describe the Mathematical Model of View Selection Problems cited from MVPP 0. We will introduce the specification in defining MVPP and introduce the formal Mathematical Model of View Selection Problems then. In Sect. 2.2, we will give a motivating example to explain this model.

2.1
Mathematical model of view selection problems

MVPP 0 (Multiple View Processing Plan) represents a possible global query access plan for the queries supported in a data warehouse system, in which the local access plan for individual queries are merged based on the shared operations on common data sets. An MVPP is labeled dag $M = (V, A, C_a^q, C_m^r, f_q, f_u)$ where V is a set of vertices, A is a set of arcs over V, such that

- for every relational algebra operation in a query tree, for every base relation, and for each distinct query, we create a vertex;
- for $v \in V$, $T(v)$ is the relation generated by corresponding vertex v. $T(v)$ can be a base relation, an intermediate result while processing a query, or the final result for a query;
- for any leaf vertex v, (that is, it has no edges coming into the vertex), $T(v)$ corresponds to a base relation, and is depicted using the symbol. Let L be a set of leaf nodes. For any vertex $v \in V$, $f_u(v)$ represents the update frequency of v;
- for any root vertex v, (that is, it has no edges going out of the vertex), $T(v)$ correspond to a global query, and it depicted using the • symbol. Let R be a set of root nodes. For every vertex $v \in V$, $f_q(v)$ represents the query access frequency of v;
- if the base relation or intermediate result relation $T(u)$ corresponding to vertex u is needed for further processing at a node v, we introduce an arc $u \rightarrow v$;
- for every vertex v, let $S(v)$ denote the source nodes which have edges pointed to v; for any $v \in L$, $S(v) = \emptyset$. Let $S^*\{v\} = S(v) \cup \{\cup_{v' \in S(v)} S^*\{v'\}\}$ be the set of descendants of v;
- for every vertex v, let $D(v)$ denote the destination nodes to which v is pointed; for any $v \in R$, $D(v) = \emptyset$. Let $D^*\{v\} = D(v) \cup \{\cup_{v' \in D(v)} D^*\{v'\}\}$ be the set of ancestors of v;
- for $v \in V$, $C_a^q(v)$ is the cost of query q accessing $T(v)$; $C_m^r(v)$ is the cost of maintaining $T(v)$ based on the changes to the base relation $S^*(v) \cap R$, if $T(v)$ is materialized.

Before we describe the problem, we again introduce all the notations used in this paper:

- There are k queries supported in a data warehouse.
- For any query q, there is a set of possible query execution plans.
- There are n nodes (sub-expressions/queries) involved in all these plans.
- An MVPP is constructed by k plans (each query presents one plan).

Now the view selection problem can be described as:

How to select an appropriate set of materialized views from a certain MVPP, so that the total query processing cost for the supported queries and the total maintenance cost of these materialized views is minimal.

More formally 0, given a MVPP, let M be a set of views in a MVPP to be materialized, f_q, f_u, the frequency of executing queries and frequency of updating base relations, respectively. Furthermore for each $v \in M$, let $C_a^q(v)$ and $C_m^r(v)$ denote the cost of access for query q using v and the cost of maintenance of view v base on changes to base relation r, respectively (where $v \in R$ is the set of queries and $r \in L$ is the set of base relations).

Then the query processing cost will be

$$C_{\text{queryprocessing}}(v) = \sum_{q \in R} f_q C_a^q(v)$$

And the materialized view maintenance cost will be

$$C_{\text{maintenance}}(v) = \sum_{r \in L} f_u C_m^r(v)$$

DEPARTMENT(DNAME, DNUMBER, MGRSSN, GGRSTARTDATE) EMPLOYEE(FNAME, LNAME, SSN, BDATE, ADDRESS, SEX, SALARY, SUPERSSN, DNO) PROJECT(PNAME, PNUMBER, PLOCATION, DNUM)
Q0:

```
SELECT *
FROM   EMPLOYEE, DEPARTMEENT
WHERE SALARY >50000 AND DUNMBER=DNO
```

Q1:

```
SELECT *
FROM   EMPLOYEE, DEPARTMEENT, PROJECT
WHERE DNO = DNUMBER AND DNUM=DNUMBER
AND PNAME = 'Intec';
```

Thus the total cost of materializing a view v is

$$C_{\text{total}}(v) = \sum_{q \in R} f_q C_a^q(v) + \sum_{r \in L} f_u C_m^r(v)$$

Therefore, the total cost of materializing a set of views M is C_{total}:

$$C_{\text{total}} = \sum_{v \in M} C_{\text{total}}(v)$$

This mathematical mode well describes the view-selection problem.

In 0, the VSP is solved by the following steps:

- Step 1: Generate the best MVPP by merging the common sup-expressions.
- Step 2: Select views to be materialized from the best MVPP.

As described above, however, some potential sets of materialized views are missed to choose when they are not in the best MVPP. In this paper, we present another approach to solve VSP. We try to visit every MVPP as much chance as possible so that we can find the best solution for VSP.

2.2 Motivating examples

Figure 2 demonstrates four simple examples of MVPP. The example is taken from a data warehouse system we simulated. The relations and the attributes of the schema for this application and two queries are given below. The schema and queries are from 0.

In Fig. 2, each graph is a candidate MVPP which is constructed by combining potential query execution plans of queries. And in each graph, the query access frequencies are labeled on the top of each query node, and the update frequencies of base relation are labeled on the bottom of each base relation node. For each node except the root (query node) and leaf (base relation node) nodes, there are two data associated with it. The left stands for the query operator and the right stands for the cost to generate the nodes from base relations. The number in the internal node except the root indicates the intermediate nodes to be selected to materialize. The same number in different MVPPs represent the same operation. In order to calculate the cost, we make the following assumptions:

- There are 10 tuples in the DEPARTMENT relation.
- There are 30 k tuples in the EMPOYEE relation and the attribute DNO is a non-null one.
- There are 20 tuples in the PROJECT relation.
- The cost of answering a query Q is the number of rows present in the table used to construct Q. The calculation of the cost to answer a query is similar to that in 0.
- The methods for implementing select and join operation are linear search and nested loop, respectively.

Based on the above assumption, the cost of each operation node in Fig. 2 is labeled at the right side of a node.

Suppose that there are some intermediate nodes to be materialized. For each query, the cost of query processing is query frequency multiplied by the cost of query access from the materialized node(s). The maintenance cost for materialized view is the cost used for constructing the view. For example, in Fig. 2a, if node 0 is chosen to be materialized, the query processing cost for Q0 and Q1 are 3*30 k and 2*30.02 k, respectively. The view maintenance cost is (8+0.5)*(300 k). The total cost for an MVPP is the sum of query processing and view maintenance costs. The goal of solving VSP is to find a set of nodes to be materialized, so that the total cost is minimal.

3 Applying genetic algorithms to view selection problems

As defined above, it is simply to find that VSP is an NP-complete problem with complexity $O(k^m 2^n)$, where k is the number of queries supported in a data warehouse, and each query has m query execution plans, and n is the number of view node in a MVPP. Genetic algorithm (GA) 0 0 0 is a well-known approach to solving NP-complete problems. Genetic Algorithms are general purpose opti-

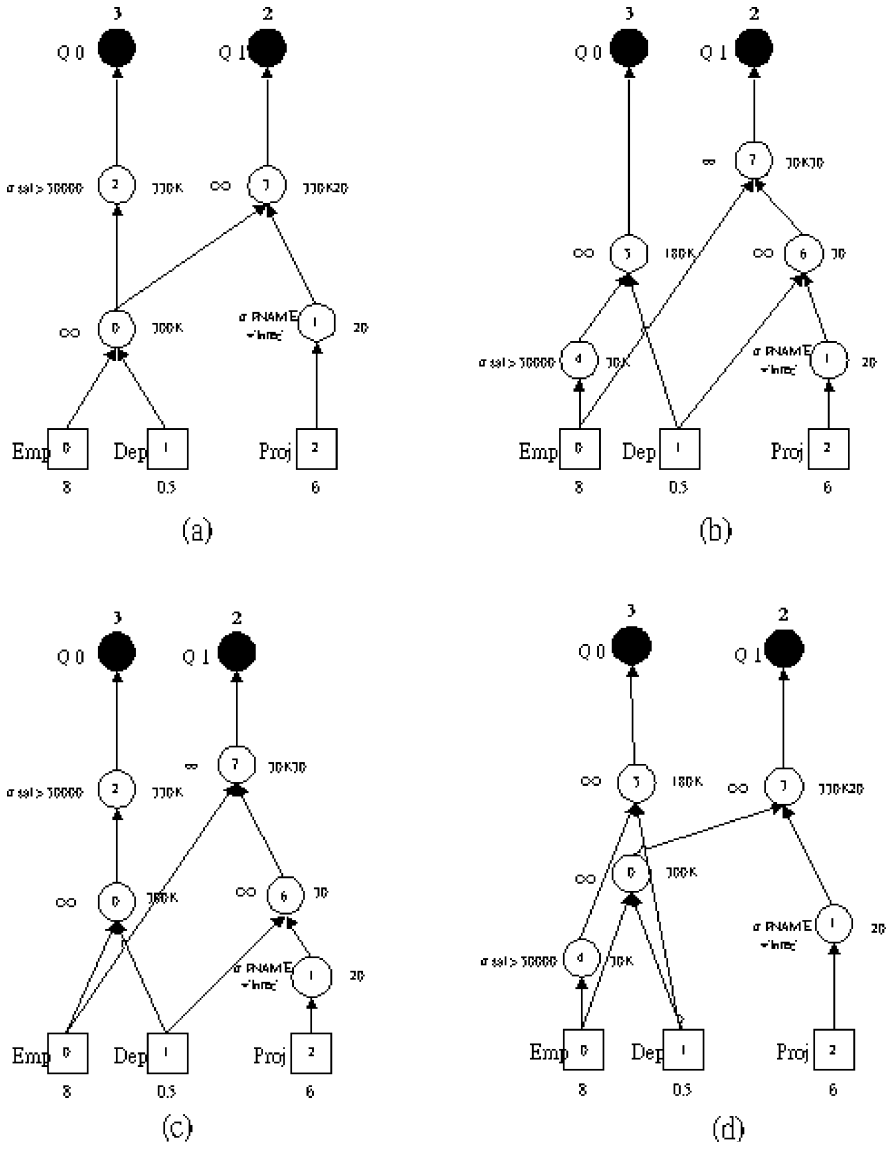

Fig. 2. Example MVPPs

mization and search algorithms based on the mechanics of natural selection and natural genetics. The genetic search proceeds over a number of generations, with each generation represented by a population of current chromosomes (bit strings). In every generation, a new set of chromosomes is created (here we call this kind of operation *crossover*) using bits and pieces of the fittest of the old. "Survival of the fittest" provides the pressure of populations to develop increasingly fit individuals. In addition, an occasional new part is tried (here we call this kind of operation *mutation*) for good measure.

3.1 Chromosome representation

In order to apply genetic algorithms, we should first encode the solution of view-selection problem into chromosome representation. But we have met problems in this step.

- How to encode both the plan number of each query and the views of each MVPP in a chromosome?
- Each chromosome maybe has different length because different set of query plans will construct different MVPP and maybe have different number of nodes.

If we want to apply genetic algorithms, we should get over the two problems in chromosome representation. Considering the two problems, we first draw query execution plans for every query which is supported in data warehouse and then enumerate all the views appearing in each plan. Each chromosome is represented by two parts. The former part, we call *query-plan string*, records which query plan is chosen to answer a query. In database query processing. A query may have many execution plans. Each execution plan will use many intermediate operations. Query optimizer finally chooses an execution plan to answer a query according to a cost model. Therefore, each query has a number to represent which query plan is chosen. The latter part, we call *view string*, records which view will be materialized in data warehouse. This part consists of binary bit string. That is, the bit 1 indicates the corresponding candidate view will be materialized. On the other hand, the corresponding

candidate view will not be materialized. The length of this part is the number of the candidate views appearing in all the MVPPs.

For instance, given 4 queries and assume each query has 3 execution plans and the total number of views appeared in these 12 plans is 20. The chromosome is represented as [0,2,1,2][1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]. The first array in the representation indicates a MVPP is constructed by the first execution plan for query 1, the 3rd plan for query 2, the second plan for query 3 and the 3rd plan for query 4. The latter array indicates the first candidate view is chosen to materialize.

Note that not each chromosome may or may not be a feasible solution because not all views appear in a MVPP. For example, if candidate view 1 does not in a MVPP which is constructed by the first, the third, the second, and the third plans for query 1, 2, 3, and 4, respectively. The chromosome [0,2,1,2] [1,0,0,0,0,0,0,0,0,0,0,0,...] is then not a feasible solution. A certain MVPP may only involve a certain of candidate views. Let us go back to the MVPPs in Figure 2. There are eight candidate views in all the MVPPs, in total. However, views 1, 2, 3, and 4 are candidate views only if we use the MVPP in Fig. 2a. In this case, the positions in 0, 1, 2, 3 (The first views is placed in the position zero in our representation.) may appear the value 1 or 0. Other positions should be zero. We only need to concern the positions once a MVPP is decided. Therefore the search space can decrease to a lower bound in our approach.

For each solution (chromosome) we have a set of materialized views which are the corresponding 1s nodes in the decided MVPP. And for each solution the cost function returns a total cost of query cost and maintenance cost. The smaller value the cost function gets, the better the solution is.

3.2
Genetic local search algorithm

Genetic local search (GLS) algorithm 000 is a hybrid heuristic that combines the advantages of population-based search such as Genetic Algorithm and local optimization. Local search iteratively moves from one solution to a better one in its neighborhood until a local minimum is reached 00. While it quickly finds good solutions in small regions of the search space, the genetic operators are suitable for exploring the whole search space in order to identify interesting regions.

In Fig. 3, a template of a genetic local search algorithm for VSP is given. As described above, not all solutions are feasible solutions. In our algorithm, we design a check-feasible function to filter illegal genes. Except to filter illegal genes the function can reduce search space. When the initial population is generated, *Check-feasible*() first enumerates the involved views according to the query-plan string and records them. The function Local-Search is used to tune the values of view string according to the value of query-plan string. Local search is performed for each individual to obtain a population of local minimum. In the main loop, crossover and mutation operators are applied to randomly choose individuals for a predefined number of times.

```
procedure GLS;
    begin
        initialize population P,
        for each individual i ∈ P do
            i := Check-feasible(i);
            i := Local-Search(i);
        repeat
            for i := 1 to #crossovers do
                select two parents i_a, i_b ∈ P randomly;
                i_c := Crossover(i_a, i_b);
                i_c := Check-feasible(i);
                i_c := Local-Search(i_c);
                add individual i_c to P;
            endfor;
            for i := 1 to #mutations do
                select an individual i ∈ P randomly;
                i_m := Mutate(i);
                i_m := Check_feasible(i);
                i_m := Local-Search(i_m);
                add individual i_m to P;
            endfor;
            P := select(P);
        Until converged;
    End;
```

Fig. 3. Genetic Local Search Algorithm

3.3
Genetic operators

In this subsection, we introduce our genetic operators including selection, crossover, mutation and selection.

Selection

The purpose of selection in genetic algorithms is to give more reproductive chances, on the whole, to those population members that are the most fit. There are many well-known kind of selection such as random selection, roulette wheel selection, ranking selection... etc. The key point of selection is to keep the best parent have the most chance to be selected out. We adopted ranking selection in our approach. We first order the parent by the cost, then we select individuals by choosing a uniform random number between 1 and $\sqrt{n}$, where n is the population size, then squaring it.

Crossover

In genetic algorithms, crossover recombines genes in two parent chromosomes to generate offspring. The crossover operator enables us to define a new starting point for a local search based on the information contained in the current population. The aim of the crossover is to determine the regions of the search space where better solutions are most likely to be found by the local search procedure.

In our approach, we apply one-point crossover to chromosome selected. A chromosome is composed of two parts. Each part is crossovered as follows.

- Query-plan string: Two query-plan strings from two chromosomes are selected to crossover and generate two offspring. For example, given 4 queries and each number of query execution plan are 2, 2, 4, 4, respectively. Applying one-point crossover to the two query-plan strings [1100] and [3311] and generate two offspring [1111] and [3300] if the cut point is in the position 2. The function Check-feasible is then applied to check whether the offspring is feasible. We use

"mod" function to adjust if it is an infeasible solution. In this example, the query-plan string [3300] is an infeasible one. We adjust it to [1100] using the "mod" function.

- View string: Simply exchange two view strings from two parent and then use check-feasible function to make the newborn chromosome legal.

Mutation

In simple genetic algorithms, mutation is the occasional (with small probability) random alteration of the value of a bit position. Mutation is needed because, even though reproduction and crossover effectively search and recombine extant notions, occasionally they may become overzealous and lose some potentially useful genetic material (1's or 0's at particular locations). In this paper, mutation in view string simply changes a 1 to a 0 and vice versa in a chromosome. Mutation in a query-plan string increases 1 to a selected query and make sure it is feasible according to its plan number.

4 Experiments and results

We experimented with the method we propose to examine the quality and stability of the solutions.

4.1 Testbed

Our experiments are designed based on the schema and instances given below. The database is borrowed from 0.

Base relations

DEPARTMENT(DNAME,DNUMBER, MGRSSN, GGR-STARTDATE) fu=0.5
EMPLOYEE(FNAME, LNAME, SSN, BDATE, SEX, SALARY SUPERSSN, DNO) fu=8
PROJECT(PNAME, PNUMBER, PLOCATION, DNUM) fu=6
WORKS_ON(ESSN, PNO, HOURS) fu=13
DEPENDENT(ESSN, DE- PEN- DENT_ NAME, SEX, BDATE, RELATIONSHIP) fu=9

The "fu" in each relation indicates the update frequency of the relation. We also use the following ten queries to test the performance of our approach. The "Fq" associated with each query is the frequency used.

```
Q0: Fq=3
SELECT      *
FROM EMPLOYEE, DEPARTMEENT
WHERE       SALARY >50000 AND DUNMBER=DNO

Q1: Fq=2
SELECT      *
FROM EMPLOYEE, DEPARTMEENT, PROJECT
WHERE       DNO = DNUMBER AND DNUM=DNUMBER
AND PNAME = 'Intec';

Q2: Fq=5
SELECT      FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMEENT
WHERE       DNAME='Research' AND DUNMBER=DNO

Q3: Fq=10
SELECT      PNUMBER, DNUM, LNAME, ADDRESS,
BDATE
FROM PROJECT, EMPLOYEE, DEPARTMEENT
WHERE       DNUM=DNUMBER AND MGRSSN=SSN
AND PLOCATION='Stafford'

Q4: Fq=2
SELECT      PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY PNUMBER, PNAME;

Q5: Fq=1
SELECT      COUNT(*)
FROM EMPLOYEE, DEPARTMENT
WHERE       DNAME='Research' AND DUNMBER=DNO

Q6: Fq=7
SELECT      PNUMBER, PNAME
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE       PNUMBER=PNO AND SSN=ESSN AND
DNO=5

Q7: Fq=3
SELECT      SSN, COUNT(*), DNAME
FROM EMPLOYEE, DEPENDENT
WHERE       SALARY >50000 AND SSN=ESSN
GROUP BY ESSN
HAVING COUNT(*) >3

Q8: Fq=1.5
SELECT      PNAME, HOURS, SSN
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE       PNUMBER=PNO AND SSN=ESSN AND
BDATE≤1950;

Q9: Fq=2.5
SELECT      PNAME, FNAME, LNAME
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE       PNUMBER=PNO AND SSN=ESSN AND
HOURS>20
AND PLOCATION='Stafford'
```

Before starting the experiments, we show the possible query execution plans for each query. The cost of each

Table 1. Tuples in each base relation

Relation(s)	DEPARTMENT	EMPLOYEE	PROJECT	WORKS_ON	DEPENDENT
Tuples	10	30 k	20	60 k	96 k

node is calculated in every plan. Figure 4 presents two possible query execution plans for query Q0.

4.2 Experimental results

In our experiments, the problem size is represented as *x*Q*xx*n. The number before the character *Q* indicates the number of queries and that after *Q* is the number of views involved in such queries. For example, the problem size 7Q46n means there are seven queries and forty-six views involved in these seven queries.

Figure 5 compares our GLS with the approach in 0, namely YKL97. The YKL97 0 approach finds the best MVPP and then select a set of views to materialize based on the MVPP. In our experiment, we will discuss from two aspects of data to compare our GLS to YKL97, one is executing time and the other is executing quality. From the experimental results, we find our approach performs better than YKL97 especially in the large problem.

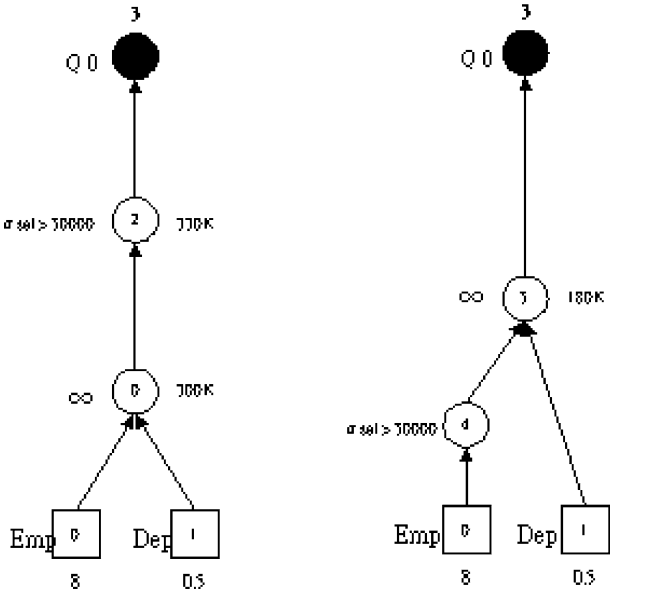

Fig. 4. Potential Query Execution Plans for Query 0

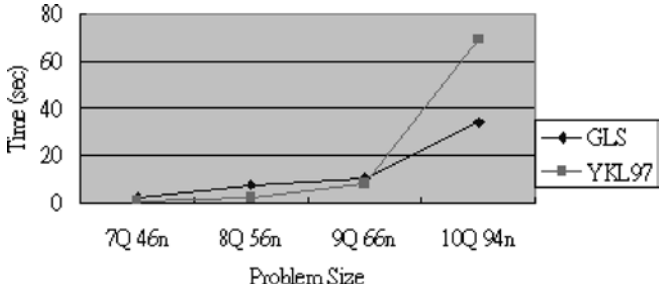

Fig. 5. Execution Time

Table 2. The comparison of execution time of different problem size

	GLS	YKL97	YKL_force
7Q 46n	2.56	1	13
8Q 56n	7.7	2	166
9Q 66n	10.2	8	3306
10Q 94n	33.9	69	NA

Table 2 shows the comparison of our approach with other approaches in the execution time of the different problem sizes. The YKL_force approach on Table 2 means it first applies YKL970 to find the best MVPP and then applies brute-force to select a set of views such that the total cost including query cost and maintenance cost is minimum. The symbol "NA" in Table 2 indicates it is impossible to test all the solutions because the search space is too large. We find our approach can save much time than YKL97 in large problem size from Table 2. In Figure 5, obviously, we found that GLS and YKL97 both got a quickly results when the problem sizes is small. But as the problem size grew, the curve of YKL97 is much steeper than GLS. So we can prove that GLS is a better/quicker solution when the problem size is big and GLS spend an acceptable time period even the problem size is small. In the later, we will explain why we need spend more time when problem size is small by comparing solution quality.

As to the solution quality, Table 3

Table 3 demonstrates the minimal cost of each approach found for different queries. The plans that are used to answer a query and the selection of set of views in Table 3 are also shown in Table 4. Plans list in Table 4 can be found in Appendix. As Table 3 and Table 4 show, GLS

Table 3. Execution quality of different problem size

	GLS	YKL97
4 query	958.65	2543.42
5 query	3478.69	5063.46
6 query	3483.19	5396.46
7 query	1264473.19	1266488.46
8 query	1714923.31	1721380.53
9 query	2165145.19	2167608.66
10 query	2176432.88	2392646.35

Table 4. Selected plans and views problem size

		Plan of each query	Set of MV
4 query	GLS	1 3 1 2	1,20,21
	YKL97	1 2 1 3	1,4,18,21
5 query	GLS	1 3 1 2 0	1,20,21
	YKL97	1 2 1 3 0	1,4,18,21
6 query	GLS	1 3 1 2 0 0	1,19,21
	YKL97	1 2 1 3 0 1	1,4,18,21
7 query	GLS	1 3 1 2 0 0 1	1,19,21,45
	YKL97	1 2 1 3 0 1 1	1,4,18,21
8 query	GLS	1 3 1 2 0 0 1 2	1,4,19,24,35
	YKL97	0 2 0 3 0 1 1 2	1,4,21,46
9 query	GLS	1 3 1 2 0 0 1 2 3	1,4,19,21,35
	YKL97	1 2 1 3 0 1 1 2 3	1,4,18,21,46,56
10 query	GLS	1 3 1 2 0 0 1 2 3 0	1,4,19,21,35
	YKL97	1 2 1 3 0 1 1 2 3 3	1,4,18,21,46,56

perform more accurate minimum solution than YKL97 in each problem size almost 100% advancement. (4–6 queries). So it' s reasonable that GLS will spend more time than YKL97 when the problem size is small.

5 Conclusion

In this paper, we have addressed and designed algorithms for the materialized view design problem, i.e., how to select a set of views to be materialized from diverse MVPPs so that the total cost of processing a set of queries and maintaining the materialized views is minimized. We developed an approach using genetic local search algorithm to solve the problem. We first formalize the view selection problem to an optimized problem. Then we use our GLSA to solve the model. We have shown the representation of the problem in genetic algorithm using the four basic operators: selection, crossover, mutation, local search, based on the principle of genetic local search algorithm. The results of the previous work of the study is shown in 0. We further identify the weakness of our previous study and then solve the problem using the genetic approach. We also use experiments to demonstrate the power of our approach. From the experimental results, we find our genetic approach performs well.

References

1. **Agrawal D, Abbadi AE, Singh AK, Yurek T** (1997) Efficient view maintenance at data warehouses. SIGMOD Conference, pp. 417–427
2. **Chang CC, Trappey AJC** (1996) A Framework of product data management system – procedures and data model. Master's thesis, Department of Industrial Engineering, National Tsing Hua University, Hsinchu, Taiwan
3. **Davis L, Smith D** (1987) Genetic algorithms and simulated annealing: an overview. In Davis L, (ed), Genetic algorithms and simulated annealing. pp 1–11. Pitman, London
4. **Elmasri R, Navathe SB** (1994) Fundamentals of DataBase Systems, Second Edition, The Benjamin/Cummings Publishing Company, Inc., CA, USA
5. **Goldberg DE** (1989) Genetic Algorithms in search, optimization, and machine learning. Addison-Wesley, Publishing Company, Inc., Reading MA, USA
6. **Gupta A, Mumick, IS** (1995) Maintenance of materialized views: problems, techniques, and applications. IEEE Data Eng. Bulletin, (Special Issue on Materialized Views and Data Warehousing) 18(2):3–18
7. **Gupta H** (1997) Selection of views to materialize in a data warehouse. Proceedings of the 23rd VLDB conference, Athens, Greece, pp. 156–165
8. **Horng JT, Chang YJ, Liu BJ** (1999) Materialized View Selection Using Genetic algorithms in Data Warehouse System. Late Breaking Papers at the Genetic and Evolutionary Computation Conference, pp. 107–115
9. **Ishibuchi H, Murata T, Tomioka S** (1997) Effectiveness of genetic local search algorithms. In: Proceedings of the 7th International Conference on Genetic Algorithms, East Lansing, USA, pp. 505–513
10. **Kolen A, Pesch E** (1994) Genetic local search in combinatorial optimization, discrete applied mathematics and combinatorial operations research and computer science, vol. 48, pp. 273–284
11. **Labio W, Zhuge Y, Wiener JL, Gupta H, Garcia-Molina H, Widom J** (1997) The WHIPS Prototype for Data Warehouse Creation and Maintenance. SIGMOD Conference, pp. 557–559
12. **Merz P, Freisleben B** (1997) A Genetic local search approach to the quadratic assignment problem. In: Proceedings of the 7th International Conference on Genetic Algorithms, East Lansing, USA, pp. 465–473
13. **Ross KA, Srivastava D, Sudarshan S** (1996) Materialized view maintenance and integrity constraint checking: Trading Space for Time. SIGMOD Conf., pp. 447–458
14. **Widom J** (1995) Research problem in data warehousing. In: Proc. of the 4th Intl. Conf. on Info. and Knowledge Mngt., pp. 25–30
15. **Yang J, Karlapalem K, Li Q** (1997) Algorithms for materialized view design in data warehousing environment. VLDB pp 136–145
16. **Zhuge Y, Garcia-Molina H, Hammer J, Widom J** (1995) View maintenance in a warehousing environment. In: Proceedings of the ACM SIGMOD Intl. Conf. on Mngt. of Data, pp. 316–327
17. **Cornell M, Paton NW, et al.** (2000) GIMS-A Data Warehouse for Storage and Analysis of Genome Sequence and Functional Data, Bioinformatics 16(6): 548–557
18. **Wong L** (2001) Bioinformatics Integration Simplified: The Kleisli Way. Frontiers in Human Genetics: Diseases and Technologies, pp. 79–90, World Scientific, Singapore