Multiobjective Genetic Algorithms for Materialized View Selection in OLAP Data Warehouses

Michael Lawrence
Faculty of Computer Science
Dalhousie University
Halifax, NS, Canada
B3H 1W5
michaell@cs.dal.ca

ABSTRACT

On-Line Analytical Processing (OLAP) tools are frequently used in business, science and health to extract useful knowledge from massive databases. An important and hard optimization problem in OLAP data warehouses is the view selection problem, consisting of selecting a set of aggregate views of the data for speeding up future query processing. A common variant of the view selection problem addressed in the literature minimizes the sum of maintenance cost and query time on the view set. Converting what is inherently an optimization problem with multiple conflicting objectives into one with a single objective ignores the need and value of a variety of solutions offering various levels of trade-off between the objectives. We apply two non-elitist multiobjective evolutionary algorithms (MOEAs) to view selection under a size constraint. Our emphasis is to determine the suitability of the combination of MOEAs with constraint handling to the view selection problem, compared to a widely used greedy algorithm. We observe that the evolutionary process mimics that of the greedy in terms of the convergence process in the population. The MOEAs are competitive with the greedy on a variety of problem instances, often finding solutions dominating it in a reasonable amount of time.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Data warehouse and repository*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms: Performance

Keywords

multiobjective optimization, genetic algorithms, data warehousing, OLAP, view selection, multiobjective evolutionary optimization

1. INTRODUCTION

Decision makers are increasingly using data warehousing and On-Line Analytical Processing (OLAP) tools to extract useful knowledge from data created in the operation of enterprises. OLAP queries usually involve aggregation, which is achieved by the `GROUP-BY` clause in the Structured Query Language (SQL). Since the results of some OLAP queries are so expensive to compute, aggregate views of the data are often pre-computed and stored in order to speed up future query processing. Ideally all views would be pre-computed and made available for answering aggregate queries, but realistically there are constraints on which views can be materialized ahead of time such as the size of the materialized view set or the time it takes for updates to be applied to it.

The problem of choosing a set of views for materialization is known as the *View Selection Problem*. In the view selection problem one wishes to select a set of views for materialization which minimizes one or more objectives, possibly subject to one or more constraints. Many variants of the view selection problem have been studied for example: i) minimizing the query cost of a materialized view set subject to a storage size constraint [12, 22, 15, 21, 3], ii) minimizing query cost and maintenance cost of a materialized view set [2, 9, 27, 14, 26, 24, 25, 30, 28, 11], and iii) minimizing query cost under a maintenance cost constraint [20, 19, 15, 29, 10].

Since the result of OLAP queries can be critical in supporting decisions, their information must be as up to date as possible. For this reason the maintenance cost of the selected view set is an important property, and minimizing maintenance cost allows the view set to be updated with minimal disruption to the system. This objective, as well as a possible storage space limitation on the set of materialized views, is in conflict with the desire to respond to queries as quickly as possible. All of the studies cited above which minimize query and maintenance cost in fact minimize a linear combination of the two objectives, attempting to provide a solution with the smallest total overall cost. Whether this solution is the best overall solution, or whether a best overall solution even exists depends on the meaning of the quantities expressed by the objectives and their relative scale, and requires the trade-off between the objectives to be known a priori in order to choose appropriate constants in front of each objective. Failure to find good constants may cause an algorithm minimizing overall cost to in fact minimize the more expensive of the two objectives without regard to the other, defeating the very purpose of considering both objectives in the first place. Also at hand is the issue of comparability of solutions. For example algorithms A and B might find solutions X and Y which have vastly different query and maintenance costs but very similar overall costs. The measure of overall cost fails to capture the difference of these solutions.

We take an approach to view selection which, surprisingly, we cannot find record of being attempted before. We apply Multiobjective Evolutionary Algorithms (MOEAs) to minimize both query

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

and maintenance costs under a size constraint, providing a variety of solutions which offer various levels of trade-off between the objectives. One solution is said to *dominate* another when it is favourable in both objectives. The output from a multiobjective genetic algorithm is a set of solutions, none of which dominates any of the others. This way an administrator can select an appropriate view set from the possible solutions without needing to make assumptions about the trade-off between maintenance and query cost, avoiding the problems of linearly combining the objectives as outlined above.

We apply two well established non-elitist MOEAs to view selection, Horn et. al.'s Niched Pareto Genetic Algorithm (NPGA) [13], and Fonseca and Fleming's Multiobjective Genetic Algorithm (MOGA) [7]. Since our problem is constrained, we also experiment with two methods for handling constraints. One method treats the constraint like an objective, and redefines dominance for infeasible solutions. The other allows infeasible solutions in the population, but implements a stochastic repair mechanism which converts them into feasible solutions for fitness calculation. For comparison we also apply the popular greedy Benefit Per Unit Space (BPUS) algorithm for view selection whose performance is well known and is documented in [12, 11].

Our results indicate that MOEAs are competitive against the greedy approach across a variety of problem instances ranging in size and skew. The MOEAs often find solutions which dominate the greedy algorithm, although the gap closes with larger problem sizes. Both MOEAs perform nearly the same for all problem instances except the highly skewed, where MOGA outperforms NPGA. We also find MOGA generally maintains better diversity than NPGA, and that the constraint handling method of repairing infeasible solutions tends to cause the population to drift up toward the middle of the pareto-front.

The rest of this paper is organized as follows. Section 2 gives a more detailed formulation of the view selection problem, as well as a brief review of previous approaches. Section 3 describes constrained multiobjective optimization, the two algorithms which are implemented, and the methods for handling constraints. Section 4 presents our experimental results comparing the MOEAs and the BPUS heuristic on some real and synthetic data sets, while our conclusions are given in Section 5.

2. THE VIEW SELECTION PROBLEM

A typical data warehouse stores its information according to a star schema having a central fact table with d feature attributes (dimensions), and some number of measure attributes. The dimensions of the fact table have a foreign-key/primary-key relationship with the dimension tables which store the details about each value for a dimension. Queries to the data warehouse request aggregated measures from the perspective of some subset of the dimensions of the joined fact and dimension tables. The aggregated table from which a query's results are collected is called a *view*, and is identified by the dimensions chosen from the fact table, as well as the corresponding attributes in the dimension tables which determine the level of hierarchy. For example the *time* dimension may have attributes day, week, month and year which define a hierarchy. If the number of hierarchal levels of dimension i is H_i, then the total number of possible views is $\prod_{i=1}^{d}(2^{H_i}+1)$.

Harinarayan et. al. introduced the *data cube lattice* in [12], which expresses the relationship between views as a partial order (directed acyclic graph). Each view is a node, and corresponds to an answer to a particular aggregate query. There is a path from a view v_1 to a view v_2 in the lattice if queries on v_2 can be answered also using v_1. For example a query grouping on the dimensions *product* and *customer* can be answered by aggregating the results of a query grouping on *product*, *customer*, and *time* (provided that the selected dimensions in this view are at the same or higher levels of their respective hierarchies), although it will likely be more expensive than if the query were answered directly on the *time*, *customer* view. Assuming equally fast indices on all views, for each q there is a corresponding view which can be used to answer q the cheapest.

The view selection problem using a directed acyclic graph can be formally defined as follows. For each view v, we have some estimate of the number of records r_v in v, and the frequency of queries f_v on v. As in most previous studies, we adopt the linear cost model presented in [12], where the cost of answering a query on a view v is r_v. The cost $q(v, M)$ of answering aggregate queries on view v using a materialized view set M is equal to the number of records in the smallest view in M which is an ancestor of v in the data cube lattice. The overall query time using M is a weighted sum of these terms

$$Q(M) = \sum_v f_v q(v, M),$$

and the size $S(M)$ of M is simply the sum of the sizes of each of the views in M. Our constraint is that $S(M) \leq S_{max}$ for some maximum size S_{max}. The maintenance cost $m(v, M)$ of a materialized view v in M is modeled based on a cost which is assigned to every edge (v_1, v_2) in the lattice, representing the cost of maintaining v_2 using updates from v_1. The maintenance cost $m(v, M)$ of a materialized view v is the sum of the costs on the cheapest path from a materialized ancestor of v. Each node v also has an update frequency g_v, and the total update cost for a set of materialized views M is

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

This modeling of maintenance cost is the same as in [10, 20, 15, 29], however, some studies [2, 14, 30, 28] assume that $m(v, M)$ is part of the input to the problem. We experiment with maintenance cost calculated according to a more specific, but simplified model in [15]. For $v \in M$, $m(v, M)$ is equal to the number of changes in v's smallest materialized ancestor, which we express as a fixed percentage of its size. For example Figure 1 shows a data cube lattice for a database with non-hierarchal dimensions *product*, *customer*, *time*, with 8 views (one for each subset of the dimensions) and their associated sizes. If the selected view set is $M = \{pct, ct\}$ (drawn with dashed lines), the probability of queries on all views is an equally likely $1/8$, and 10% of updates to each view propagate to its descendents, then $Q(M) = 4(1/8)(1000) + 4(1/8)(600) = 800$ (since queries on the views pct, pc, pt and p are answered on the pct view at cost 1000, and queries on ct, c, t and $none$ are answered on ct at cost 600), and $U(M) = 100 + 100 = 200$ (100 updates applied to pct which also propagate to ct). If the c or t views were materialized they could be updated at an additonal cost of 10 each, since they only need the changes from ct, however if p were materialized its update cost would be 100 since it must be updated from the changes to pct.

2.1 Previous Approaches

Numerous solutions have been proposed to the view selection problem on data cubes. The first is a greedy algorithm called Benefit Per Unit Space (BPUS) presented by Harinarayan et. al. in [12]. At each greedy step, it chooses the view to add to the current solution based on maximizing the improvement in objective function it provides per unit of space it consumes. For example, when the objective is to minimize the sum $Q(M)+U(M)$, the view v added

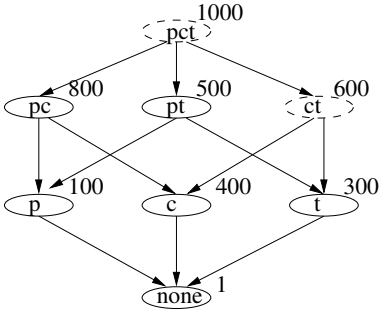


Figure 1: Example data cube lattice for a data warehouse with the non-hierarchal dimensions *product, customer,* and *time*. Associated with each view is its size in rows.

to the current solution M at each step is the one for which

$$\frac{(Q(M) + U(M)) - (Q(M \cup \{v\}) + U(M \cup \{v\}))}{r_v}$$

is maximal. BPUS has been proven to find a solution within 63% of the optimal. In [11] the same heuristic was extended to minimize the sum of query and maintenance cost in AND-OR/DAGs (a more flexible model for data cubes), as well as an algorithm given to include selection of indexes. In [2], the sum of query and maintenance cost are minimized in a data cube lattice with respect to a fixed set of queries. Shukla et. al. give a heuristic minimizing query cost in [22] which is asymptotically faster than that of [12], but achieves the same solution only under certain conditions. In [27] a greedy algorithm for minimizing the sum of query and update cost is given under a different modeling than that of [12]. Gupta gives the first solution to the view selection problem minimizing query cost under a maintenance cost constraint in [10]. In [1] Agrawal et. al. present a tool and algorithms for selecting a set of views based on a cost metric involving query cost, maintenance cost, index construction and other factors. In [20], Liang et. al. follow up on Gupta's work in [10] by giving two algorithms minimizing query cost under a maintenance cost constraint. Nadeau and Teorey give a greedy algorithm minimizing query cost under a space constraint which is polynomial in the number of dimensions [21]. Kalnis et. al. use randomized algorithms to search the solution space of view sets in [15], achieving reasonable solutions more quickly than some systematic alternatives.

Genetic algorithms were applied to view selection in [30, 19, 29]. In [30] a simple GA is used to perform view selection in Multiple View Processing Plans (MVPPs)[1] minimizing the sum of query and maintenance cost with no constraints. Their algorithm is compared against a systematic approach, and they find that the solution found by the GA is better, although the algorithm takes about 100 times longer to run. In [19] genetic algorithms are applied to minimize query cost under a maintenance cost constraint in randomly generated OR-DAGs (data cubes), with the constraint being enforced via a penalty function applied to the fitness of infeasible solutions. They compare their GAs to an exhaustive method and find their solution to be much faster and reliably fall within 10% of the optimal solution. However their experiments involve only a very small number (5 to 20) of views, where as in practice data warehouses may have a very large number (hundreds or thousands) of possible views. In [29] query cost is minimized under a maintenance cost constraint in data cube lattices. They handle constraints via a stochastic ranking procedure, where during population sorting, feasible solutions are compared by objective value, and infeasible (one or both) solutions are compared either according to objective value or a penalty function. They compare their GA to the approach in [19], the optimal A*-Heuristic presented in [10] and an exhaustive algorithm. They find that their algorithm finds solutions very close to the optimal in a fraction of the time. Their algorithm finds solutions much better than that of [19], although it takes significantly longer to run. Their experiments are also on small lattices with no more than 256 views (8 non-hierarchal dimensions).

[1]An MVPP is an alternate way to data cubes of specifying the relationship between a fixed set of queries and the tables in a relational database.

3. CONSTRAINED MULTIOBJECTIVE GENETIC ALGORITHMS

A multiobjective optimization algorithm should learn to solve different sub-problems of the overall problem. The solution therefore takes the form of a set of individuals known as the pareto-optimal front, where for each member there is no other individual which is preferable in all objectives. A major challenge in MOEAs is maintaining a diverse set of individuals which spans the pareto-optimal front, while exploiting those individuals to push the front in the direction of better objective values. The MOEAs we apply to view selection are similar in that an individual's chance of survival and of being selected for recombination depend on the number of individuals dominating it in the current population and the number of individuals in the population which are close in objective space, hence encouraging both diversity and forward progress to the pareto-optimal front.

Two well established MOEAs are considered: Horn et. al.'s Niched Pareto Genetic Algorithm (NPGA) [13], and Fonseca and Fleming's Multiobjective Genetic Algorithm (MOGA) [7], both instances of a non-elitist MOEA. MOGA assigns fitnesses to individuals on the basis of how many individuals dominate them and how many individuals have similar fitnesses. NPGA does not assign fitness, instead, replacement and selection of parents for recombination is based on a stochastic binary tournament whose outcome is based on the dominance relation and phenotypic similarity. Many developments in MOEAs taken place since the proposal of these algorithms, however our interest lies in addressing the combination of these tried and tested MOEAs with constraints on the application domain of the view selection problem. To our knowledge, MOEAs have not been applied to view selection before, and our application of them also aims to gain more insight to the nature of the problem itself.

3.1 MOGA

Fitness calculation in MOGA involves finding the number of individuals in the population P which dominate each individual a. Adding 1 to this value gives the *rank* of a. The ranks are then interpolated from best to worst, so that a gets a fitness $f'(a)$ which is proportional to its rank and the sum of the ranks of all individuals in P (with higher fitnesses being better). The fitness of each individual is then shared so that

$$f(a) = \frac{f'(a)}{\sum_{b \in P} sh(d(a,b))} \tag{1}$$

where $d(a,b)$ is the euclidean distance in objective space between a and b, and sh is a sharing function. For efficiency we use a linear interpolation, and the triangular sharing function [6]

$$sh(d) = \begin{cases} 1 - (d/\sigma_{niche})^2 & \text{if } d \leq \sigma_{niche} \\ 0 & otherwise \end{cases}.$$

σ_{niche} defines the niche radius, a parameter which is set on the basis of an estimate of the minimal desired phenotypical distance between individuals. Once the fitnesses have been calculated, any desired selection mechanism may be used (e.g. tournament, roulette wheel). We use roulette wheel selection, and replacement is done by keeping the $|P|$ fittest individuals. The entire MOGA algorithm is described in Algorithm 1.

Algorithm 1 MOGA replacement as applied here to view selection.

Input: Generation i population P_i
Output: Generation $i+1$ population P_{i+1}
1: **for all** $a \in P_i$ **do**
2: Compute $f(a)$ according to Equation 1
3: **end for**
4: $O \leftarrow \emptyset$ // Set of offspring
5: **while** $|O| < |P|$ **do**
6: Apply roulette wheel selection to select two individuals a and b from P_i
7: Apply uniform crossover or mutation to a and b to create offspring a' and b'
8: $O \leftarrow O \cup \{a', b'\}$
9: **end while**
10: $P_{i+1} \leftarrow$ the fittest $|P_i|$ individuals from $P_i \cup O$

3.2 NPGA

NPGA performs a dominance based tournament selection when choosing individuals for recombination, and when filling the new population, avoiding explicit fitness assignment. To decide a tournament between individuals a and b, a comparison set containing t_{dom} individuals is chosen from the population (if the tournament is selecting parents for recombination), or from the partially filled next population (if the tournament is selecting individuals from the parents and offspring to replace the current population). The tournament will prefer the individual which is not dominated by any of the individuals in the comparison set. If either both a and b are dominated by an individual in the comparison set, or both a and b are not dominated by any members of the comparison set, then the tournament prefers the individual which has a smaller number of members of the comparison set within a σ_{niche} radius of it (called *equivalence class sharing*). This process is outlined in Algorithm 2.

Algorithm 2 NPGA recombination and replacement.

Input: Generation i population P_i
Output: Generation $i+1$ population P_{i+1}
1: $O \leftarrow \emptyset$ // Set of offspring
2: **while** $|O| < |P|$ **do**
3: Select two individuals a and b from P_i using NPGA tournament selection as described at the start of Section 3.2
4: Apply uniform crossover or mutation to a and b to create offspring a' and b'
5: $O \leftarrow O \cup \{a', b'\}$
6: **end while**
7: $P_{i+1} \leftarrow \emptyset$
8: **while** $|P_{i+1}| < |P|$ **do**
9: Select two individuals a and b from $P_i \cup O$ using NPGA tournament selection where the comparison set is chosen from P_{i+1} if possible.
10: $P_{i+1} \leftarrow P_{i+1} \cup \{a, b\}$
11: **end while**

Choosing the comparison set from the currently filled next population is called *continuously updated sharing* [8]. It is utilized to enforce non-dominance and phenotypic diversity with respect to the generation being formed. Adjusting t_{dom} allows the domination pressure to be controlled. A larger value will give a better approximation of the niche sizes of a and b, as well as increase the chances that there are individuals in the comparison set which do not dominate a and b, making the tournament rely more on niche sizes. A smaller value will increase the chance that only one of a or b is dominated by a member of the comparison set, but will give poorer niche size estimates. Choosing a value too small will increase the probability that neither a nor b is dominated, and give poorer niche size estimates.

3.3 Constraints

When handling constraints, our general strategy is not to search too far from the fringe of infeasibility, since the constraints are usually in conflict with some of the objectives and we do not want to preclude solutions which favour these objectives. In our case the size constraint is in conflict with the query cost objective, and our strategy should explore view sets which are nearly full as much as it explores view sets which are nearly empty.

There are several possible ways with which constraints can be handled in MOEAs [6].

1. Make the constraint value into an objective, and adjust the definition of dominance so that an infeasible individual is always dominated by a feasible one.
2. Allow infeasible individuals, but implement a repair mechanism which converts an infeasible individual into one which is feasible when performing fitness calculation.
3. Do not allow infeasible individuals by repairing those which result from recombination operations.
4. Re-trying any recombination operation which results in an infeasible individual until a feasible one is generated.
5. Choose a representation or decoder function which cannot represent infeasible individuals.
6. Apply a penalty function to the fitness of individuals that violate the constraint.

We desire to have some infeasible solutions in the population based on the possibility that they may produce fit, feasible offspring. For this reason as well as those in the following discussion we choose the first two methods. With the first method, the pressure for individuals to be non-dominated also controls the pressure for them to be feasible. This allows us to control how thoroughly we explore near the fringe of infeasibility. By not considering the third objective in determining dominance of feasible individuals we do not care how close an individual is to being infeasible so long as it is feasible. However dominance is redefined for infeasible solutions to be based solely on the constraint objective, encouraging individuals which are nearer to being feasible. This is of benefit because we use uniform mutation and crossover, where the offspring are more likely to be feasible if the parents are feasible or close to feasible. For the second method we repair an infeasible individual by randomly removing views until it is feasible. The third method is similar except it will never re-combine infeasible individuals (as we would like to do). Method 4 will not either, but guarantees that feasible individuals are a representation of their parents (where as stochastic repair causes a loss of information about an individual's parents). The fifth is not chosen since it would certainly involve

a more complicated cost function and genotype, threatening efficiency. The sixth is similar to the first except it is still possible for an infeasible solution to dominate a feasible one, and it furthermore adds to the burden of tuning the algorithms.

4. EXPERIMENTAL EVALUATION

We experimentally evaluate both algorithms and constraint handling mechanisms whilst providing a comparison to the BPUS approximation algorithm presented in [12, 11] and as summarized in Section 2.1. We implemented the MOEAs in C++ using the Evolving Objects library [17], and BPUS in C. All experiments are run on a 1400 MHz AMD Athlon XP processor with 256 MB main memory running Fedora Core 3 and gcc/g++ version 3.4.2. The primary goal of the experiments is to determine the suitability and gain an understanding of the search characteristics of each algorithm/constraint handling mechanism for varying instances of the view selection problem as affected by dimensionality, and data and query skew. The comparison with BPUS was in order to provide a more absolute sense of where the genetic algorithms' solutions lie in the objective space, since BPUS guarantees a solution within 63% of the optimal. An exhaustive approach to find the pareto-optimal front would be ideal, but infeasible due to the large search space of size 2^{2^d}. Our secondary concern is scalability to larger dimensionality in terms of both running time and solution quality. Since each algorithm has search parameters we are also concerned with the sensitivity of each algorithm to these parameters, and give guidelines for choosing appropriate values.

4.1 Problem Instances

We use a total of 5 different data sets, an 8 dimensional data set of world hydrological data, and the four synthetic data sets for each combination of 10 or 12 dimensions with skewed or uniform view size distributions. All data sets have no dimension hierarchies. We estimate view sizes using the formula of [23], which takes as input the number of rows and cardinality of each dimension. The world hydrological data set (hereafter referred to as *world*) has 124676620 rows and moderate dimension cardinality variance. The 10 and 12 dimensional data sets each have 50 million rows. The view size distributions for these data sets are described as follows:

- The *uniform* view size distribution has all dimension cardinalities equal to 100, and further requires that a view is at most 90% the size of its smallest parent. This ensures a nice steady decrease in view size with the number of grouping attributes, and no variance in size of views with the same number of grouping attributes.

- The *2-pow* view size distribution has the cardinality of dimension i equal to 2^i, resulting in highly skewed view sizes.

We use a total of three different query distributions ranging in skew:

- The *uniform* query distribution has each query equally likely.

- The *gaussian* query distribution has queries distributed according to a gaussian curve over the number of grouping attributes, with a mean of $d/2$ and variance of 1.

- The *focus* query distribution, similar to the "hot regions" of [16], uniformly distributes 90% of the query distribution to a set of 10% of the views with less than $d/3$ grouping attributes selected at random. The other 10% of the query distribution is uniformly distributed to the other 90% of the views.

The *focus* distribution represents an extreme query skew, and is expected to be a reasonable scenario since: (1) users generally want a visual representation of their queries, which is more easily done in lower dimensions and (2) many of the views are likely to be uninteresting, with a select few combinations of dimensions being of the most importance. In all experiments we fix g_v, the frequency of updates on view v, to 0.01 for all v. The cost of maintaining a view v_1 using updates from one of its ancestors v_2 is equal to 10% of the size of v_2, representing 10% of the updates to each view propagating to its children. These parameters were chosen to give an equal balance in update and query costs, since too much of an imbalance causes BPUS to optimize one of the two objectives and would make for a poor comparison. S_{max} is fixed at 10% of the size of the total data cube lattice.

In all of the experiments we fix the probability of crossover/mutation at 0.6/0.4 respectively. The mutation operation has a 2% chance of changing each gene in an individual. In order to represent view sets as bit strings, views are ordered by the value of a binary code of length d indicating the grouping dimensions of each view. For this reason we use uniform crossover, since slicing a genotype in half does not correspond to a natural partition of views in the lattice.

4.2 Experimental Results

We compare the algorithms and constraint handling mechanisms on a selection of the available data sets. The number of generations and population size was 75, 100, and 150 respectively for the *world*, 10D, and 12D data sets. These values were chosen to capture a sufficient variety in the individuals relative to the problem size, and to provide reasonable run times, although these values do not represent the point at which the algorithms lose momentum. BPUS runs in roughly 0.3, 2.5, and 26 seconds CPU time on our experimental platform for the *world*, 10D, and 12D datasets respectively, where as the MOEAs run in 2 to 4, 12 to 18, and 140 to 165 seconds. A discussion of the parameters is in Section 4.3 where the effects of t_{dom}, σ_{niche} as well as further general observations are described.

Figure 2 shows a random sample of 25% of the feasible individuals in the final population of each MOEAs, as well as the solution of BPUS plotted in the objective space for four problem instances. Each run shown is a representative example of multiple independent runs, where little variance in solutions was observed. An example initial generation population is shown to serve as an indicator of how much and in which direction progress has been made. Not surprisingly, the final generations on the uniform problem instances (Figures 2-(b) and (c)) are not very distant from the initial generation in the objective space, illustrating a small difference in objective values between random and good solutions. For the highly skewed problem instance in Figure 2-(c) an initial population is not shown, as it is so far away from the final populations as to force the range of the plot to be much larger, causing a loss in visible resolution amongst the algorithms. This, being a "harder" problem instance, is the only one where one of the MOEAs largely dominates the others. MOGA with objective constraint handling tends to progress the fastest, likely a result of its strict pressure on individuals to be non-dominated from fitness calculation and roulette wheel selection, providing an absolute measure of non-dominance. NPGA's tournaments are a relative measure of non-dominance, only as good as the comparison set chosen. Hence it may take longer for NPGA to gain momentum away from the initial population when the comparison sets are random individuals.

Each of the small clusters in the final populations of Figure 2-(d) is likely a set of individuals which have some large views in common. The larger views are a sort of coarse tuning (adding a larger view is equivalent to moving from one cluster to another),

while the smaller views are the fine tuning of solutions within that cluster. This is discussed further in Section 4.3. Also in both the uniform problem instances shown in Figures 2-(b) and (d) the policy of repairing infeasible solutions has caused both NPGA and MOGA (NPGA to a greater extent) to favour a lower relative query time, shifting up and away from the initial population to individuals with a larger number of views. This suggests that mutation is the predominantly successful operation in creating non-dominated offspring, since the population is initialized so that individuals have only 10% or 2% (in the 10 and 12 dimensional cases respectively) of the total number of views, meaning mutation is likely to add views to them. Notice that in all cases except the high dimensional (Figure 2-(d)) there are solutions dominating the solution of BPUS. This suggests that dimensionality has an impact on the performance of the MOEAs. At the 150th generation the MOEAs still have enough momentum to creep past BPUS's solution in less than 50 generations, although for even higher dimensional instances it may take substantial search time to provide any solutions which compete with BPUS. Another trend in all of the tests shown in Figure 2 is that NPGA spreads out less than MOGA. This can be attributed to the fact that niching is secondary to domination in its tournament selection. We observed that 50% to 75% of all NPGA tournaments were resolved by domination alone, and hence phenotypic diversity is not frequently considered in NPGA's selection.

4.3 Tuning and Further Observations

In NPGA, the parameters t_{dom} and σ_{niche} are essentially controls which adjust the non-domination pressure, the diversity pressure, and the accuracy of the diversity measure. t_{dom} was adjusted to maximize the number of distinctions based on dominance, which we measure. We observed that this was necessary for NPGA to have sufficient non-domination pressure. However the trade-off is a lack of diversity in as seen in Figure 2. Neither MOGA nor NPGA in our application were extremely sensitive to changes in these parameters, and we found estimating σ_{niche} based on the separation of solutions in the objective space to be sufficient, aiming for a radius encompassing 10 or fewer solutions.

During the tuning process we also observed that the MOEAs tend not to spread very far from the region where the initial population lies, their behaviour is typically to quickly spread out along a front and then slowly creep forward. It is not obvious from the plots in Figure 2, but an initial population with even a slightly different proportion of 1s causes the algorithms to spread out along a different, non-intersecting front. This is because crossover does not change the number of 1s in the population, and mutation only if it is successful in producing good offspring. A possible way to alleviate this is to start with an initial population with each individual having different proportions of 1s and 0s, however this may cause crossover to have difficulty in exploiting good features in a population with so much variety.

A further experiment we conducted measures the convergence in the population based upon the dimensionality of views. At each generation, we measured the average entropy in the population members' choices for views of each number of dimensions. Entropy is a measure of disorder, and a lower entropy for k-dimensional views indicates the population's tendency to agree on which k-dimensional views should or should not be selected. Figure 3 shows some of the results. As can be seen in the highly skewed instance (Figure 3-(a)), MOGA with objective constraint handling has a direct correspondence between view dimensionality and entropy, with higher dimensional views having lower entropy. The population agrees to a large extent on which high dimensional views to include while still maintaining some diversity with respect to the lower dimensional views. Further, convergence on the high dimensional views occurs earlier than the lower dimensional views, indicating a process of fine and coarse tuning as is observed in BPUS. On the same problem, NPGA with objective constraint handling shows little correlation between view dimensionality and entropy, with a large degree of convergence on all views (Figure 3-(b)). Compared to the skewed problem in Figure 3-(a), the relationship between view dimensionality and entropy is weaker for the uniform instance in Figure 3-(c). This is reflective of our general results with the entropy experiments, which show a higher correlation between view level and convergence for the more skewed instances than the uniform. We also find that the population tends to converge more in general when we repair our infeasible solutions, however this is not shown due to space limitations.

5. CONCLUSIONS

The main contribution of our work is the application of MOEAs to the view selection problem in OLAP data warehousing. Previous approaches have linearly combined the multiple objectives into a single objective, applying heuristics, randomized search, and GAs to the resulting optimization problem. We have applied two well established non-elitist MOEAs to view selection: Fonseca and Fleming's MOGA [7], and Horn et. al.'s NPGA [13]. An important constraint in OLAP view selection is on the size of the selected set of views, and we experiment with two constraint handling methods in combination with the two MOEAs.

Our experiments on both real and synthetic data sets with varying distributions show that the MOEAs are very competitive against the leading greedy algorithm, BPUS [12]. Depending on where the initial population lies in objective space, the MOEAs were able to find solutions which dominate that of BPUS on nearly all problems. The performance gap between BPUS and the MOEAs closes with increasing problem size. We found the performance of the two MOEAs to be similar on most problem instances, however MOGA out-performs NPGA for highly skewed instances. With regards to the constraint handling techniques, the method of repairing infeasible solutions for fitness evaluation tends to cause the population to drift up towards the middle of the pareto-optimal front, indicating mutation to be more successful than crossover in producing fit offspring.

One promising direction for future work is to experiment with elitist MOEAs, e.g. [18, 5, 4]. An elitist scheme, which partitions the population based on fitness or non-dominance, has the advantages that the best individuals are given a greater opportunity to breed with one another, which should allow quicker progress towards the pareto-optimal front.

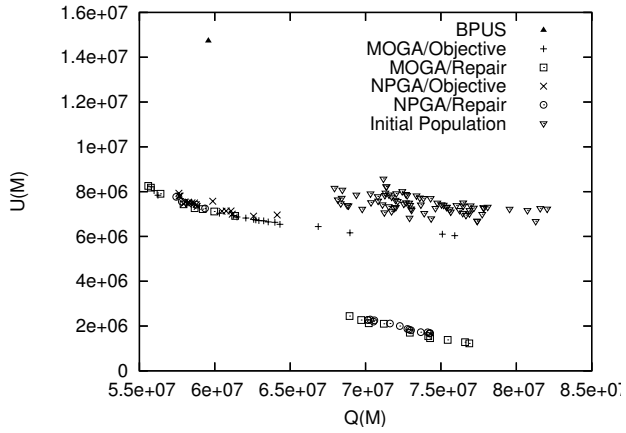
6. ACKNOWLEDGEMENTS

The author is very grateful to Malcolm Heywood for helpful suggestions and feedback during preparation of this manuscript.

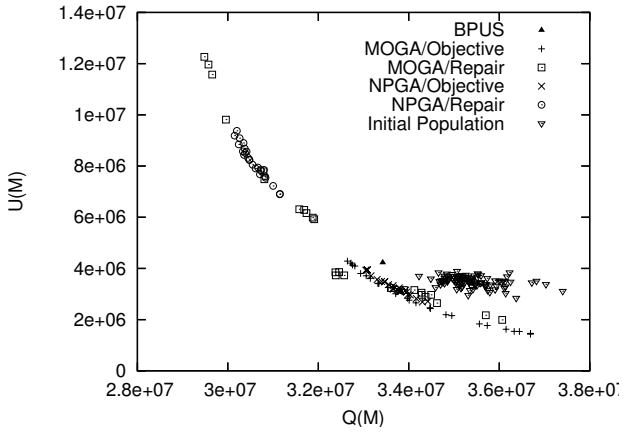
7. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *Proc. VLDB 2000*, pages 496–505, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers.

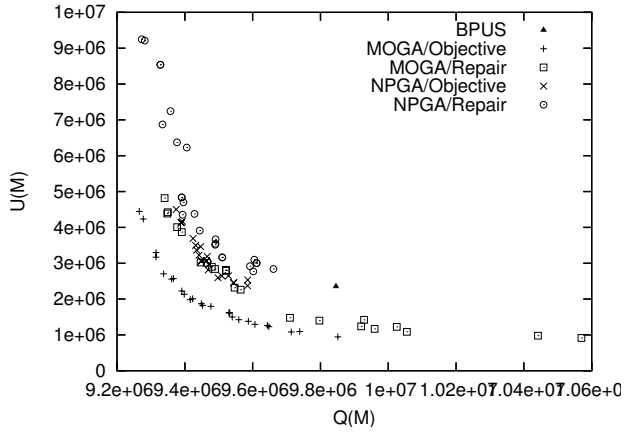
[2] E. Baralis, S. Paraboschi, and E. Teniente. Materialized views selection in a multidimensional database. In *Proc. VLDB 1997*, pages 156–165, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers.



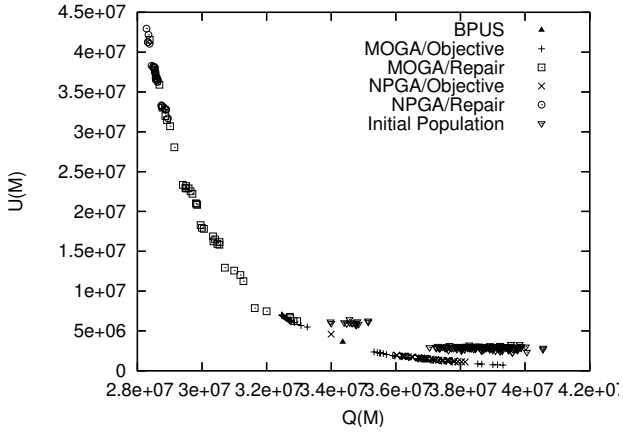
(a) *World*(8D)/*gaussian* (75th generation)



(b) 10D/*uniform*/*uniform* (100th generation)



(c) 10D/*focus*/*2pow* (100th generation)



(d) 12D/*uniform*/*uniform* (150th generation)

Figure 2: The final population of each MOEA with infeasible solutions removed, and the solution of BPUS plotted in objective space for four dimension/query/view size combinations. The population size is equal to the number of generations in each instance. For the *world* data set, the initializer probability is 40%, for 10D it is 10%, and for 12D it is 2%.

[3] R. Chirkova, A. Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. *The VLDB Journal*, 11(3):216–237, 2002.

[4] L. Costa and P. Oliveira. An elitist genetic algorithm for multiobjective optimization. *Metaheuristics: computer decision-making*, pages 217–236, 2004.

[5] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. In *Proc. PPSN 2000*, pages 849–858, London, UK, 2000. Springer-Verlag.

[6] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[7] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation discussion and generalization. In *Proc. of the 5th Intl. Conf. on Genetic Algorithms*, pages 416–423. Morgan Kaufmann Publishers., 1993.

[8] D. Goldberg, K. Deb, and J. Horn. Massive multimodality, deception and genetic algorithms. In *Parallel Problem Solving From Nature*, volume 2, pages 37–46. North-Holland, 1994.

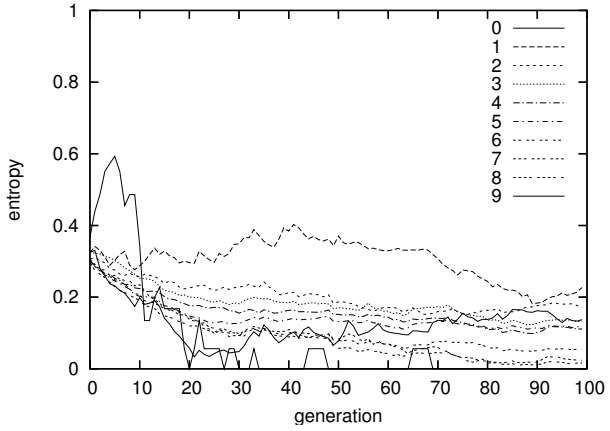
[9] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *Proc. ICDE 1997*, pages 208–219, Washington, DC, USA, 1997. IEEE Computer Society.

[10] H. Gupta and I. S. Mumick. Selection of views to materialize under a maintenance cost constraint. In *Proc. ICDT 1999*, pages 453–470, London, UK, 1999. Springer-Verlag.

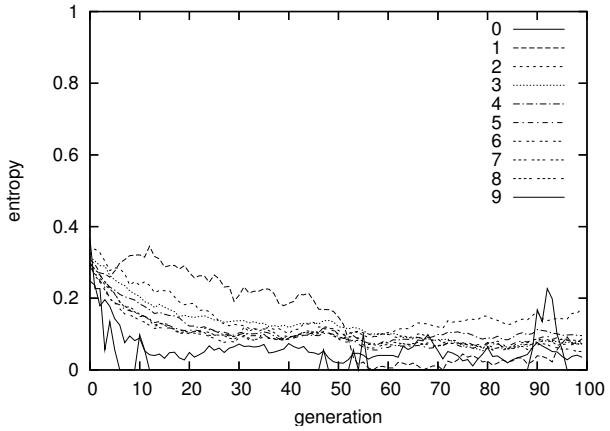
[11] H. Gupta and I. S. Mumick. Selection of views to materialize in a data warehouse. *IEEE Trans. on Knowledge and Data Eng.*, 17(1):24–43, January 2005.

[12] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. SIGMOD 1996*, pages 205–216, New York, NY, USA, 1996. ACM Press.

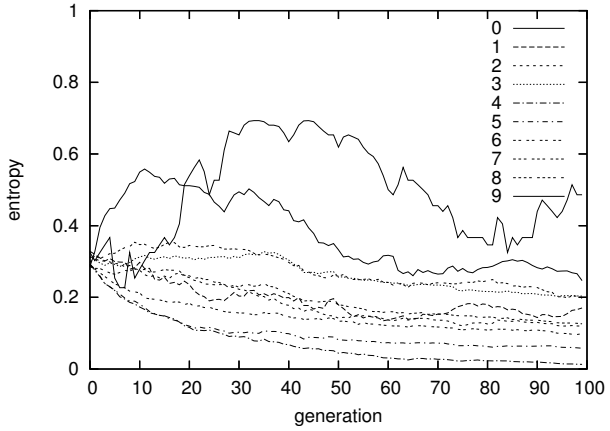
[13] J. Horn, N. Nafpliotis, and D. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proc. of the First IEEE Conf. on Computational Intelligence.*, pages 82–87, 1994.



(a) 10D/*focus*/*2pow*, MOGA/Objective



(b) 10D/*focus*/*2pow* NPGA/Objective



(c) 10D/*uniform*/*uniform* MOGA/Objective

Figure 3: The entropy of views selected within the population grouped by level (number of aggregation dimensions), plotted over the generations. The results shown are averaged over 30 independent trials each.

[14] J.-T. Horng, Y.-J. Chang, B.-J. Liu, and C.-Y. Kao. Materialized view selection using genetic algorithms in a data warehouse system. In *Proc. CEC 1999*, volume 3, pages –2227, 1999.

[15] P. Kalnis, N. Mamoulis, and D. Papadias. View selection using randomized search. *J. Data Knowl. Eng.*, 42(1):89–111, 2002.

[16] P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K.-L. Tan. An adaptive peer-to-peer network for distributed caching of OLAP results. In *Proc. SIGMOD 2002*, pages 25–36, New York, NY, USA, 2002. ACM Press.

[17] M. Keijzer, J. J. M. Guervós, G. Romero, and M. Schoenauer. Evolving objects: A general purpose evolutionary computation library. In *5th European Conf. on Artificial Evolution*, pages 231–244. Springer-Verlag, 2002.

[18] J. Knowles and D. Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Proc. CEC 1999*, pages 98–105. IEEE, 1999.

[19] M. Lee and J. Hammer. Speeding up materialized view selection in data warehouses using a randomized algorithm. *Intl. J. of Cooperative Information Systems*, 10(3):327–353, 2001.

[20] W. Liang, H. Wang, and M. E. Orlowska. Materialized view selection under the maintenance time constraint. *J. Data Knowl. Eng.*, 37(2):203–216, 2001.

[21] T. P. Nadeau and T. J. Teorey. Achieving scalability in OLAP materialized view selection. In *Proc. DOLAP '02*, pages 28–34, New York, NY, USA, 2002. ACM Press.

[22] A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. In *Proc. VLDB 1998*, pages 488–499. Morgan Kaufmann Publishers., 1998.

[23] A. Shukla, P. Deshpande, J. F. Naughton, and K. Ramasamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. In *The VLDB Journal*, pages 522–531, 1996.

[24] D. Theodoratos, T. Dalamagas, A. Simitsis, and M. Stavropoulos. A randomized approach for the incremental design of an evolving data warehouse. In *Proc. ER 2001*, pages 325–338, London, UK, 2001. Springer-Verlag.

[25] D. Theodoratos, S. Ligoudistianos, and T. Sellis. View selection for designing the global data warehouse. *J. Data Knowl. Eng.*, 39(3):219–240, 2001.

[26] D. Theodoratos and T. Sellis. Incremental design of a data warehouse. *J. Intell. Inf. Syst.*, 15(1):7–27, 2000.

[27] H. Uchiyama, K. Runapongsa, and T. J. Teorey. A progressive view materialization algorithm. In *Proc. DOLAP 1999*, pages 36–41, New York, NY, USA, 1999. ACM Press.

[28] S. R. Valluri, S. Vadapalli, and K. Karlapalem. View relevance driven materialized view selection in data warehousing environment. In *Proc. CRPITS 2002*, pages 187–196, Darlinghurst, Australia, Australia, 2002. Australian Computer Society.

[29] J. X. Yu, X. Yao, C.-H. Choi, and G. Gou. Materialized view selection as constrained evolutionary optimization. In *IEEE Transactions on Systems, Man and Cybernetics, Part C*, volume 33, pages 458–467. IEEE Computer Society, Nov. 2003.

[30] C. Zhang, X. Yao, and J. Yang. An evolutionary approach to materialized views selection in a datawarehouse environment. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 31(3):282–294, August 2001.