Using an evolutionary algorithm to solve the weighted view materialisation problem for data warehouses

Neal Wagner* and Vikas Agrawal

School of Business and Economics,
Fayetteville State University,
1200 Murchison Road, Fayetteville, NC, 28301, USA
E-mail: nwagner@uncfsu.edu
E-mail: neal.f.wagner@gmail.com
E-mail: vagrawal@uncfsu.edu
*Corresponding author

Abstract: A common problem in data warehousing is reduction of response time for ad hoc queries. To reduce query processing time, a selected number of views are materialised. Selecting the optimal number of views in a data warehouse is known to be an NP-complete problem as no feasible deterministic algorithm exists. In this paper, we discuss a weighted materialised view selection (MVS) problem where both the amount and importance of data retrieved are considered. Existing versions of the MVS problem only consider the amount of data retrieved and ignore the relative importance of data to the data warehouse user. We apply an evolutionary algorithm (EA) to solve the weighted MVS problem in which a higher priority is given to the optimisation of high-demand queries over infrequently-used queries. Several experiments are conducted on large sized datasets that are commonly found in real world applications. EA results are compared to those obtained by the best known heuristic algorithm for this weighted MVS problem from the current literature. EA is shown to outperform the heuristic algorithm for all experiments conducted.

Keywords: evolutionary algorithms; view materialisation; weighted view; optimisation; data warehouse.

Reference to this paper should be made as follows: Wagner, N. and Agrawal, V. (xxxx) 'Using an evolutionary algorithm to solve the weighted view materialisation problem for data warehouses', *Int. J. Intelligent Information and Database Systems*, Vol. X, No. Y, pp.000–000.

Biographical notes: Neal Wagner is an Assistant Professor of Information Systems at Fayetteville State University, North Carolina, USA and a Scientific Advisor for SolveIT Software a software company which specialises in supply chain optimisation software for large corporations. His research interests lie in the field of computational intelligence, particularly in the application of bio-inspired algorithms for modelling and simulation, prediction, and optimisation in real world problem domains. He has publications in well-known journals including *IEEE Transactions on Evolutionary Computation*, *Applied Financial Economics, Defence and Peace Economics*, and *Journal of Intelligent Computing and Cybernetics*. He holds a PhD in Information Technology and MS in Computer Science both from the University of North Carolina at Charlotte, and BA in Mathematics from the University of North Carolina at Asheville.

Vikas Agrawal is an Assistant Professor in the Department of Management. He teaches courses related to MIS, operations management and project management. He has several years of industrial experience in the area of project management, operations and maintenance, and software engineering. He is also a certified supply chain professional. His area of research includes database, data warehousing, optimisation techniques, agent-based simulation, knowledge management, supply chain management and operations management. He has expertise in web programming, SQL, PL/SQL, JAVA, VB.NET and has working experience with Java web server and DotNet framework. He has published papers in journals like *Journal of Database Management*, *International Journal of Data Warehousing and Mining*, *Review of Business Research*, *International Journal of Operations and Quantitative Management* and *Review of Business Information Systems*.

1 Introduction

In today's fast-paced, ever-changing and results-driven economy, information is a key business resource needed to gain a competitive business advantage. Effective use of this information requires good decision support systems (DSSs). Most DSSs require reliable and elaborate data backbone which needs to be converted into useful information. With the widespread availability and ever-decreasing cost of computers, telecommunications technologies, and internet access, most businesses have collected a wealth of data. However, that is only the first and easiest step. Many firms are becoming data rich but remain information and knowledge poor (Gray and Watson, 1998; Grover, 1998; Han and Kamber, 2006; Nemati et al., 2002). To alleviate this problem, many corporations have built unified decision-support databases called data warehouses on which decision makers can carry out their analysis. A data warehouse is a very large database that integrates information extracted from multiple, independent, heterogeneous data sources into one centralised data repository to support business analysis activities and decision-making tasks (Han and Kamber, 2006).

Business analysts run complex queries over the centralised data repository housed in a data warehouse in order to gain insights from vast amounts of data and to mine for hidden knowledge. The key to gaining such insight is to design a DSS which would get the right information to the right person at the right time so that sound strategic decisions can be made. In order to achieve this objective, design of the data warehouse architecture plays a pivotal role. There are many architectural issues concerning the efficient design of a data warehouse system. Lee et al. (2001) highlighted the importance of metadata for implementing a data warehouse. They pointed out that integrating a data warehouse with its metadata offers a new opportunity to create a more adaptive information system. Furtado (2006) proposed the concept of node partitioning, a method of parallelism, to improve the performance of a data warehouse system. Huang et al. (2005) proposed an intelligent cache mechanism for a data warehouse system in a mobile environment.

Data cube design is one such important aspect of the data warehouse architecture. Data cubes are constructs to store subsets of summarised data by some measures of interest for easy and quick access, and for timely and dynamic updates of these summarised data on an ongoing basis (Chun et al., 2004).

Accessing data from a data cube, if not materialised, can be a time consuming and resource intensive process. A data cube consists of many views[1] with existing interrelated dependencies among themselves. If such a view is stored, it is denoted as a materialised view. The problem of quick and easy access to a data cube may be alleviated by an efficient selection of a set of views to be materialised. Since not all views in a data cube may be materialised due to constraints imposed on the system, selecting the optimal set of views to materialise is a critical part of the design of a data cube and its associated views. An efficient design will dramatically reduce the execution time of decision support queries and hence is pivotal in delivering competitive advantage. However, for even moderately-sized data warehouses this can be difficult as the number of possible cuboids to consider varies exponentially with the number of tables, dimensions, and their corresponding levels. Selecting the optimal set of materialised views is essentially a combinatorial searching problem in which a subset of views must be chosen from a given set of all possible views. This problem is known to be NP-complete as no feasible deterministic algorithm exists.[2]

Many researchers have studied the problem of selecting the optimal set of views to be materialised in a data cube in order to minimise decision support query response time. The problem is generally described as the materialised view selection (MVS) problem, which has the objective of minimising the access time subject to constraints on either the number of views that may be materialised or the storage space that may be used for materialisation of views (Gupta and Mumick, 2005; Harinarayan et al., 1996, 1999). Harinarayan et al. (1999) was the first to develop a lattice framework architecture for data cube representation. While this framework proved to be a significant contribution to data warehouse design, its shortcoming lies in its lack of consideration of the relative importance of user queries. It is common in real world applications for data warehouse users to place more weight on some queries than others. This added weight may be due simply to query frequency and/or to some aspect of business logic.

For this study, we are concerned with a weighted MVS problem in which a higher priority is given to the optimisation of more important (higher weight) queries over less important (lower weight) queries. The weighted MVS problem has greater relevance for real world applications than the non-weighted version of the problem as it is highly unlikely that all user queries have equivalent importance in any realistically-sized data warehouse. Thus, the goal here is to minimise the maximum weighted number of pages to be retrieved in a data cube given a constraint on the maximum number of views that can be materialised.

The framework for the weighted MVS problem was first developed by Agrawal et al. (2007).[3] In Agrawal et al. (2007), the weighted MVS problem was formally defined and a heuristic optimisation algorithm to solve the problem was developed and tested. To date the deterministic optimisation algorithm developed in the study is the best known model for addressing the weighted MVS problem from the current literature.

In this paper, we investigate an evolutionary algorithm (EA) approach to the weighted MVS problem with a focus toward solving the problem for real world instances. EA is a technique from the field of computational intelligence and is widely-known for its applicability to optimisation problems (Michalewicz and Fogel, 2004). We develop and implement an EA optimisation model applicable for real world instances of the weighted MVS problem in which a higher priority is given to the optimisation of high-demand/important queries over low-demand/unimportant queries. In order to

evaluate the model, several experiments are conducted on large sized datasets that are commonly found in real world applications and results are compared to those obtained by the heuristic optimisation model proposed by Agrawal et al. (2007).

The contribution of this paper is in its investigation of a non-deterministic evolution-based technique for the weighted MVS problem that seeks to address dataset sizes that are practical in real world problem instances and improve upon the performance of the best optimisation model from the literature. Additionally, it is our assertion that the weighted MVS problem is more practical from a real world perspective than the non-weighted MVS problem as it does not ignore the relative importance of user queries in a data warehouse.

The rest of paper is organised as follows: Section 2 gives a review of existing studies concerning the MVS problem, Section 3 revisits the weighted MVS framework and heuristic algorithm developed by Agrawal et al. (2007) and discusses the algorithms limitations, Section 4 presents a EA approach to the weighted MVS problem, Section 5 details experiments conducted comparing the EA technique to the heuristic technique described in Section 3, Section 6 provides a discussion of the practical implications, and Section 7 concludes and offers future directions.

2 Literature review

In a typical organisation, information is spread over many different multiple, independent, heterogeneous and remote data sources. Acting as a DSS, a data warehouse extracts, integrates, and stores relevant information from these data sources into one centralised data repository to support the decision-making needs of the organisation.

Business analysts run complex business queries over this centralised data repository to gain insights from vast amounts of data and mine for hidden knowledge. For more efficient computation, results of such queries are generally pre-computed and stored ahead of time at the data warehouse in the form of materialised views. Such materialisation of views reduces the query execution time to minutes or seconds which may otherwise take hours or even days to complete.[4]

While operational databases maintain current transactional information, data warehouses typically maintain information from a historical perspective. Hence, data warehouses tend to be very large and grow over time. Also, users of DSS are more interested in identifying hidden trends rather than looking at individual records in isolation. As a result, decision support queries are much more complex than online transaction processing (OLTP) queries and call for heavy use of aggregations.

The size of the data warehouse and the complexity of queries can cause decision support queries to take a very long time to complete. This delay can be critical in a real world business environment as some strategic decisions must be made within a brief window of time. The usual requirement for most query execution times is only a few seconds or at most a few minutes (Gupta and Mumick, 2005; Harinarayan et al., 1996, 1999).

Many techniques have been discussed in the literature to improve the query response time performance goals. Verma and Kumar Vijay (2010) presented a novel greedy approach to materialised views selection in a data warehouse by first identifying frequent views (i.e., views access most frequently in the past) from among all views in the lattice, and then greedily selecting beneficial views from among these frequent views. Chen

(2011) has proposed a new algorithm called dynamic selection algorithm of materialised view (DSAMV) based on PBUS algorithm, which chooses the best view for materialisation dynamically, and further improve the efficiency of OLAP queries.

Query optimisation and query evaluation techniques can be enhanced to handle the aggregations better (Chaudhuri and Shim, 1994; Gupta et al., 1995). Also different indexing strategies like bit-mapped indexes and join indexes could be used to handle group-by(s) better (ONeil and Graefe, 1995).

One method of tackling the demand for RAM space and the need for quick response is to compute the necessary aggregate values more efficiently. Chun et al. (2004) provide an example of a new representation for the prefix sum cube which reduces storage space and update times.

Judicious management of data warehouses can also improve overall performance efficiency and query response time. For a recent successful incorporation of data governance strategies and its impact on data warehouse management in Blue Cross and Blue Shield of North Carolina, please refer to Watson et al. (2004).

The focus of our paper is to improve access times across users in an egalitarian or prioritised fashion in a data warehouse. A commonly used technique to improve net access time is to materialise (pre-compute and store) the results of frequently asked queries. But picking the optimal set of views to materialise is a non-trivial task. For example, one may want to materialise a relatively infrequently asked query if it helps in answering many other frequent queries faster. One cannot materialise all the views in a given data cube as such materialisation is constrained by the availability of storage space, view maintenance cost, and computational time. On the other extreme, if one does not materialise any view then business queries have to be run over the source data a process that would take considerable time leading to intolerable delays. Between these two extremes, one needs to find the optimum number of views to be materialised that will give reasonably good query response time while satisfying all the constraints. The MVS problem with the objective of minimising sum of access times has been shown to be NP-complete (Harinarayan et al., 1999). The weighted MVS problem, which is the focus of this paper, is likely to be difficult to solve optimally as it attempts to minimise the maximum weighted number of pages to be retrieved. Next, we address the literature specifically related to the materialised views.

In the 1980s, materialised views were investigated to speed up the data retrieval process for running queries on views in very large databases (Adiba and Lindsay, 1980). Subsequently, further research studies were reported in view and index maintenance along with comparative evaluations of materialised views on the performance of queries (Blakeley and Martin, 1990; Qian and Wiederhold, 1991; Segev and Fang, 1991).

Gray et al. (1997) proposed the data cube as a relational aggregation operator generalising group-by, cross-tabs, and subtotals. Harinarayan et al. (1996, 1999) have discussed the major features of the MVS problem in detail. They employed a lattice framework to capture the dependencies among views. This lattice framework was then used to develop a greedy algorithm. Zhang and Yang (1999) proposed a genetic algorithm approach to solve the non-weighted MVS problem and demonstrated that it was practical and effective compared with heuristic approaches.

Kalnis et al. (2002) used a randomised local search algorithm to generate the optimal set of views. Park et al. (2002) assume that the set of materialised views present is given and then ask the question: How do we to rewrite the given OLAP query to make the best

use of existing materialised views? They developed algorithms for rewriting and identifying materialised views that will best answer a given query.

Ashadevi and Balasubramanian (2009) presented a framework for selecting views to materialise so as to achieve the best combination of good query response, low query processing cost and low view maintenance cost in a given storage space constraints. For more in-depth study about various methodologies to select materialised view that exist in the literature, one can refer to Dhote and Ali (2009), and Mami and Bellahsen (2012).

Agrawal et al. (2007) proposed an efficient heuristic technique to solve the weighted MVS problem. Their results show that the heuristic technique yields efficient solutions with significantly reduced computation time when compared to a linear integer programming technique. The next section details the weighted MVS problem and the heuristic technique developed by Agrawal et al. (2007), discusses the heuristics limitations, and outlines potential directions for improvement.

3 Heuristic approach to weighted MVS problem

As mentioned above, Agrawal et al. (2007) developed the framework for the weighted MVS problem and a heuristic technique to solve it. The motivation for the framework stems from the desire to include a measure of query importance into the original MVS framework developed by Harinarayan et al. (1999). As mentioned in Section 1, typically not all queries are of equal importance to data warehouse users. Some queries have higher weight due to either frequency of use and/or domain specific business logic. Any practical algorithm for data warehouse design must take into account query weight when optimisation is performed. For example, suppose two view materialisation algorithms, one that considers query weight and one that does not, are to be compared on a single set of all available views for a particular data warehouse. If the set of views contains queries of unequal weights, then the algorithm that ignores these weights is likely to leave out the materialisation of highly important views (i.e., views with higher weight) in favour of less important views. In contrast, the view materialisation algorithm that takes into account query weight is more likely to avoid this design error. This study is concerned with optimal design for practical data warehouse applications and, thus the weighted MVS problem is considered.

Agrawal et al. (2007) developed an efficient heuristic algorithm to solve the weighted MVS problem. The motivation for this heuristic stems from the desire to handle larger problem instances that are prevalent in many real world data warehouses. Brute force (BF) and linear integer programming techniques guarantee the optimal solution but are not feasible for data warehouses with a numerous cuboids because the computation time increases exponentially with the size of data cube.

The heuristic is essentially a greedy algorithm which builds a set of views to be materialised. The first view included in the set is the root node of the data cube. From this point on, the heuristic searches for the best new node to add to the set such that the Z value[5] is minimised. It continues adding new nodes to the set one at a time until there is no reduction in Z value upon checking all remaining nodes.

Although the heuristic performed well for the experiments conducted, it suffers from some notable shortcomings which are particularly pronounced when very large problem instances are examined. Consider the data cube example given by Agrawal et al. (2007, pp.4–8) shown in Figure 1.

Figure 1 A graphical representation of a data cube with associated views

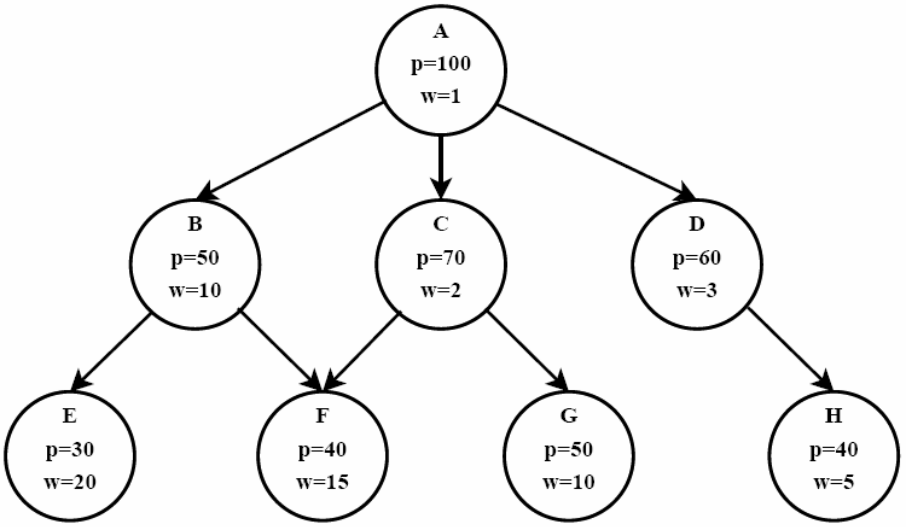


Figure 1 presents a data cube and its associated views organised as a lattice framework for a hypothetical data warehouse. Each node represents a view (i.e., cuboid) and the numbers inside each node represent the number of pages (p) that must be retrieved to respond to the underlying query and the weight (w) that is associated with each view, which is the function of frequency of access and/or domain specific business logic.

The objective is to find a subset of views to be materialised such that the Z value is minimised given a constraint on the maximum number of views that can be included. If for example the constraint on views is 5 for the above problem, then the heuristic builds a set of views by first including the root node A with Z value = 2,000. It then checks all remaining nodes by adding them each in turn to the current set and calculating the new Z value. It will select a new node to add to the set such that the Z value is minimised. For this example, the second node to be added to the set is B which generates a minimum Z value = 1,000. The heuristic repeats this procedure until checking all remaining nodes yields no improvement in Z value. For this example, the addition of any other node to the current set {A, B} does not reduce the Z value, and thus the heuristic completes.

The advantage of this heuristic is that it significantly reduces computation time required to generate the set of materialised views. The shortcoming is that the solution quality can be quite poor for larger problem instances that are commonly seen in real world data warehouses. This shortcoming can be seen even in the above example which is a very small instance of the weighted MVS problem. The optimal set of materialised views for the above example is {A, B, E, F, G} which yields a Z value = 600. The heuristics failure to achieve this optimal set lies in its greedy strategy which considers only one view at a time. Once the heuristic has built the set {A, B} with Z value = 1,000, it considers each remaining view by adding it to the set and calculating the new Z value. At this point, adding any single node to the set yields no improvement in Z value. However, adding nodes E and G simultaneously to the set would reduce the Z value to

750 as the most constrained node would now be F. Thus, the heuristic terminates prematurely and fails to find the optimal solution.

In larger instances of the weighted MVS problem this issue becomes increasingly prevalent. The heuristics failure is caused by nodes with identical weighted number of pages. In the above example when the set contains {A, B}, nodes E and G have identical weighted number of pages = 1,000. For the heuristic to reduce this value further it should add both of these nodes to the set but cannot as it only considers one node at a time. For the heuristic to successfully progress to the optimal, all nodes of the data cube tree must have unique values for weighted number of pages at every step of the procedure. The probability of this being true decreases as the size of the data cube tree increases. In practical data cube trees, it is highly unlikely that nodes with identical weighted number of pages would not occur at any step of the heuristic procedure. The next section proposes an EA approach to the weighted MVS problem that overcomes this issue.

4 EA approach to the weighted MVS problem

As mentioned in Section 1, EA is a well-known optimisation technique that has been successfully applied to a wide variety of problems. EAs use the principle of survival of the fittest as a model for building computer programmes that can solve problems in a way that is commonly found in nature. The idea is to view the problem as a search through a space of potential solutions. The EA procedure defines a method to search through this space and locate the optimal solution. For problems in which the number of potential solutions is not too large, a BF approach is sufficient. For problems with larger solution spaces such techniques are infeasible. As mentioned above, the weighted MVS problem for even moderately sized data warehouses boils down to a search through a very large solution space and is known to be NP-complete.

The general procedure of the EA search process is as follows:

1 A population of candidate solutions is randomly (or otherwise) created.

2 Each solution is ranked based on its proximity to the optimal.

3 A new population of solutions is generated by selecting fitter solutions from the population and performing genetic operations.

4 Common genetic operations include mutation and/or crossover. Mutation alters a single solution while crossover produces a new solution that is some combination of two original solutions.

5 This process of generating new populations is repeated for some set number of iterations or until no improvement in solution quality is seen.

6 The final solution is the highest ranking solution from the last population generated.

For the weighted MVS problem, The EA approach represents candidate solutions as sets of data cube tree nodes such that the following constraints are followed:

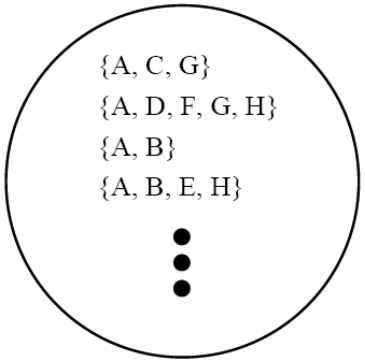
1 The root node must be included in the set.

2 No duplicate nodes should be included in the set.

3 Set size should be between 2 and *n* where *n* is the maximum number of views that can be materialised.

Figure 2 gives an example EA initial population for the data cube tree shown in Figure 1. The figure represents a potential population of EA candidate solutions with maximum number of views n = 5. Note that each candidate solution in the figure adheres to the constraints listed above. In the next step of the EA procedure candidate solutions are ranked. For the weighted MVS problem, solutions are ranked based on their Z values. The solutions shown in Figure 2 have the following Z values: {A, C, G} Z value = 2,000, {A, D, F, G, H} Z value = 2,000, {A, B} Z value = 1,000, {A, B, E, H} Z value = 1,000. The next step specifies that fitter candidate solutions are selected for genetic operations. In this case, fitter solutions are those with lower Z values. For the case of multiple solutions with equivalent Z values, solutions that specify a smaller set of views are favoured. The selection process is probabilistic in nature and works in the following way: two candidate solutions from the initial population are randomly selected; after comparison the one with the lower Z value is chosen. Say for example solutions {A, D, F, G, H} and {A, B} are randomly chosen for comparison. {A, B} is selected to undergo genetic operations based on its lower Z value.

Figure 2 Example EA initial population



For the weighted MVS problem, the genetic operations employed are the following three mutation operations: add view(s) mutation, remove view(s) mutation, and replace view(s) mutation. For this application, genetic crossover operators were not used because combining views from two candidate solution sets typically results in duplication of views, and thus violates one of the problem constraints. As mentioned above, the genetic mutation operator alters a single candidate solution. The procedure used to execute the mutation operation is as follows:

1 First, one of the three mutation operators (add, remove, or replace view(s)) is chosen at random.

2 Then, the procedure randomly selects a number of views to alter that is within the range defined by the minimum number of views (2) and the maximum number of views (*n*) to be materialised.

3 The procedure then randomly selects the specific views from the candidate solution's view set to be altered such that they do not violate the constraints listed above.

4 Finally, the select views from the candidate solution are altered in the way specified by the selected mutation operator.
   - If the selected mutation operator is 'add', then the selected views are added to the candidate's view set. It is important to note that when the add mutation operator is selected only views not already included in the candidate's view set are selected for addition so as not to cause duplication of views.
   - If the selected mutation operator is 'remove', then the selected views are removed from the candidate's view set. It is important to note that the root node is never selected for removal so as not to violate problem constraints.
   - Selection of the 'replace' mutation operator is the equivalent of consecutive execution of the remove and add mutation operations, respectively.

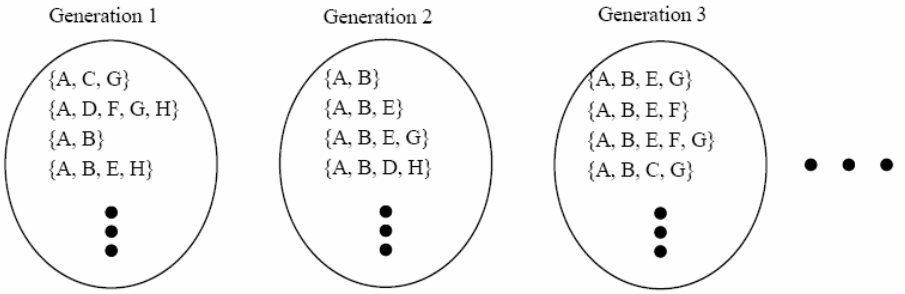
For example, if candidate solution {A, B} has been selected for mutation, then the EA procedure randomly chooses a mutation operation to be executed. If for example the selected operation is the add view(s) operation, then the procedure randomly selects a number of views to add to the candidate set {A, B} such that none of the constraints mentioned above are violated. Say two new views, E and G are added during this operation yielding set {A, B, E, G} with Z value = 750. This mutated candidate solution is then inserted into a new population. Mutated candidate solutions are generated and inserted in this way until a population of sufficient size is formed. A new population is complete when it has the same number of candidate solutions as its predecessor population.

There is one special case to consider when generating a new population, that of the best candidate from the previous population. The best candidate is directly copied into the next population unaltered. This procedure is known as *elitism* (Michalewicz, 1998) and is commonly used to preserve the best ranking candidate solution from generation to generation during the evolutionary search process.

Figure 3 depicts an example progression of EA generations for the data cube tree of Figure 1. In the figure, each oval represents a new population created by the procedure described above. Note that in the figure each subsequent generation includes a candidate solution that is at least as good as or better than the best ranking candidate of the previous generation.

The EA search process has the capability to overcome the shortcomings of the heuristic technique described in Section 3 as it can consider more than one new node to add, remove, or replace from the set when building the optimal set of views to be materialised. Thus, the problem of nodes with identical weighted number of pages that is not overcome by the heuristic technique can potentially be overcome by the EA approach. In preliminary experiments, the EA systematically found the optimal set of materialised views, set {A, B, E, F, G} with Z value = 600, for the data cube tree of Figure 1 while the heuristic technique could only produce set {A, B} with Z value = 1,000. The next section details experiments that compare the EA approach to the heuristic algorithm and a BF technique and provides detailed analyses.

Figure 3 Example EA generations



5 Experiments

The purpose of this study is to investigate larger, real world instances of the weighted MVS problem. As discussed in Section 1, we seek to compare the proposed EA optimisation model with the heuristic optimisation model of Agrawal et al. (2007). With this in mind, data cube trees of sizes from ranging from 2^7 to 2^{10} nodes are constructed and the EA and heuristic optimisation models are executed on each of these trees. For comparison purposes, we also add a BF optimisation model to the experiments.

Additionally, different constraints on the maximum number of views to be materialised are enforced for each of the data cube trees constructed. These differing constraint specifications are necessary because they affect the solution space size. For example, for a data cube tree of 2^n nodes, a constraint of k views to be materialised gives a solution space of size ${}_{(2^n-1)}C_{(k-1)}$ because the root node is always materialised. Thus, for a data cube tree with $n = 10$ and $k = 5$ the solution space size = 4.54×10^{10}. If constraint k is increased to 20 views, the solution space size increases to 1.07×10^{40}. This trend continues as constraint k is increased until its value reaches a number that is equal to $\frac{1}{2}$ the total number of nodes in the data cube tree.

For our experiments, data cube trees of sizes 27, 2^8, 2^9, and 2^{10} are constructed with randomly specified weights and number of pages and the constraint on the maximum number of views to be materialised is varied from 5 to 20 in steps of 5 for each data cube tree. The EA, heuristic, and BF techniques are executed for each data cube tree/constraint combination and the performance of each technique is measured.

EA experiments typically require specification of the following parameters that govern the search process:

1 candidate solution population size

2 maximum number of generations per run

3 total number of EA runs to execute per experiment.

These parameters are concerned with controlling the amount of computational resources required to execute EA experiments.

The candidate solution population size gives the number of solutions that are to be created for each generation of an EA run. An EA generation includes the creation and

evaluation of a single population of candidate solutions. The maximum number of generations controls the duration of a single EA run.

The EA procedure is essentially a fitness-driven stochastic search through a space of potential solutions. This process mimics the evolutionary selection process observed in nature. Because the procedure includes stochasticity, results can vary from one EA run to another. Therefore, it is necessary to execute multiple EA runs for a single experiment and select the best solution generated over all runs. For all experiments conducted in this study, the parameter values used are given in Table 1. Population size and number of generations govern the memory usage and duration of a single EA run and values for these parameters must be selected based on available computational resources such as RAM capacity and processor speed. Number of EA runs per experiment governs the duration of a single EA experiment and the value for this must be selected based on available processor speed. The values specified in Table 1 were selected based on preliminary experiments to determine feasible running times and memory usage.[6]

Table 1 Parameters for EA experiments

Parameter	*Value*
Population size	25
No. of generations per EA run	500
No. of EA runs per experiment	10

Experimental results comparing the BF, heuristic, and EA techniques are given in Tables 2, 3, and 4. Table 2 gives the maximum weighted number of pages (Z value) in units of 10^7. In the table, the Z value is shown for data cube trees of 128 (2^7), 256 (2^8), 512 (2^9), and 1,024 (2^{10}) nodes with the maximum number of views to be materialised constraint varied from 5 to 20 in steps of 5 for each of the three techniques. Table 3 gives the number of views actually materialised in the solutions produced by each of the three techniques. The values shown in the table correspond to same experiments as given in Table 2. Table 4 is constructed in the same way as Tables 2 and 3 and gives computation times in minutes for each experiment. It is important to note that the times for EA experiments shown in Table 4 do not represent the time for a single EA run, but instead represent the total time taken to execute an entire set of EA runs with the same experimental conditions. As discussed above, an EA experiment consists of a set of multiple EA runs.

In Table 2, for data cube tree of 128 nodes and 5 maximum views constraint, the BF technique yields the optimal Z value of 8.6×10^7, the heuristic (H) technique gives a Z value of 8.73×10^7, while the EA technique also yields the optimal Z value of 8.6×10^7. As can be seen in the corresponding cells of Table 4, the computation times for BF, H, and EA techniques are 60.44, 0.07, and 0.59 minutes, respectively. Although the BF technique guarantees the optimal solution, its shortcoming lies in its much greater running time for even smaller data cube trees. For the data cube tree of 128 nodes when the maximum number of views to materialise is constrained to 5, the total number of solutions in the search space is ${}_{127}C_4 \approx 1.03 \times 10^7$. This rather large number causes the extra computation time for the BF technique. When the maximum number of views to be materialised constraint is increased to 10, the search space size increases to ${}_{127}C_9 \approx 1.77 \times 10^{13}$. This exponential increase in search space size causes a similar increase in computation time for the BF technique. The other experiments all have a search space

size of equal or greater than this, and thus the BF technique could not be run for these. In Tables 2, 3, and 4, the symbol '*' represents an experiment that could not complete due to extensive computation time.

Table 2 Experimental results: max. weighted no. of pages (Zval) in units of 10^7

		Data cube trees											
		128 nodes			*256 nodes*			*512 nodes*			*1,024 nodes*		
		BF	*H*	*EA*	*BF*	*H*	*EA*	*BF*	*H*	*EA*	*BF*	*H*	*EA*
		Zval	*Zval*	*Zval*	*Zval*	*Zval*	*Zval*	*Zval*	*Zval*	*Zval*	*Zval*	*Zval*	*Zval*
Max. views	5	8.6	8.73	8.6	*	9.7	9.4	*	10	9.6	*	10	9.9
	10	*	8.73	8.6	*	9.7	8.6	*	10	8.7	*	10	9.7
	15	*	8.73	8.6	*	9.7	8.6	*	10	8.46	*	10	8.5
	20	*	8.73	8.6	*	9.7	8.6	*	10	7.65	*	10	8.6

Table 3 Experimental results: no. of views materialised

		Data cube trees											
		128 nodes			*256 nodes*			*512 nodes*			*1,024 nodes*		
		BF	*H*	*EA*	*BF*	*H*	*EA*	*BF*	*H*	*EA*	*BF*	*H*	*EA*
		#vws	*#vws*	*#vws*	*#vws*	*#vws*	*#vws*	*#vws*	*#vws*	*#vws*	*#vws*	*#vws*	*#vws*
Max. views	5	5	4	5	*	2	4	*	1	5	*	1	3
	10	*	4	5	*	2	6	*	1	9	*	1	4
	15	*	4	5	*	2	6	*	1	8	*	1	8
	20	*	4	5	*	2	6	*	1	9	*	1	7

Table 4 Experimental results: computation time (t) in minutes

		Data cube trees											
		128 nodes			*256 nodes*			*512 nodes*			*1,024 nodes*		
		BF	*H*	*EA*	*BF*	*H*	*EA*	*BF*	*H*	*EA*	*BF*	*H*	*EA*
		t	*t*	*t*	*t*	*t*	*t*	*t*	*t*	*t*	*t*	*t*	*t*
Max. views	5	60.44	0.07	0.59	*	0.02	1.58	*	0.03	7.65	*	0.11	15.82
	10	*	0.02	1.11	*	0.02	3.8	*	0.03	17.7	*	0.11	25.51
	15	*	0.01	1.65	*	0.02	4.33	*	0.03	27.76	*	0.11	44.51
	20	*	0.01	2.16	*	0.02	5.03	*	0.03	41	*	0.11	28.25

For the remainder of experiments on the 128 node data cube tree in which the maximum number of views constraint is increased to 10, 15, and 20, the EA consistently outperforms the H technique. Note that for each of these experiments the number of views that the EA materialises is 5 regardless of the more relaxed constraint specifications. This is attributed to the fact that the EA process places a higher value on solutions with a smaller number of views to materialise when the Z values of comparable solution sets are equivalent.

As can be seen in Tables 2 and 3, EA results consistently outperform H results for the remaining data cube trees of 256, 512, and 1,024 nodes. In fact, for the larger problem

instances of 512 and 1,024 node trees, the H technique fails to find a non-trivial solution that is it reports a solution materialising only the root node of the data cube tree.[7] As described in Section 3, larger problem instances increase the probability of nodes with identical weighted number of pages and the H technique fails to prevail over such a circumstance as it only considers one view at a time. It is for this reason that the H technique is unable to find a non-trivial set of views to materialise. As described in Section 4, the motivation for the EA technique lies in its ability to overcome this issue and this is reflected in the results of Tables 2 and 3.

As can be seen in Table 4, the computation time of the H technique remains relatively stable as the data cube tree size is increased from 128 to 1,024 nodes. This is due to the fact that for larger problem instances the H technique is unable to generate anything other than the root view, and thus terminates its search process prematurely. Conversely, the computation time of the EA technique increases with the size of the data cube tree. This is expected as the EA techniques search process does not terminate prematurely and can continue to improve itself through multiple generations. Although the EA running time does increase with the size of the problem, the increases are still within reasonable limits. The maximum running time seen for these experiments occurs for the 1,024 node data cube tree and is only approximately 45 minutes. From a practical perspective, this is quite short as the running time for even fairly common business queries can take as long as days to complete.

6 Practical implications

As discussed in the previous section, EA outperforms the H technique for experiments of larger instances of the weighted MVS problem. In general, real world data warehouses represent very large instances of the weighted MVS problem, and thus the EA technique is a viable approach for such situations. Deterministic approaches tend to perform inadequately when applied to large NP-complete problems such as MVS problems (Sedgewick, 1988). This is because for these problems the size of the search space increases exponentially with the size of the problem and deterministic algorithms often generate a solution that represents a local optimum rather than a global one. For such applications non-deterministic algorithms such as EA are the only practical alternative. Here, the EA succeeds due to its ability to overcome the issue of nodes with identical weighted number of pages whereas the H technique cannot. Additionally, the computation time for EA remains within reasonable limits. For the experiments conducted, the longest running EA execution lasted only approximately 45 minutes. Thus a practitioner could employ an EA to select the optimal set of views to materialise for even a large data warehouse in such a way that the selection could be completed in a reasonable time frame.

In fact, it is quite likely that a practitioner using the proposed EA optimisation model could expect results at least as good as or better than the results of this study for the same data cube trees. This is because for our experiments it was necessary to select a maximum number of EA runs per experiment that was feasible in light of the multiple experiments (each experiment including a different tree size and maximum view constraint) that needed to be executed. A practitioner in a real setting would not need to run numerous EA experiments; he/she would simply run a single EA experiment with a single tree size and maximum view constraint. This would allow him/her to select a significantly larger

number of EA runs to execute for the experiment and would make the likelihood of equalling or improving upon the results seen here to be quite high.

Moreover, it is our assertion that solving the weighted MVS problem is more practical from a real world perspective than solving the traditional MVS problem described by Harinarayan et al. (1999). This is because the traditional MVS problem ignores the relative importance of user queries in a data warehouse.

7 Conclusions and future directions

Data warehouses are seen as a strategic weapon to gain competitive advantage for businesses. A data warehouse extracts, integrates, and stores relevant information from multiple, independent, and heterogeneous data sources into one centralised data repository to support the information needs of decision makers in an organisation. Business analysts run complex business queries over the data stored at the data warehouse to mine the valuable information and identify hidden business trends. Results of such queries are generally pre-computed and stored ahead of time at the data warehouse in the form of materialised views. This drastically reduces the query execution time to minutes or seconds which otherwise may take hours or even days to complete.

There are many architectural issues involved in the design of a data warehouse. In this context, selecting the optimal set of views to be materialised in a data cube is a major concern. In this paper, we have presented an EA approach for solving the weighted MVS problem that is applicable to larger problem instances that are prevalent in real world data warehouses. The EA approach is compared to the best-known deterministic heuristic approach from the recent literature as well as a BF technique and results are reported for large data cube trees ranging in size from 2^7 to 2^{10} number of nodes.

Experimental results show that the EA approach significantly outperforms both the BF and heuristic approaches in terms of solution quality and feasibility of computation time. These findings indicate the viability of the EA approach to real world data warehouse applications and suggest further directions of inquiry. One such future study might focus on validating the proposed EA technique against industrial class data warehouses in a real world setting.

Another promising direction of future work is in the area of parallelisation. The experiments in this study use an EA that is serial in nature, that is it executes all steps one after the other. The performance of the EA could be improved in terms of computation time if parallel processing is used. Parallel processing involves the execution of independent EA functions (such as solution evaluation) and/or independent EA runs in parallel in a multi-processor environment.

Another potential direction is to combine the EA approach with a Tabu search technique. Tabu search is a heuristic optimisation technique that can enhance local search techniques by using memory structures. Once a solution has been generated by the EA it is marked as taboo meaning that it should not be revisited again in future generations so that the algorithm does not duplicate its search efforts repeatedly. This allows the EA to more efficiently explore the solution search space and may lead to higher quality solutions.

References

Adiba, M.E. and Lindsay, B. (1980) 'Database snapshots', *Proceedings of the 6th International Conference on Very Large Databases (VLDB 1980)*, Montreal, Canada, pp.86–91.

Agrawal, V., Sundararaghavan, P.S., Ahmed, M. and Nandkeolyar, U. (2007) 'View materialization in a data cube: optimization models and heuristics', *Journal of Database Management*, Vol. 18, No. 3, pp.1–20.

Ashadevi, B. and Balasubramanian, R. (2009) 'Optimized cost effective approach for selection of materialized views in data warehousing', *International Journal of Computer Science and Technology*, Vol. 9, No. 1, pp.21–26.

Blakeley, J.A. and Martin, N.L. (1990) 'Join index, materialized view, and hybrid hash join: a performance analysis', *Proceedings of the 6th IEEE Conference on Data Engineering (ICDE 1990)*, Los Angeles, USA, pp.256–263.

Chaudhuri, S. and Shim, K. (1994) 'Including group-by in query optimization', *Proceedings of the 20th International Conference on Vary Large Databases (VLDB 1994)*, Santiago, Chile, pp.354–366.

Chen, W. (2011) 'Dynamic modulating strategy of materialized views in data warehouse', *3rd International Conference on Data Mining and Intelligent Information Technology Applications (ICMiA)*, Coloane, Macao, pp.102–104.

Chun, S., Chung, C. and Lee, S. (2004) 'Space-efficient cubes for OLAP range-sum queries', *Decision Support Systems*, Vol. 37, No. 1, pp.83–102.

Dhote, C.A. and Ali, M.S. (2009) 'Materialized view selection in data warehousing: a survey', *Journal of Applied Sciences*, Vol. 9, No. 3, pp.401–414.

Furtado, P. (2006) 'Node partitioned data warehouses', *Journal of Database Management*, Vol. 17, No. 2, pp.43–61.

Gray, J., Chaudhuri, S., Bosworth, A., Layman, D., Reichart, D. and Venkatrao, M. (1997) 'Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals', *Data Mining and Knowledge Discovery*, Vol. 1, No. 1, pp.29–53.

Gray, P. and Watson, H.J. (1998) *Decision Support in the Data Warehouse*, Prentice Hall, New Jersey.

Grover, R. (1998) 'Identification of factors affecting the implementation of data warehouse', PhD dissertation, Auburn University, USA.

Gupta, A., Harinarayan, V. and Quass, D. (1995) 'Generalized projections: a powerful approach to aggregation', *21st International Conference on Very Large Databases (VLDB 1995)*, Zurich, Switzerland, pp.358–369.

Gupta, H. and Mumick, I.S. (2005) 'Selection of views to materialize in a data warehouse', *IEEE Transaction on Data and Knowledge Engineering*, Vol. 17, No. 1, pp.24–43.

Han, J. and Kamber, M. (2006) *Data Mining Concepts and Techniques*, Morgan Kaufmann, San Francisco.

Harinarayan, V., Rajaraman, A. and Ullman, J.D. (1996) 'Implementing data cubes efficiently', *Proceedings of the ACM-SIGMOD International Conference on Management of Data (SIGMOD 1996)*, Montreal, Canada, pp.205–216.

Harinarayan, V., Rajaraman, A. and Ullman, J.D. (1999) 'Implementing data cubes efficiently', in Gupta, A. and Mumick, I. (Eds.): *Materialized Views: Techniques, Implementation and Applications*, pp.343–360, MIT Press.

Huang, S.M., Lin, B. and Deng, Q.S. (2005) 'Intelligent cache management for mobile data warehouse', *Journal of Database Management*, Vol. 16, No. 2, pp.46–65.

Kalnis, P., Mamoulis, N. and Papadias, D. (2002) 'View selection using randomized search', *Data and Knowledge Engineering*, Vol. 42, No. 1, pp.89–111.

Knuth, D. (2011) *The Art of Computer Programming: Combinatorial Algorithms*, Addison-Wesley, Boston, MA, USA.

Lee, H., Kim, T. and Kim, J. (2001) 'A metadata oriented architecture for building datawarehouse', *Journal of Database Management*, Vol. 12, No. 4, pp.15–25.

Mami, I. and Bellahsen, Z. (2012) 'A survey of view selection methods', *ACM SIGMOD Record*, Vol. 41, No. 1, pp.20–29.

Michalewicz, Z. (1998) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, New York, NY, USA.

Michalewicz, Z. and Fogel, D. (2004) *How to Solve it: Modern Heuristics*, Springer, New York, NY, USA.

Nemati, H.R., Steiger, D.M., Iyer, L.S. and Herschel, R.T. (2002) 'Knowledge warehouse: an architectural integration of knowledge management, decision support artificial intelligence and data warehousing', *Decision Support Systems*, Vol. 33, No. 2, pp.143–161.

ONeil, P. and Graefe, G. (1995) 'Multi-table joins through bitmapped join indices', *ACM SIGMOD Record*, Vol. 24, No. 3, pp.8–11.

Park, C., Kim, M.H. and Lee, Y. (2002) 'Finding an efficient rewriting of OLAP queries using materialized views in data warehouses', *Decision Support Systems*, Vol. 32, No. 4, pp.379–399.

Qian, X. and Wiederhold, G. (1991) 'Incremental recomputation of active relational expressions', *IEEE Transaction on Knowledge and Data Engineering*, Vol. 3, No. 3, pp.227–341.

Sedgewick, R. (1988) *Algorithms*, Addison-Wesley, Boston, MA, USA.

Segev, A. and Fang, W. (1991) 'Optimal update policies for distributed materialized views', *Management Science*, Vol. 37, No. 7, pp.851–870.

Verma, A. and Kumar Vijay, T.V. (2010) 'Materializing views in data warehouse', *Communications in Dependability and Quality Management*, Vol. 13, No. 3, pp.76–85.

Watson, H.J., Fuller, C. and Ariyachandra, T. (2004) 'Data warehouse governance: best practices at Blue Cross and Blue Shield of North Carolina', *Decision Support Systems*, Vol. 38, No. 3, pp.435–450.

Zhang, C. and Yang, J. (1999) 'Genetic algorithm for materialized view selection in data warehouse environments', *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery (DaWak99)*, pp.116–125.

Notes

1 Views in a data cube are sometimes called cuboids (Han and Kamber, 2006).

2 For a detailed discussion of the combinatorial searching problem and its computational complexity, please consult Knuth (2011).

3 In Agrawal et al. (2007), the weighted MVS problem is referred to as a bottleneck MVS problem.

4 For a discussion of knowledge warehouses, please see Nemati et al. (2002).

5 In Agrawal et al. (2007), the maximum weighted number of pages is referred to as the Z value.

6 All experiments were executed on an Intel(R) CoreTM 2 Duo CPU with 2.50 GHz processor and 4 GB RAM on Windows 7 Pro, 64-bit operating system.

7 The root node of a data cube tree is always materialised by default (Agrawal et al., 2007).