VIEW MATERIALIZATION

A dissertation submitted to the Jawaharlal Nehru University
in partial fulfillment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE AND TECHNOLOGY

BY

ALOKE GHOSHAL

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

JULY 2008

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

DECLARATION

This is to certify that the dissertation titled "**View Materialization**" being submitted to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Computer Science & Technology**, is a record of bonafide work carried out by me.

The matter embodied in the dissertation has not been submitted in part or full to any University or Institution for the award of any degree or diploma.

Aloke Ghoshal
(06/10/MT/03)

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

CERTIFICATE

This is to certify that this dissertation titled "**View Materialization**" submitted by **Mr. Aloke Ghoshal**, to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, for the award of degree of **Master of Technology** in **Computer Science & Technology**, is a research work carried out by him.

T. V. Vijay Kumar
28-07-2008

Dr. T. V. Vijay Kumar
(Supervisor)

4/8/08

Prof. Parimala N.
(Dean)

ACKNOWLEDGEMENTS

This is a rare opportunity to express my heartfelt gratitude to those dearest to me. To start with are my parents, who come above all on this list. They are two lovely people, who have taught us well and instilled in us good values. Always given morals priority and in this regard, set us up high standards to achieve. Thanks mama, thanks papa.

Next up on the list is my sister, Piu. My biggest critic and my biggest supporter. This stint at JNU is to a great extent due to her motivation. It's been a real grounding experience and the learning will be there with me for ever. Two equally important people are dadu and dima. They have looked after us very well in this phase of our lives, better than any parents could have. Tuki mashi and her family has also been a big support from amongst close family.

I have been lucky to have such an approachable person like Dr. T.V. Vijay Kumar as my supervisor. For me he has been the proverbial friend, philosopher and guide who has slowly, but surely, managed to improve my approach towards work. Especially with his mantra of revise, re-revise, re-re-revise, ad-infinitum, he was able to get the best out of me in terms of the quality of our final work. Francis Bacon in his treatise "the advancement of learning (1605)", talked about the dual importance of registering and proposing doubt about the work that one is doing. Firstly to guard against potential error or slip ups, and secondly to bring about an increased knowledge about the subject, since one is having to constantly challenge and defend every little bit of their work. Vijay sir, thank you for being the 'doubt planter', and for showing me that the theory works for real! From JNU, another person due for thanks is Haider, for his untiring support close to the finish line.

Finally at the end, it is a mention of names of my good friends - Arijit, Punit, Sahil, Virmani, Gunjan, Saint and Sumiti. It has been a special association with each one of you.

Aloke Ghoshal

ABSTRACT

View materialization deals with materializing views containing data and their specification. This dissertation focuses on the view selection issue of view materialization. View selection generally deals with selecting an optimal set of beneficial views for materialization subject to constraints like space, response time, etc. The problem of view selection has been shown to be in NP. Several heuristics based view selection algorithms exist in the literature of which the most fundamental greedy based algorithm is HRU, which uses a multidimensional lattice framework to determine a good set of views to materialize. HRU exhibits a high run time complexity. This issue has been addressed by the algorithms proposed in this dissertation.

One reason for high run time complexity of HRU has been found to be the frequent re-computations of benefit values after every selection of view for materialization. An algorithm Reduced Lattice Greedy Algorithm (RLGA) has been proposed that addresses this issue by using a heuristic to reduce the number of dependencies among views. This results in a reduced lattice, with respect to the dependencies, from which beneficial views are selected greedily. RLGA was experimentally found to have fewer re-computations and an improved execution time in comparison to HRU.

Another reason for high run-time complexity of HRU is that the number of views is exponential in the number of dimensions, making it infeasible for high dimensional data sets. Algorithm Reduced Candidate Greedy Algorithm (RCGA) has been proposed that improves the scalability by recommending views using the RLGA heuristic followed by greedily selecting the most beneficial views from them. RCGA was experimentally found to have a better execution time than HRU but with a slight loss in terms of benefit values. RCGA was also compared with another existing algorithm PGA and found to have similar execution time and slightly higher benefit values for high dimensional data sets.

CONTENTS

CHAPTER 1

INTRODUCTION

A view is a specification for generating data from the underlying data repository [CD01, R97]. The view, when queried, is recomputed as per the specification given. Such a view is then said to be materialized if it contains data along with the specification of the view [R97] and the results obtained can be stored [CD01]. A materialized view can be queried as any other data within the repository.

Many views get computed within a data repository over time, but most views are discarded after being used once [R97]. The cost of generating the view is high in such a scenario and needs to be borne each time a similar query is posed. This cost becomes very high in a data warehouse type of set-up, where there is a large volume of subject-oriented, integrated, time-variant and non-volatile data[I03]. The frequent re-computation of views results in a poor performance from the system. Instead by caching results (storing as materialized views) from prior views [R97, YYL07] and by re-using these results, a substantial saving can be effected during subsequent queries over the same views. The queries posed later would be able to get results within a very short span of time [TU] from the same views that were computed and stored in earlier iterations. It is obvious that the materialized views would need to be stored to the disk. So there is an additional space overhead on the system with materialized views[CHS01, HRU96, GHRU97, R97, TU, YYL07].

The number of possible views within a system is related to the number of dimensions of the data, in fact, an exponential function of the number of dimensions[NT02]. So trying to materialize all possible views is infeasible in most real world systems. A better alternative is to choose the optimal subset of views to materialize that bring about the maximum improvement in the performance of the system. However, making such a selection is NP complete [CHS01,GHRU97, SRR06], so a better strategy or a heuristic needs to be used to choose the most beneficial set of views for a given context. This has been the focus of most of the existing literature in the area of view materialization. The related work in the area of view materialization is discussed next.

1.1 Related Work

Research in the area of materialization of views started off in the 80's [BLT86, H87, RK86, SF89, SF90, SP89a, SP89b]. The initial focus was on decoupling views from the underlying data store, also referred to as pure data in [R97], and materializing these views and storing them in separate tables (or memory) as opposed to keeping them purely virtual [RK86, SF90, SP89a]. Since data in the materialized views would need to be updated with changes to the underlying data, alternatives suggested were either to re-compute the entire view [BLT86, RK86] or to follow an update procedure that is able to work with the differentials [BLT86, H87, SP89a]. In [SP89a] an approach is given using differential files to which the modified tuples are appended followed by duplicates elimination, whenever changes are seen. Since all changes to the backing store may not be necessary for propagation to the view, a screen test approach was suggested [BLT86, SP89a] to filter out the irrelevant ones. Approach for updates to views in the case of distributed systems was presented in [SP89b, SF89, SF90]. A currency concept was introduced [SP89b] to indicate the lag between the data present in the view and that of the backing store. The currency value allows identification of the optimal time for executing the update queries for the materialized views. The updates could be run in a scheduled manner or on demand. A hybrid of the two approaches is presented in [SP89b]. This work was extended in [SF90] to be able to choose the most optimal sources for an update (another view or backing store) for a set of distributed materialized views. The one major drawback of the algorithm presented in [SF90] is that it basically works for acyclic graphs, i.e. not having duplicate views at different sites. Though the authors have handled cyclic graphs by reducing it to an acyclic graph, the computation complexity for such a pre-processing step would be phenomenal in most distributed system.

[CHS01] provides a definition for the view selection problem as the choice of views to materialize given set of workload queries, storage space and database schema. The authors go on to show a similarity between the approach for selection of views to materialize and the approach for placement and replication of data over nodes in a distributed network [CHS01]. The key contribution of [CHS01] being to show that selection of views to materialize is a function of the statistics about the size of the views available with the query optimizer. With complete statistics availability assumption, which happens to be the basis for many heuristics (like HRU96, NT02, etc.), the complexity of the optimum view selection problem is shown to be exponential in the size of dimensions of the data cube. On the other hand when heuristic are used for estimating the size, it reduces to polynomial complexity. But the cost for selection of indexes and the cost for maintenance of views have not been considered in the analysis.

[AD98] does a complexity analysis for the problem of making use of materialized views to answer user queries. The problem is analyzed in two cases, first for a closed world assuming all relevant tuples are materialized, and then for an open world assuming only a few relevant tuples are materialized. With the open world assumption, the materialized view is incomplete and only able to partly answer user queries. This tolerance for approximation (or error) in results in the open world assumption brings with a great saving in computation costs as shown in [AD98]. The authors also briefly touched upon benefits of using such an approach for self-maintenance of views to cater to updates on the underlying data. Since with the open world assumption, the view is perceived as an incomplete representation of the database, updates can automatically be compensated for with such a model. On the other hand for a closed world assumption all such computations become intractable.

[TB00] goes on to better generalize the issues in materialized views and present a number of design goals for selection of views. These include the different cost functions (query evaluation, maintenance, etc.) and the constraints (space, response time, etc.) that need to be considered. [RSS96] proposed an approach for materializing some additional views that in turn improve the query response time for other dependent views.

A survey on some of the existing literature for the period can be found in [CD01, MSRK99, SRR06]. The authors [CD01] have categorized the surveyed work based on the approach taken for view selection (e.g. space time trade-off), for view maintenance (e.g. incremental) and on applicability to distributed systems. In [MSRK99] an overview of the major research challenges in the area of data warehousing is given, one of the problems being related to view materialization. In [SRR06] the survey is arranged along the lines of pure static, pure dynamic and hybrid materialization of views. The authors [SRR06] further go on to show the benefit of taking a hybrid approach to view materialization and an approach on the hybrid lines. The authors [SRR06] performed extensive tests to compare their algorithm against another dynamic view management algorithm Dynamat [KR01].

One of the key issues in view materialization is materialized view selection. The view selection problem is the selection of the subset of views to materialize, keeping certain constraints like cost of materialization, space available to store views, etc. in focus [TB00]. This has been stated more formally in [CHS01] as the issue of selecting a set of views V over a schema R, such that the some constraint X (like storage space) is not exceeded by the set of views selected. Further, the cost estimation function for query processing, the cost of selecting the views V over R, using a given set of workload queries Q, should be the minimum [CHS01]. The materialized view selection is discussed next

1.2 Materialized View Selection

As mentioned earlier one of the key issues surrounding materialized views is the issue of selection of the appropriate views to materialize. There is usually an exponentially high number of possible views to be materialized within any system[NT02], but the space available for storing these views is restricted, so only a small number of views can be materialized at a time. This subset of views needs to be selected from among all possible views to bring about the maximum improvement in the performance of the system. However, making such a selection is NP-complete [CHS01, GHRU97, SRR06]. So the entire effort in the area of selection of views to materialize is devoted to identifying a strategy that on the one hand is effective in selecting beneficial views, while on the other is feasible for practical purposes.

Some key considerations while making the selection of views to materialize are mentioned below.

Query evaluation cost minimization – Query evaluation cost is associated with the query execution plan that is chosen for answering a particular query. In the presence of materialized views, these materialized views should get a higher priority for computing results for queries as compared to the base relations [BPT97, G97, GM99, HRU96, TB00, YKL97]. The other factor related to query evaluation cost minimization is the interdependence among views, where some auxiliary views, though not directly used for answering queries, might have many other views dependent on them [RSS96, TB00]. In such situations it may be beneficial to materialize these auxiliary views even though they are not used directly for answering queries.

Operational cost minimization – Operation cost is a function of the query evaluation and maintenance cost [BPT97, CD01, RSS96, TB00]. Query cost is related to the cost of computing results for a query from materialized views. While maintenance cost is related to the frequency of change to backing data repository and to the cost of propagating these changes to the materialized view.

Multiple Query Optimization (MQO) – MQO deals with identifying the optimal plan for executing multiple queries from the same point in time [I00, YKL97] by sharing of temporary results between the queries. Sharing of temporary results would bring about a cost saving for answering queries. In this area algorithms by [YKL97] using Multi-View Processing Plan (MVPP) are found to be very comprehensive. [YAE05] extends this work to include maintenance process optimization and incremental maintenance algorithms.

Partly Materialized Views – Recent research direction is towards materializing a portion of the view [L07, LWL06, ZLGD07]. Such empirical algorithms [SRR06] are facilitated by making use of historic trends and access patterns at relational and tuple levels.

A detailed categorization of the work done by authors in the area of selection of views to be materialized is given below.

1.2.1 Greedy Algorithm Based

As shown in [HRU96], the problem of selection of the set of views to materialize for data cubes is NP-complete. The authors thus went on to propose algorithms based on the greedy approach to select the beneficial views for materialization. The algorithm works with the assumption of knowing from before hand the size of different possible views to be materialized. For making the estimate about the size of the view, authors in [HRU96] have suggested making use of sampling and analytical techniques. Authors have also established the lower bound value for benefit for working with the greedy algorithm as (e-1)/e, where e is base of natural logarithms, i.e. 63% of the optimal algorithm. In the extension to the basic model of greedy algorithm two other cases are considered [GHRU97], where the probability of all views being queried need not be equal and a space constraint might be applied restricting, the number of views to be materialized. Authors have also looked at the space-time trade off aspect within the hypercube lattice.

[G97, GHRU97] present polynomial time greedy algorithms for selection of views to materialize AND graphs and OR graphs, and an exponential time algorithm for AND-OR graphs. However, the authors only consider the cost of materialization given a constraint on the maximum space available for materialization of views. [G97, GHRU97] originally did not consider the cost of maintenance of materialized views. This cost is included in the more recent work by [GM05].

A variant of the greedy algorithm which is able to identify the optimum set of views in polynomial time (PGA) is given in [NT02]. PGA operates in two phases – nomination and selection. In the first phase starting from the root of the data hypercube lattice, the smallest node from all its child nodes is nominated as the candidate for materialization. The process is repeated and nodes added to the candidate set till the bottom of the lattice is reached. The assumption for nomination phase being that, smaller views tend to be more beneficial per unit cost of materialization as compared to larger views. Next in the selection phase, the benefit of materializing each candidate is estimated to finally pick the view that yields most benefit in a greedy manner. The time complexity of PGA is shown to be $O(d^2k^2)$ for d levels in the lattice and k candidates which compares favourably over the algorithm in [HRU96] having complexity of $O(k2^{2d})$. Although, PGA does have an additional overhead of $O(d^2k)$ in terms of space needed to maintain metadata information about the candidate views in each iteration.

[EM07] presents another algorithm based on the greedy approach that makes use of the number of dependent relationship (utilization rate of a view) and the frequency of update to the base relations to compute the cost of materializing a view. The basis of the algorithm being that dependent relationships can utilize the same materialized view (i.e. if there's been no update), so there can be a cost saving if the optimum set of dependent relationships are identified based on execution cost or in a greedy manner. The set of dependent relationships so identified are then given as input to the algorithm in [EM07] which selects the optimum set of materialized views.

There are many other algorithms [CLF01, EM07, G97, GHRU97, GM05, HRU96, NT02, YYL07] that are based on the greedy algorithm. In [CLF01] an adapted greedy algorithm is presented and some elaborate test results to ascertain the benefit of working with the proposed cost model and for making use of the hybrid approach in data warehouse. The hybrid approach only selects those views for materialization that offer cost benefit, while the rest of the views are kept virtual and computed on the fly.

1.2.2 Index Selection Based

In [ACN00, GHRU97, GM05], the authors have considered selection of indexes alongside selection of materialized views. Indexes have been shown to be similar to materialized views, in the sense that both are additional physical structures added to improve the response time for queries. Both the structures present a space overhead on the system over which they have been defined. [ACN00] further goes on to prove indexes to be a special kind of materialized view that has been projected from a single table. It has also been suggested in [ACN00] that indexes and materialized views should be able to make use of one another, though no specific strategy for doing so is mentioned. In [ACN00], all candidate views for materialization are identified along with the indexes on the base table by application of heuristics for the Microsoft SQL server 2000. In contrast to this, [GM05] (and their earlier work [GHRU97]) present a more general approach for selection of indexes, which have been defined for views, which have been materialized. These indexes are built on views and are relevant in bringing down the computation cost for queries, only if the corresponding views have been materialized. [GHRU97, GM05] present a few algorithms (e.g. r-greedy, inner level greedy) based on the greedy approach [HRU96] that consider the selection of materialized views and indexes for a given amount of storage space. The drawback of this approach is that in case more than one indexes to a particular view are present, the advantage of using one index over the other cannot be evaluated until all possible dependent views have also been materialized [CD01].

1.2.3 Genetic Algorithms Based

Application of randomized search algorithms and Genetic Algorithms (GA) to materialized view selection is also gaining focus. In this regard the pioneering work was done by [HCLK99, ZYY99]. Both papers considered the query access cost and the update cost in the cost functions presented. The work in [HCLK99] gave a high level approach for performing genetic local search to identify the set of materialized views using AND-OR graphs. In [ZYY99] the problem was defined with respect to MVPP. A fitness function to be maximized was defined, after transforming the cost function appropriately. Some experimental results from [ZYY99] also seemed to show that the GA based algorithms were able to identify better than what conventional heuristic based algorithms could do, though the GA based algorithms were taking far longer to execute. In the subsequent work by the same authors [ZYY01], the selection problem has been split into two different problems operating at two levels of hierarchy. At the higher level, the problem is selection of the optimized global processing plan by merging many individual processing plans. While at the lower level, the problem is to select the optimum set of views (nodes) to be materialized from the optimum global processing plan identified at the higher level. The authors applied evolutionary algorithms at both levels and were able to compare results from the different settings (GA applied at either level, at both level, etc.).

[HCL03] extended their earlier work to present a two part representation for the genome, first one to identify the query plan and the second one for the view (node) to be materialized from the plan. Using such a representation, the plan and the view are directly known from the resulting population of the subsequent generations. The test results show GA yields better quality results than the greedy algorithm though its execution time is high for small number of queries and shows improvement only for large number of queries. The higher execution time for GA based was also seen in [ZYY01].

[LB06] present two evolutionary algorithms for selection of views to materialize incase of multiple distributed OLAP (M-OLAP) data cubes, including a co-evolutionary genetic algorithm. For representation of the genome, eight bits per cube has been used where a value of 1 indicates that the corresponding node from the cube is to be materialized. For the normal GA, each individual is represented by one genome string obtained by merging M (number of data cubes) eight bit strings. While in the co-evolutionary algorithm, each individual is represented by M genomes each of length eight bits. In their experimental evaluation on the TPC-R database over a LAN set-up, [LB06] found that the co-evolutionary algorithm outperformed simple GA and Greedy algorithms in terms of the total cost of solutions. The co-evolutionary algorithm also scales up better than the other two for an increase in the number of nodes in the data cubes (varying it from 64 to 256) and an increase in the number of cubes.

Unlike the work where different objectives of the view selection problem (like query cost) are combined into one (linear) cost function to be minimized, in [L06] a multi-objective function is used. Two objectives for the selection problem are identified as the query cost and the maintenance cost, that are simultaneously minimized using multi-objective GA. By minimizing such a multi-objective function, a number of optimal (pareto) solutions can be obtained. From these solutions, the appropriate set of views can thus be materialized trading one objective for the other depending upon their importance. The author performed tests with two standard non-elitist multi-objective genetic algorithms and found the results compared favourably against greedy algorithm.

[VGP07] presents two hybrid (genetic and greedy) algorithms for selection of an optimal set of objects (materialized views, vertical fragments and indexes) from the data cube. In the greedy step the set of views with highest usability ratio are picked up and appended to the set of chromosomes from the previous generation randomly. Thereby modulating the solution space in which the greedy algorithm is applied. In this way the chromosome length keeps on growing until the length yielding optimal results (convergence) is found.

1.2.4 Web View Materialization

Alongside research in view materialization for databases and data warehouses, there is also ongoing work in the area of materialization of views for the web [CFP99, KSCP04, LR99, LR00, LR01a, LR01b, LR02, LR04, MMM02, SKRP01, SSA07, YFIV00]. Web presents its own set of challenges like having to work with highly dynamic content that gets updated frequently, operating in a 24X7 online mode, etc. that affect selection of views. Materialized web views enhance performance of the system on the web, by caching some of the query results outside the backing data store, which can be re-used for answering future queries [LR02].

System architecture for a web based system is given in [LR01a, LR01b, LR02]. These essentially make use of an asynchronous cache layer, between the application and db server layers, where views are materialized. Such a cache can be refreshed on update without having to be invalidated, thereby improving the response time. The response time is measured in [LR02] by the quality of service (QoS) metric, by recording the average time for servicing a request. While the freshness of data is measured using the quality of data (QoD) metric, which measures the average period from the time when an update was received to the time when the update was propagated to the relevant cached materialized views. The online view selection algorithm (OVIS) thus proposed in [LR02], works for a user defined QoD threshold value, constantly monitoring and balancing any surplus (or deficit) in QoD by decreasing (or increasing) the number of materialized views. In other words the algorithm does a QoD to QoS trade-off to ensure the system is providing results with QoD around the threshold. Experimental results in support of the OVIS algorithm is found in [LR04]. [LR04] also presents a few other metrics for measurement of QoD based on the freshness value of fragments. Depending on the requirements of the application, page fragments of the appropriate level of freshness can be aggregated to form corresponding views.

In [SKRP01] also the potent combination of parameterized fragments and XML is seen for generation of web views. Several such fragments can be pre-computed, thus saving computation costs during peak load hours. [YFIV00] evaluate different caching strategies such as caching pre-computed pages, XML file, DB caching, etc., using a declarative specification (weave) for data intensive websites.

A somewhat different treatment of web views is found in [MMM02]. Here authors have tried to find out navigational paths within structured websites, by pulling out relational abstractions from these websites. Since search based interfaces to hypertext documents are inadequate at finding the optimal navigational path between hypertext pages, so the relational abstraction is proposed on which SQL queries can be executed. A cost model, with parameter such as network access [MMM02], can then be applied on the results of the queries to identify the optimal navigational path. The approach thus allows users to work with declarative queries, which can be translated to navigational paths in the hypertext pages.

A recent work by [SSA07] uses mining technique over the web log files to identify the most relevant set of web views for a web based system. A great deal of indicative information about the user's access and navigational patterns within the system get captured in these web log files. This information can be useful for empirically [SRR06] deciding upon the best set of web views to materialize. The authors [SSA07] work with a similar intermediary cache based architectural system presented in [LR02].

1.3 Aim

Among the algorithms for view selection discussed above, the greedy based view selection algorithm presented in [HRU96] forms the foundation of many of the other algorithms for view selection. The [HRU96] algorithm works with a set of views and the dependencies among them, and from them the most beneficial view in each of the iteration is selected for materialization. The algorithm exhibits high run time complexity[NT02, SR06], which may be attributed to following reasons:

- Frequent re-computation of benefit values while selecting views for materialization i.e. with every selection of view, the benefit value of other views in the lattice may get affected and therefore require re-computation. The number of these re-computations may be high and thus contribute to the high run time complexity. This high run time complexity can be reduced by reducing the number of re-computations of benefit values. An algorithm Reduced Lattice Greedy Algorithm (RLGA) is proposed to reduce the re-computation of benefit values based on a heuristic that defines the criteria for reducing the set of dependencies among views. The reduced set of dependencies results in a reduced lattice, which when used for greedily selecting views may result in a significantly lesser number of re-computations than those required for a complete lattice. This may in turn improve run time complexity.

- The number of possible views is exponential with respect to the number of dimensions[NT02]. The algorithm is infeasible for high dimensional data sets as its execution time becomes phenomenally high. An algorithm Reduced Candidate Greedy Algorithm (RCGA) is presented that improves the scalability, with respect to number of dimensions, for selecting views for materialization. The algorithm RCGA, utilizes the heuristic used in RLGA to identify view for materialization in polynomial time relative to the number of dimensions.

1.4 Organization of the Dissertation

The dissertation is organized as follows: Chapter 2 discusses the algorithms in [HRU96] and [NT02] in detail, followed by a discussion about the two proposed algorithms RLGA and RCGA. The comparisons of the proposed algorithms with the existing algorithms are also given in Chapter 2. The Conclusions are given in Chapter 3.

CHAPTER 2

MATERIALIZED VIEW SELECTION

Materialized view selection has been identified as a key issue in view materialization. A view selection problem generally deals with selecting an optimal set of views for materialization subject to constraints like space, response time, etc. [TB00]. An optimal selection of views usually refers to the set of views that are most beneficial in answering user queries and may thus reduce the operational costs [TB00].

As per [YYL07], there are three possible ways of selecting views to materialize namely materialize all, materialize none and materialize a select few. Each of these ways trades in different amounts the space requirement for storing views, against the response time for answering user queries. Although materializing all views (2^d views, for d dimensions) may result in the least possible response time, storing these views would have a high space overhead. At the other extreme with no views materialized, the view's results would have to be recomputed each time slowing down the response time. Thus, the only option available is to selectively materialize a subset of views statically that are likely to be most beneficial in answering user queries, while keeping the remaining views dynamic [SRR06, YYL07].

However, due to the exponentially high number of possible views existing in a system, the search space is very large, so the problem of selecting the optimal subset from it for materialization has been shown to be in NP [CHS01, GHRU97, SRR06]. Thus, for all practical purposes, the selection of the subset of views to materialize is done by pruning of the search space, in either of the two ways mentioned below [TU].

1. *Empirically*: By analyzing past user querying patterns. Using the log of queries posed in the past, information like volume of data, commonly accessed portions of the data, etc. can be computed. This information can guide the selection of appropriate views to materialize that are likely to be useful for answering future queries.
2. *Heuristically*: By identifying the best possible subset of views using a heuristic based algorithm. The heuristics commonly used are Greedy based [CLF01, EM07, G97, GHRU97, GM05, HRU96, NT02, YYL07], A*[GM05, GYCL03], Genetic based [HCLK99, L06, VGP07], etc. These heuristics provide a reasonably good solution for selecting views to materialize for many real world problems.

Several heuristics based approaches have been discussed in the literature[ACN00, CLF01, EM07, G97, GHRU97, GM05, HRU96, HCLK99, HLJ06, KSCP04, L06, LB06, LR00, LYSW07, MMM02, NT02, SKRP01, SSA07, YYL07, ZLGD07, ZYY99, ZYY01]. Most of these are greedy based approaches. The greedy based approach at each step selects an additional view, that has the maximum benefit per unit space and that fits within the space available for view materialization. Several papers discuss the greedy heuristic for selecting views for materialization [CLF01, EM07, G97, GHRU97, GM05, HRU96, NT02, YYL07]. One of the fundamental greedy based algorithms was proposed in [HRU96]. This algorithm uses a multidimensional lattice framework, expressing dependencies among views, to determine a good set of views to materialize.

The algorithm in [HRU96] exhibits a high run time complexity. This high run time complexity may be due to frequent re-computation of benefit values while selecting views for materialization i.e. with every selection of view, the benefit values of other views in the lattice may get affected and require re-computation of their benefit values. The number of re-computations may be high in each iteration and may therefore contribute to the high run time complexity. This high run time complexity can be reduced by reducing the number of re-computation of benefit values. An algorithm Reduced Lattice Greedy Algorithm (RLGA) is proposed that reduces the re-computation of benefit values of the views by considering only those dependencies in the lattice that have an impact on the benefit values. The resultant lattice is a reduced lattice with respect to the dependencies among views in it. The views are selected greedily over the reduced lattice.

Another prime reason for high run time complexity for algorithm in [HRU96] is that the number of possible views is exponential with respect to the number of dimensions[NT02]. This makes the algorithm infeasible for high dimensional data sets as its execution time becomes phenomenally high. An algorithm Reduced Candidate Greedy Algorithm (RCGA) is presented that improves the scalability, with respect to number of dimensions, for selecting views for materialization. The algorithm RCGA attempts to achieve solution in polynomial time relative to the number of dimensions.

Before discussing the algorithms RLGA and RCGA, a brief overview of greedy based algorithms is given with emphasis on algorithms in [HRU96] and in [NT02].

2.1 Greedy Based Algorithms

As discussed earlier, greedy based algorithms are widely used algorithms for selecting views to materialize. These algorithms are based on selecting views in decreasing order of their benefit per unit space values subject to space being available for their materialization. Most greedy based algorithms work with the multidimensional lattice framework[SRR96]. One of the fundamental greedy based algorithms is the one proposed in [HRU96] that uses the dependencies among views in the lattice framework to select the best set of views to materialize. The HRU algorithm exhibits high run time complexity due to a large number of views to analyze for selecting the beneficial views[NT02]. The HRU algorithm is discussed in section 2.1.1. Another reason for high run time complexity is that the number of possible views is exponential with respect to the number of dimensions[NT02]. A scalable solution to this problem was presented as Polynomial Greedy Algorithm (PGA) in [NT02]. The PGA algorithm attempts to achieve solution in polynomial time relative to the number of dimensions. The PGA algorithm is discussed in section 2.1.2.

Since these algorithms use the Lattice framework, the Lattice framework is discussed next.

Lattice Framework

The Lattice framework [HRU96, SRR06, MSRK99] represents all possible views to be materialized as its nodes. The top most node of the lattice, i.e. root node, represents the base fact table computed from an aggregation on all dimensions. For construction of a lattice, at least a partial ordering must exist for the views of the lattice where all views of the lattice depend upon the root node of the lattice, directly or indirectly.

A node (view) X is said to be dependent on another node (view) Y, if queries on X can be answered using the view corresponding to node Y. Direct dependencies within nodes (views) get captured within the lattice by defining an edge between the corresponding nodes, i.e. X→Y. While indirect dependencies get captured transitively, i.e. if X→Y and Y→Z, then it implies that X→Z. A node in a lattice is referred to as the ancestor node of all nodes that appear at a level lower than it in the lattice and are dependent on it (directly or indirectly). The lattice concept can be better understood from the example given below.

Example 1: Consider a lattice constructed for a data set comprising three dimensions A, B and C. There will thus be a total of 2^3, i.e. 8 possible nodes (views) in the lattice. The corresponding lattice is shown below in Figure 1.

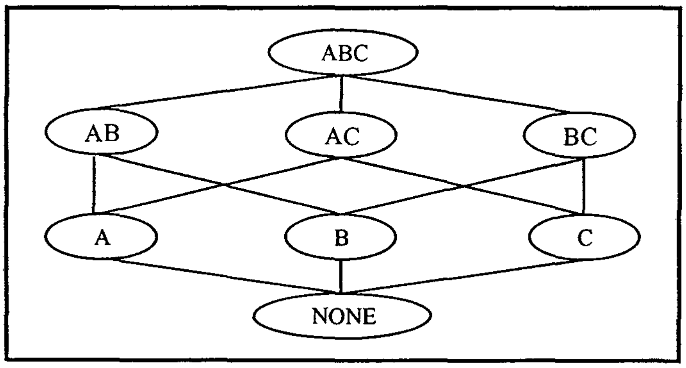


Figure 1: Lattice Structure for a 3-Dimensional Data

The Lattice, shown in Figure 1, shows that all views directly or indirectly depend on the top view, i.e. *ABC* and view *NONE*, which has no dimension associated with it, has no dependent view. All nodes have edges connecting them to their direct ancestors. Like view AB, which is the ancestor of views A and B, has edges to these dependent views. Further, using the lattice, cost and benefit values can be identified for the views corresponding to each node of the lattice.

2.1.1 The HRU Algorithm

The [HRU96] algorithm works with a linear cost model, where cost is computed in terms of the size of the view. The basis for this cost model is that there is an almost linear relationship between the size of the view on which a particular query is executed and the execution time of the query, as all tuples of the view may be needed to answer a query. This assumption of the linear cost model has been experimentally justified in [HRU96].

The algorithm in [HRU96] requires the size values for each view (node) of the lattice to be available prior to use of the algorithm. Since the size values are synonymous with cost, the accuracy of the algorithm depends on the correctness of these size values. However, given that the actual number of tuples change over time, it is difficult to determine the size at any given instant of time. The size of views can be estimated using sampling or analytical techniques [HRU96]. The algorithm based on [HRU96] is given in Figure 2 and the method for construction of the complete lattice from dependencies in Figure 3. An example showing the construction of reduced lattice using this method is shown after the Figure 3.

```
Input:   Dependencies D among views V and size of the views
Output:  Top T views
Method:
Step 1: //Arrive at a complete lattice L
        L=ConstructLattice(D)
Step 2: // Select Top T views
        Count :=0; S:={Root View};
        B(V,S) =0, initially;
        1.  FOR (Count :=1 to  T)
                a.  ComputeBenefit()
                b.  Select top view ($V_{TOP}$) from not in S such that B($V_{TOP}$, S) is maximized.
                c.  S := S UNION {$V_{TOP}$}
                d.  Count ++
        2.  RETURN S
        // Method to compute benefit value of views of the lattice
        METHOD ComputeBenefit()
        For every view V from the lattice L, compute:
                B(V,S) := Benefit of the view V, relative to the nearest materialized ancestor
                view in the set S
```

Figure 2: Algorithm based on HRU

```
Input:  Dependencies D among views V and size of the views
Output: Complete Lattice L
Method:
    L:= $Node_{ROOT}$ corresponding to Root View
    FOR (Every Dependency $d_{IJ}$ from D between two views $V_I$, $V_J$ from V, such that $V_I \rightarrow V_J$)
    1. Get nodes $Node_I$ and $Node_J$ corresponding to views $V_I$ and $V_J$
    2. Add Edge between $Node_I$ and $Node_J$ in case there is no path (set of edges) connecting
       $Node_I$ to $Node_J$
    3. IF ($Node_J$ is not the same as $Node_{ROOT}$)
           a. Add Edge between $Node_J$ and $Node_{ROOT}$ in case there is no path (set of edges)
              connecting $Node_J$ to $Node_{ROOT}$
           b. Remove Edge between $Node_I$ and $Node_{ROOT}$ in case a direct edge exists
              between $Node_I$ and $Node_{ROOT}$
    Return L
```

Figure 3: Method ConstructLattice

Example 2: For the following dependencies D among Views V of Sizes S, arrive at a complete lattice

V	S
ABC	5K
AB	5K
AC	0.9K
BC	3K
A	0.09K
B	1K
C	0.5K

D = {AB→ ABC, AC→ ABC, BC → ABC, A→ ABC, A→AB, A→AC,

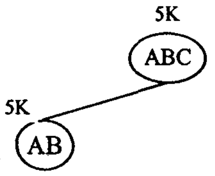
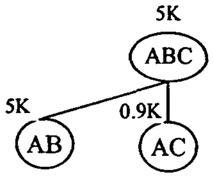
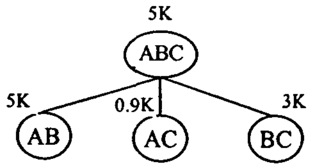
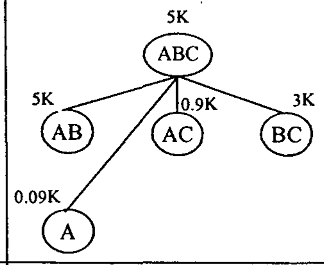
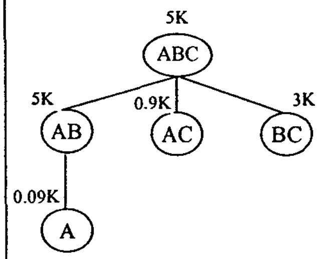
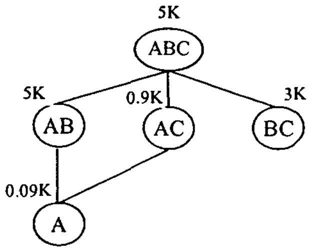
B→ ABC, B → AB, B →BC, C → ABC, C → AC, C → BC}

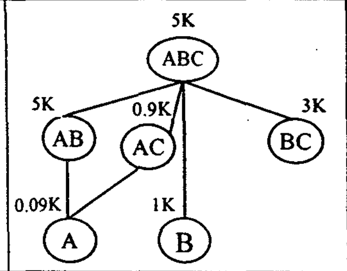
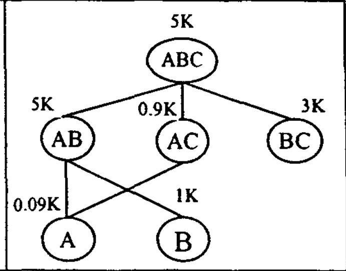
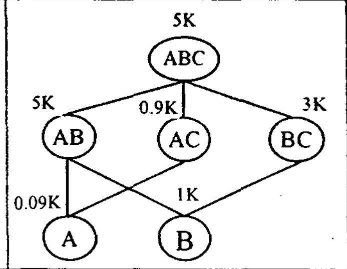
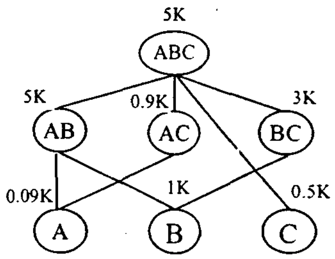
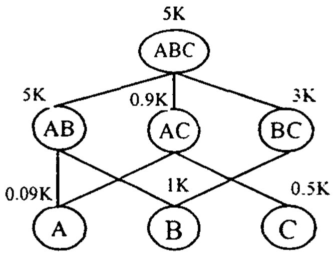
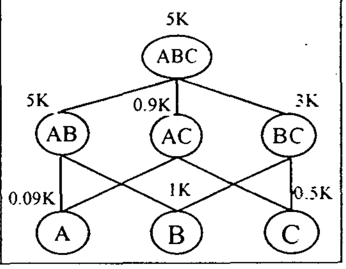
First, a node is created for the root view ABC, and a reference to the lattice L is assigned this node.

$Node_{ABC:}$ = Create new node for view ABC

L: = $Node_{ABC}$

The step wise construction of the complete lattice from given dependencies among views is shown below

I	Dependency	Execution Steps	Lattice
First Iteration	AB→ ABC	1. Get nodes for views ABC and AB. Node$_{ABC}$ for view ABC exists. Node$_{AB}$ = Create new node for view AB 2. Add an edge between Node$_{ABC}$ and Node$_{AB}$, since it does not exist 3. N.A. 4. NA	
Second	AC → ABC	1. Get nodes for views ABC and AC. Node$_{ABC}$ for view ABC exists. Node$_{AC}$ = Create new node for view AC 2. Add an edge between Node$_{ABC}$ and Node$_{AC}$, since it does not exist 3. N.A. 4. N.A.	
Third Iteration	BC → ABC	1. Get nodes for views ABC and BC. Node$_{ABC}$ for view ABC exists. Node$_{BC}$ = Create new node for view BC 2. Add an edge between Node$_{ABC}$ and Node$_{BC}$, since it does not exist 3. N.A. 4. N.A.	
Fourth Iteration	A → ABC	1. Get nodes for views ABC and A. Node$_{ABC}$ for view ABC exists. Node$_{A}$ = Create new node for view A 2. Add an edge between Node$_{ABC}$ and Node$_{A}$, since it does not exist 3. N.A. 4. N.A.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K A
Fifth Iteration	A → AB	1. Get nodes for views AB and A. Both nodes Node$_{AB}$ for view AB and Node$_{A}$ for view A exist. 2. Add an edge between Node$_{AB}$ and Node$_{A}$, since it does not exist. 3. Edge (path) exists between Node$_{AB}$ and root node Node$_{ABC}$, so new edge does not have to be added between them. 4. Remove edge between Node$_{A}$ and Node$_{ABC}$.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K A
Sixth Iteration	A → AC	1. Get nodes for views AC and A. Both nodes Node$_{AC}$ for view AC and Node$_{A}$ for view A exist. 2. Add an edge between Node$_{AC}$ and Node$_{A}$, since it does not exist. 3. Edge (path) exists between Node$_{AC}$ and root node Node$_{ABC}$, so new edge does not have to be added between them. 4. Remove edge between Node$_{A}$ and Node$_{ABC}$.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K A

I	Dependency	Execution Steps	Lattice
Seventh Iteration	B → ABC	1. Get nodes for views ABC and B. Node$_{ABC}$ for view ABC exists. Node$_{B}$ = Create new node for view B 2. Add an edge between Node$_{ABC}$ and Node$_{B}$, since it does not exist 3. N.A. 4. N.A.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K A B
Eighth Iteration	B → AB	1. Get nodes for views AB and B. Both nodes Node$_{AB}$ for view AB and Node$_{B}$ for view B exist. 2. Add an edge between Node$_{AB}$ and Node$_{B}$, since it does not exist. 3. Edge (path) exists between Node$_{AB}$ and root node Node$_{ABC}$, so new edge does not have to be added between them. 4. Remove edge between Node$_{B}$ and Node$_{ABC}$.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K A B
Ninth Iteration	B→ BC	1. Get nodes for views BC and B. Both nodes Node$_{BC}$ for view BC and Node$_{B}$ for view B exist. 2. Add an edge between Node$_{BC}$ and Node$_{B}$, since it does not exist. 3. Edge (path) exists between Node$_{BC}$ and root node Node$_{ABC}$, so new edge does not have to be added between them. 4. Remove edge between Node$_{B}$ and Node$_{ABC}$.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K A B
Tenth Iteration	C → ABC	1. Get nodes for views ABC and C. Node$_{ABC}$ for view ABC exists. Node$_{C}$ = Create new node for view C 2. Add an edge between Node$_{ABC}$ and Node$_{C}$, since it does not exist 3. N.A. 4. N.A.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K 0.5K A B C
Eleventh Iteration	C → AC	1. Get nodes for views AC and C. Both nodes Node$_{AC}$ for view AC and Node$_{C}$ for view C exist. 2. Add an edge between Node$_{AC}$ and Node$_{C}$, since it does not exist. 3. Edge (path) exists between Node$_{AC}$ and root node Node$_{ABC}$, so new edge does not have to be added between them. 4. Remove edge between Node$_{C}$ and Node$_{ABC}$.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K 0.5K A B C
Twelfth Iteration	C→ BC	1. Get nodes for views BC and C. Both nodes Node$_{BC}$ for view BC and Node$_{B}$ for view B exist. 2. Add an edge between Node$_{BC}$ and Node$_{C}$, since it does not exist. 3. Edge (path) exists between Node$_{BC}$ and root node Node$_{ABC}$, so new edge does not have to be added between them. 4. Remove edge between Node$_{C}$ and Node$_{ABC}$.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K 0.5K A B C

The HRU algorithm selects views greedily using the lattice corresponding to the dependencies among views. In the HRU algorithm, the benefit of a view is computed using the cost (size) associated with the view[HRU96]. The benefit value is a function of the number of dependent views and the difference in size of a view and that of its nearest materialized ancestor view. Initially the root node (view) of the lattice is assumed as materialized and all benefit values are computed using it. In each of the iteration, the most beneficial view is selected for materialization. With each view selection, the benefit values of other views in the lattice change and are recomputed with respect to the nearest materialized ancestor view. The algorithm at each of the iteration continues to select the best view for materialization till a predefined number of views have been selected.

In the algorithm given in [HRU96], at every iteration the benefit value of all the views other than the view selected for materialization are recomputed. This may result in high number of re-computations when the number of dimensions is high. This in turn may lead to high run time complexity[NT02, SRR06, URT99, VVK02]. This run time complexity therefore can be improved upon by decreasing the number of re-computations. An algorithm Reduced Lattice Greedy Algorithm (RLGA) is proposed that attempts to reduce the number of re-computations while selecting the beneficial views for materialization. The algorithm RLGA is discussed in section 2.2. Another reason for exponential runtime complexity of the HRU algorithm is that in each of the iteration of execution of the algorithm, the benefit values of nearly all views of the lattice have to be evaluated. This exponentially high runtime complexity makes the HRU algorithm infeasible for high dimensional data sets as its execution time becomes phenomenally high. A scalable solution to this problem was presented as Polynomial Greedy Algorithm (PGA) in [NT02]. The PGA algorithm is discussed next.

2.1.2 The PGA Algorithm

The PGA algorithm was proposed in [NT02] as a solution to the exponential run time complexity of the HRU algorithm. The PGA algorithm works with a two-stage process of nomination and selection. In the first stage only a small number of views, termed as the most promising set of views, belonging to the smallest size view's hierarchy starting from the top of the lattice get nominated. In the second stage, a greedy based selection is performed over only those views that were nominated. Since the nomination phase significantly reduces the number of views under consideration, to at most m views in each the iteration for m levels of the lattice, the PGA algorithm is able to perform the selection of views in quadratic time in the number of dimensions of the data. The PGA algorithm is given in Figure 4. The PGA algorithm takes dependencies among views as input and produces top T views as output.

```
Input:   Dependencies D among views V and size of the views
Output:  Top T views
Method:
Count :=0; S:={Root View};
L:=NULL; // Lattice
Step 1: //Arrive at a complete lattice L
          L=ConstructLattice(D)
          S:= {}
Step 2:  // Nominate views
     R:= $Node_{ROOT}$
     W:={} // Set of nominated views
     WHILE (More nodes to consider at a level below R)
          1.   FOR (Each Node from one level below R)
                    Identify smallest node, $r_{SMALLEST}$ that has not been nominated earlier
          2.   W:= W U { $r_{SMALLEST}$ }
          3.   R:= $r_{SMALLEST}$
Step 3: // Materialized view selection
          1.   FOR (Every view V from the candidates set W)
                    Compute Benefit using ComputeBenefit() function from HRU
          2.   Select top view ($V_{TOP}$) from W having the highest benefit
          3.   S := S UNION {$V_{TOP}$}
          4.   Count ++
          IF (Count < T)
                    Go to Step 2; // Nominate more views
          ELSE
                    RETURN S;
```

Figure 4: Algorithm based on PGA [NT02]

The PGA algorithm nominates views in each of the iteration. The nomination starts at the root view by nominating the smallest sized child view that has not yet been nominated. The nomination moves down a level with the nominated child as the parent and the smallest child of it being nominated as next nominated view. The nominations continue until the bottom of the lattice is reached. In the selection phase, the most beneficial view from the nominated views is selected for materialization. In every subsequent iteration, the nomination of views, from the top view, and selection of the beneficial view are carried similarly until top T views are selected. The PGA algorithm greedily selects views for materialization from a few sets of nominated views instead of all the views in the lattice as done by HRU algorithm. This enables PGA algorithm to select top K views for materialization in O(Kd^2) time instead of O(K2^{2d}) time taken by the HRU algorithm, for a data set having d dimensions. The experimental comparisons between PGA and HRU are given in section 2.1.3. An algorithm RCGA has been proposed on the same lines as PGA but it utilizes the heuristic used in the algorithm RLGA, for recommending views for selection. The algorithm RCGA comprises of the recommendation phase and the selection phase. The recommendation phase recommends views based on the heuristic used in RLGA. The selection phase selects greedily most beneficial recommended view. The algorithm RCGA is discussed in section 2.3.

Example 3: Consider the dependencies among views of Example 2. The corresponding lattice is shown in Figure 5. Arrive at the top 3 materialized views using algorithm HRU and PGA.

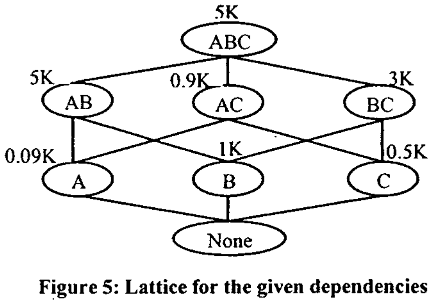


Figure 5: Lattice for the given dependencies

The selection of top 3 views using HRU and PGA algorithms are shown in Figure 6 and Figure 7 respectively.

	View (V)	Size of View (Size$_V$)	Size of the Nearest Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit (Size$_{NMA}$ - Size$_V$) × D
First Iteration	AB	5K	5K {ABC}	3 {AB, A, B}	(5-5) × 3 = **0**
	AC	0.9K	5K {ABC}	3 {AC, A, C}	(5-0.9) × 3 = **12.3**
	BC	3K	5K {ABC}	3 {BC, B, C}	(5-3) × 3 = **6**
	A	0.09K	5K {ABC}	1 {A}	(5-0.09) × 1 = **4.91**
	B	1K	5K {ABC}	1{B}	(5-1) × 1 = **4**
	C	0.5K	5K {ABC}	1 {C}	(5-0.5) × 1 = **4.5**

	View (V)	Size of View (Size$_V$)	Size of the Nearest Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit (Size$_{NMA}$ - Size$_V$) × D
Second Iteration	AB	5K	5K {ABC}	2 {AB, B}	(5-5) × 2 = **0**
	BC	3K	5K {ABC}	2 {BC, B}	(5-3) × 2 = **4**
	A	0.09K	0.9K {AC}	1 {A}	(0.9-0.09) × 1= **0.81**
	B	1K	5K {ABC}	1 {B}	(5-1) × 1 = **4**
	C	0.5K	0.9K {AC}	1 {C}	(0.9-0.5) × 1 = **0.4**

	View (V)	Size of View (Size$_V$)	Size of the Nearest Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit (Size$_{NMA}$ - Size$_V$) × D
Third Iteration	AB	5K	5K {ABC}	1 {AB}	(5-5) × 1 = **0**
	A	0.09K	0.9K {AC}	1 {A}	(0.9-0.09) × 1 = **0.81**
	B	1K	3K {BC}	1 {B}	(3-1) × 1 = **2**
	C	0.5K	0.9K {ABC}	1 {C}	(0.9-0.5) × 1 = **0.4**

Figure 6: Selection of top 3 views using HRU algorithm

	Recommended Views (V)	Size of the View (Size$_V$)	Size of the Nearest Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit = (Size$_{MAN}$- Size$_V$) × D
First Iteration	ABC	5	5K {ABC}	7 {AB, AC, BC, AC, A, B, C}	(5-5) * 7 = **0**
	AC	0.9	5K {ABC}	3 {AC, A, C}	(5-0.9) * 3 = **12.3**
	A	0.09	5K {ABC}	1 {A}	(5-0.09) * 1 = **4.91**

	Recommended Views (V)	Size of the View (Size$_V$)	Size of the Nearest Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit = (Size$_{MAN}$- Size$_V$) × D
Second Iteration	BC	3	5K {ABC}	1 {BC, C}	(5-3) * 2 = **4**
	C	0.5	0.9K {AC}	1 {C}	(0.9 – 0.5) * 1 = **0.4**

	Recommended Views (V)	Size of the View (Size$_V$)	Size of the Nearest Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit = (Size$_{MAN}$- Size$_V$) × D
Third Iteration	AB	5	5K {ABC}	2 {AB, B}	(5-5) * 2 = **0**
	B	1	3K {BC}	1 {B}	(3 – 1) * 1 = **2**

Figure 7: Selection of top 3 views using PGA algorithm

2.1.3 Comparisons (HRU Vs. PGA)

The HRU and PGA algorithms were compared by conducting experiments on an Intel based 2.3 GHz PC having 2 GB RAM. The two algorithms were compared on parameters like the execution time and the benefit value.

First, a graph was plotted to compare HRU and PGA algorithms on execution time against the number of dimensions. The graph is shown in Figure 8.

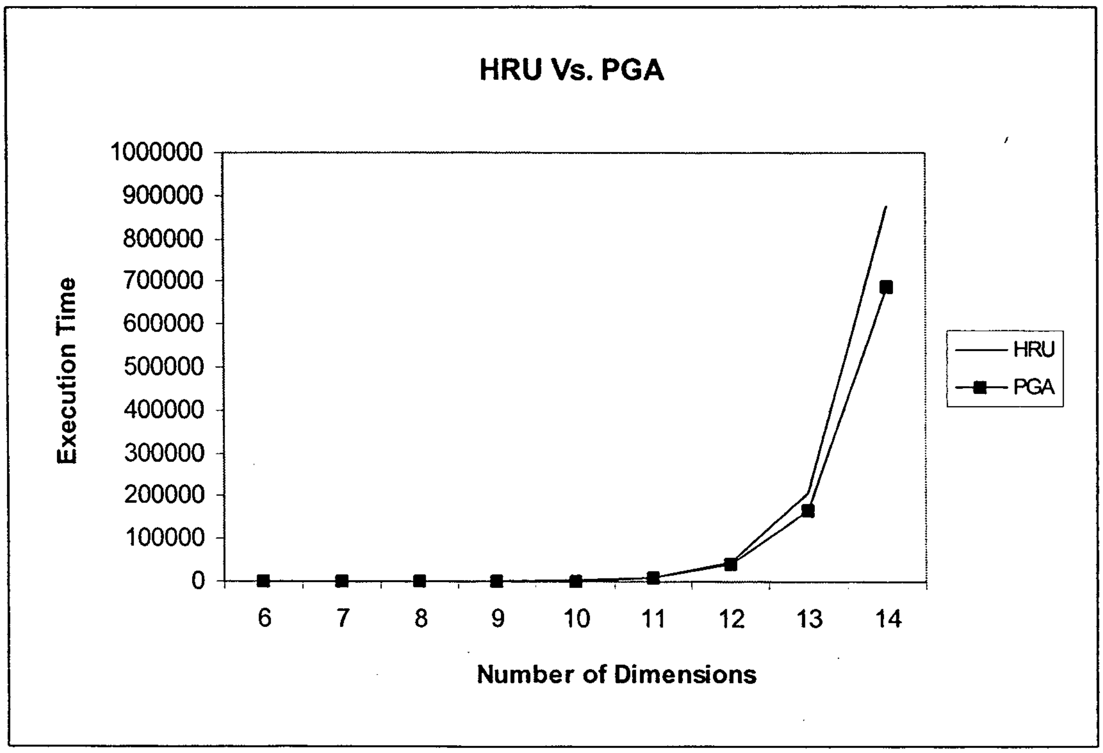


Figure 8: HRU Vs. PGA: Execution Time Vs. Number of Dimensions

It can be noted from the graph that the execution time increases with the increase in the number of dimensions. This graph establishes the claim in [NT02] that PGA has a better execution time than HRU. The same trend can be observed in the graphs, shown in Figures 9 - 12, plotted for Execution Time versus Iteration for the dimensions 8, 10, 12 and 14.

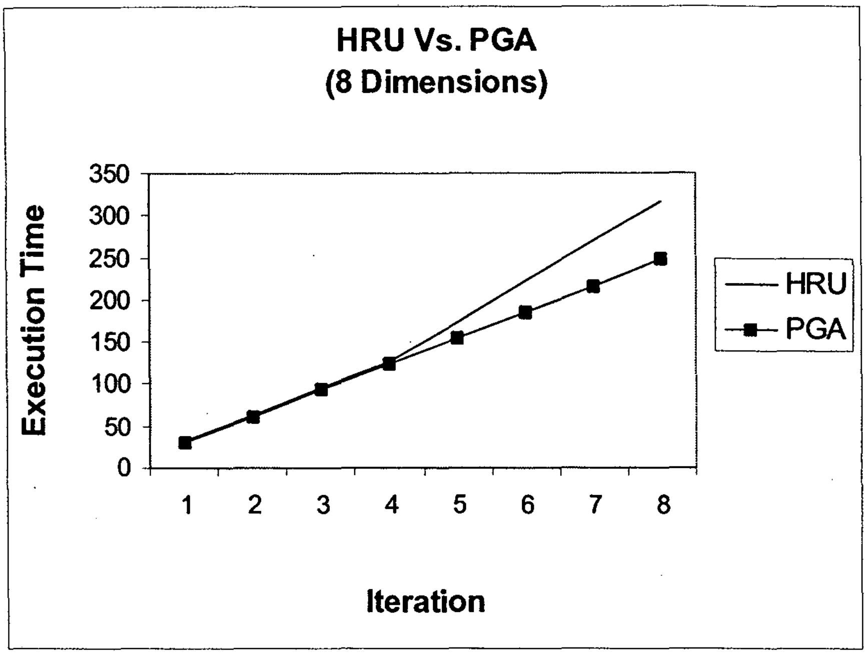


Figure 9: HRU Vs. PGA: Execution Time Vs. Iteration: 8 Dimensional Data

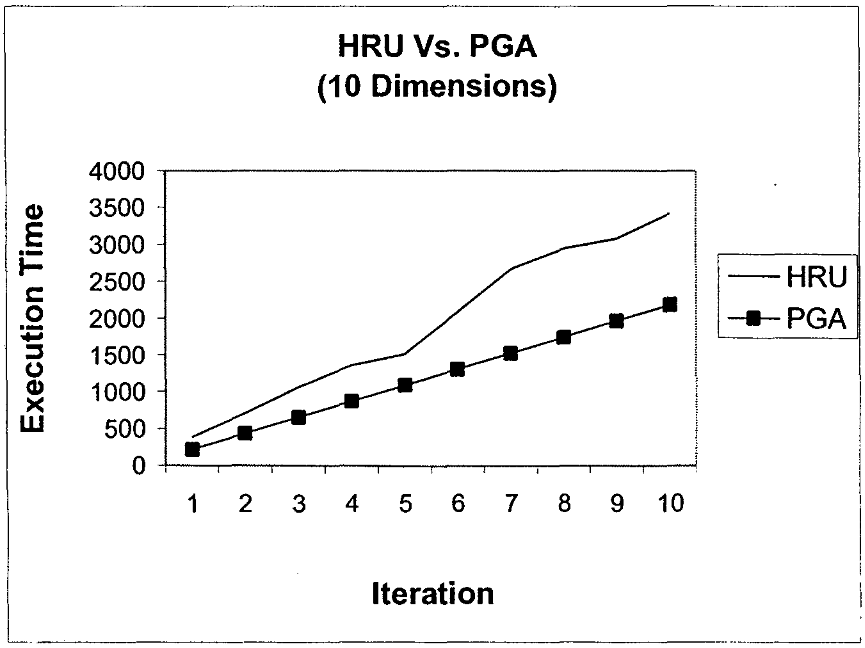


Figure 10: HRU Vs. PGA: Execution Time Vs. Iteration: 10 Dimensional Data

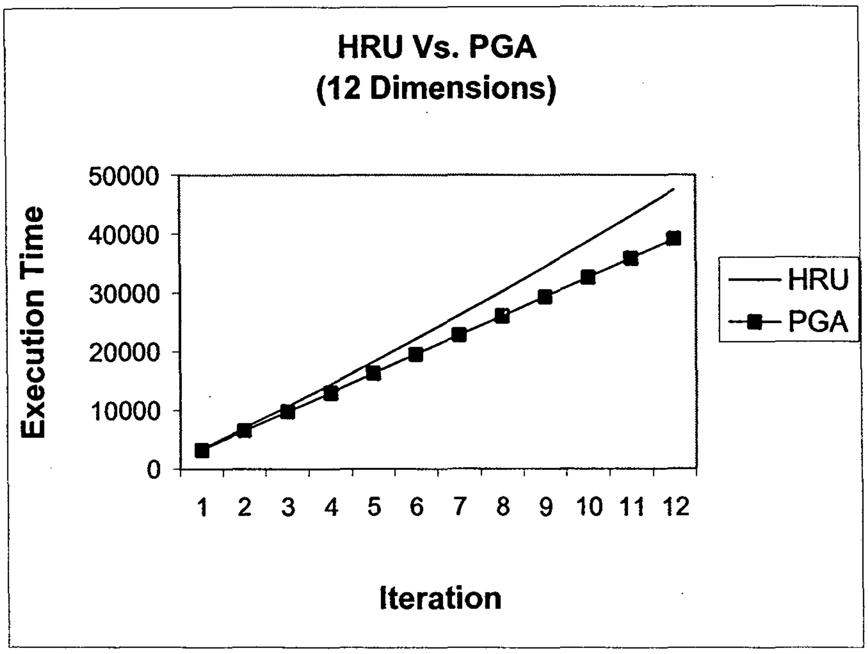


Figure 11: HRU Vs. PGA: Execution Time Vs. Iteration: 12 Dimensional Data

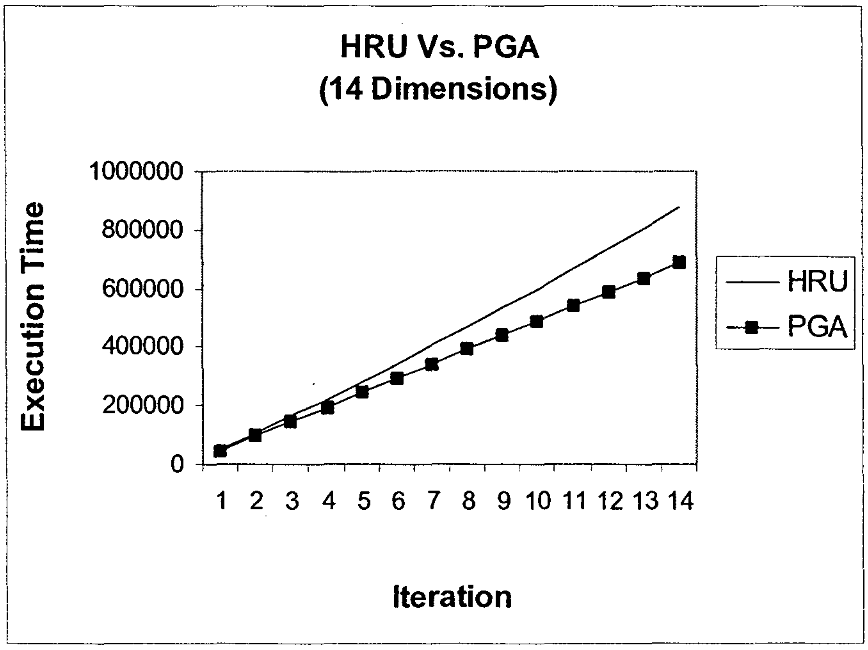


Figure 12: HRU Vs. PGA: Execution Time Vs. Iteration: 14 Dimensional Data

Next, graph was plotted to observe benefit value of the views selected versus the number of dimensions. The benefit value corresponds to at most top d views selected for materialization where d is number of dimensions. The graph is shown in Figure 13.

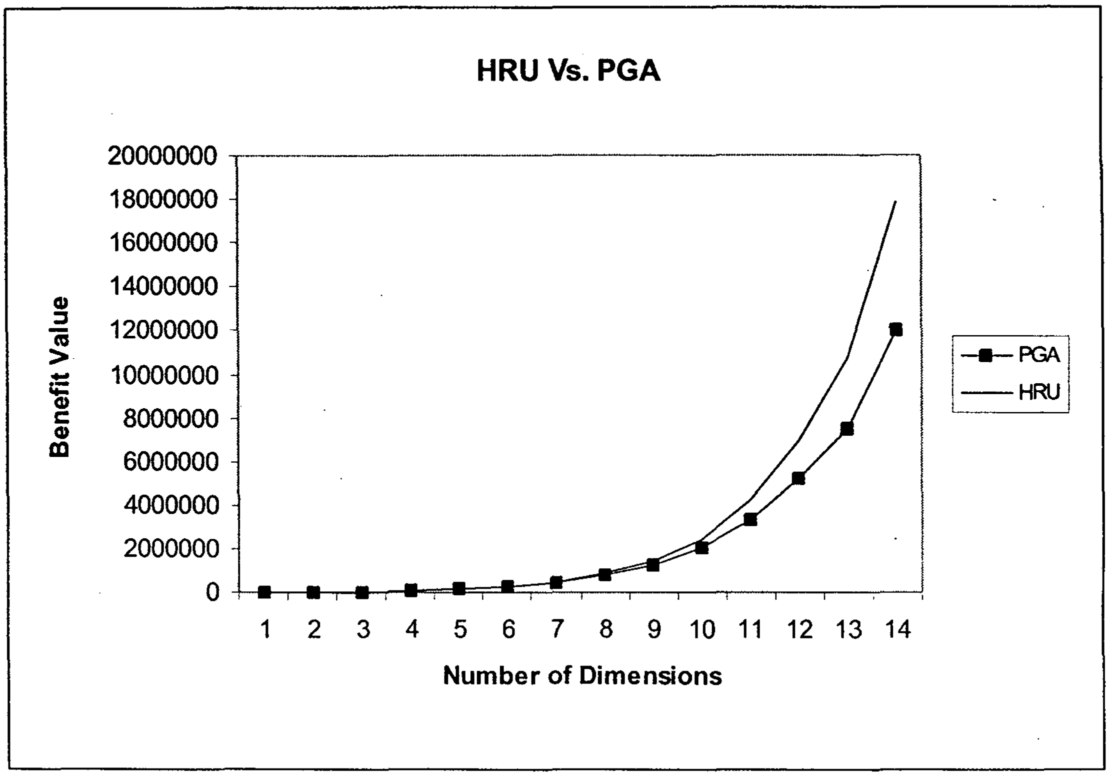


Figure 13: HRU Vs. PGA: Benefit Value Vs. Number of Dimensions

The graph shows the anticipated results as stated in [NT02] that the benefit value of PGA is lower than that of HRU. This is because the PGA algorithm is based on a heuristic that trades benefit value for an improvement in execution time.

To understand the difference in the benefit values, graphs were plotted for dimensions 8, 10, 12 and 14 against the iteration. The graphs are shown in Figures 14 - 17. These graphs also show that HRU has a benefit higher than that of PGA.

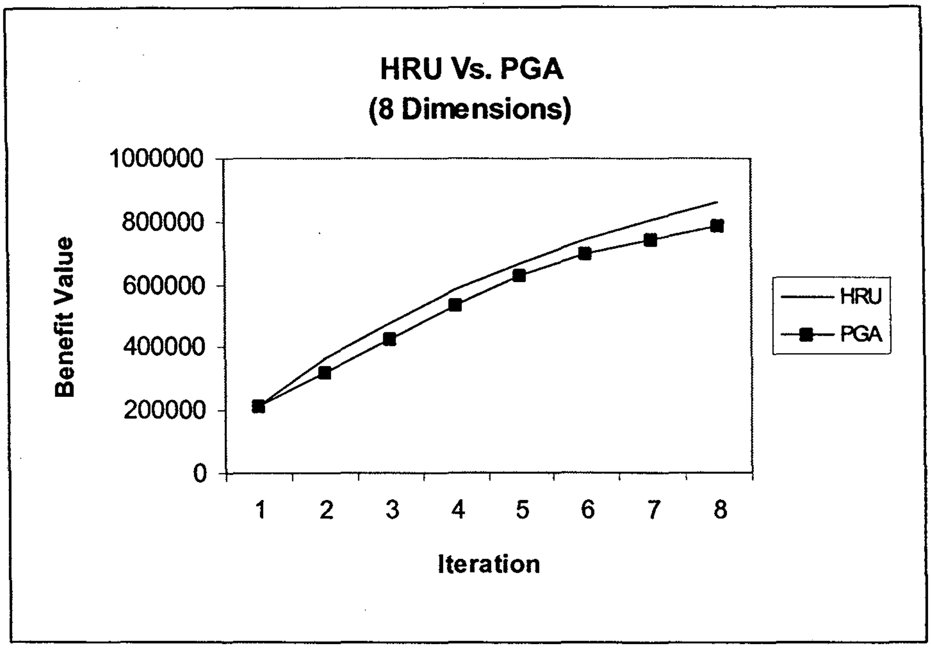


Figure 14: HRU Vs. PGA: Benefit Value Vs. Iteration: 8 Dimensional Data

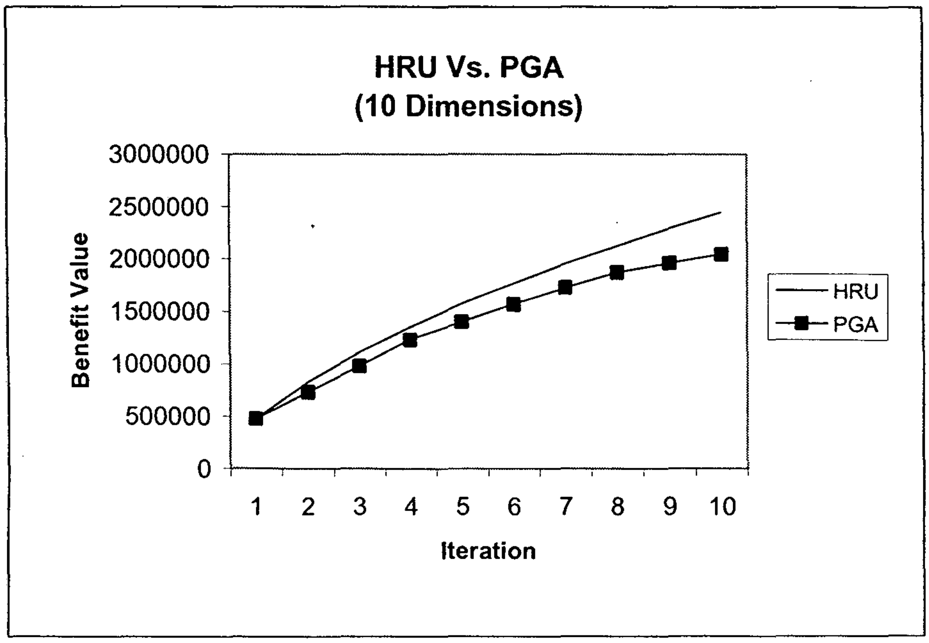


Figure 15: HRU Vs. PGA: Benefit Value Vs. Iteration: 10 Dimensional Data

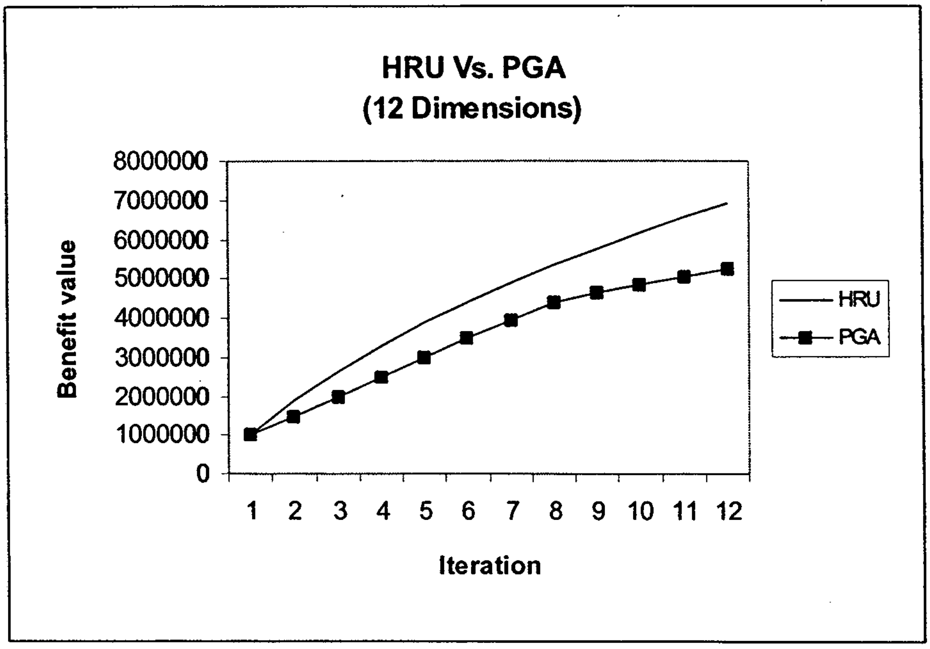


Figure 16: HRU Vs. PGA: Benefit Value Vs. Iteration: 12 Dimensional Data

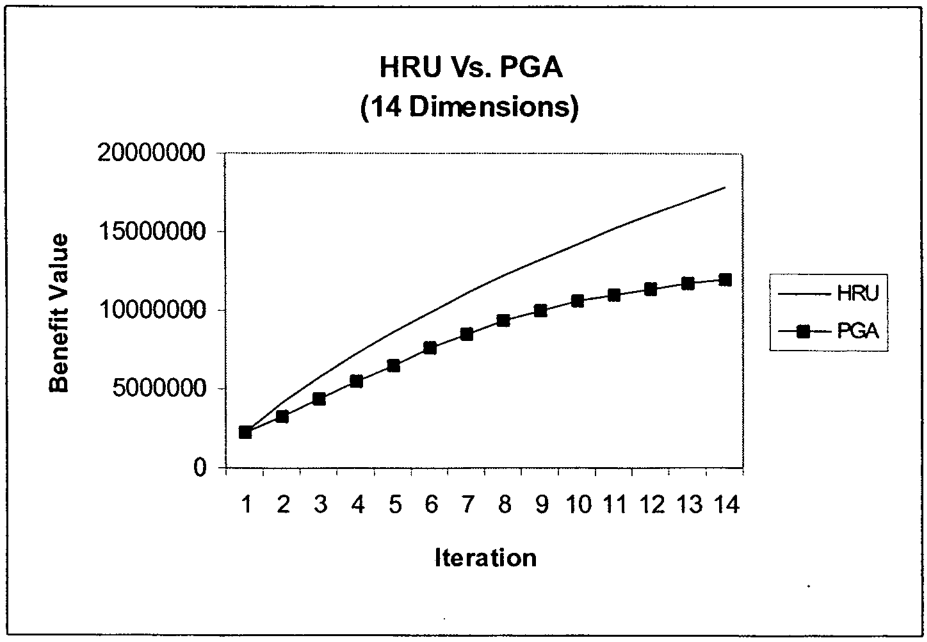


Figure 17: HRU Vs. PGA: Benefit Value Vs. Iteration: 14 Dimensional Data

2.2 Reduced Lattice Greedy Algorithm (RLGA)

The HRU algorithm exhibits high run time complexity and one reason for high run time complexity may be due to frequent re-computation of benefit values while selecting views for materialization. This high run time complexity can be improved upon by decreasing the number of re-computations of the benefit value. RLGA is proposed in this dissertation that attempts to reduce these re-computations by reducing the dependencies among the views. RLGA is based on a reduced lattice framework where instead of considering the complete lattice, as in HRU algorithm, a reduced lattice is arrived at from the set of dependencies among the views. This reduced lattice shows only those dependencies that can contribute to selecting the beneficial views. A heuristic is coined to identify such dependencies. The heuristic is defined as follows:

For each view, its dependency with its smallest sized parent is retained

The basis of this heuristic is that in HRU algorithm the view having the smallest size, among all views at a level of the lattice, has the maximum benefit at that level with respect to all the views on which it is dependent. This implies that picking the smallest sized view can maximize the benefit with respect to views above it in a lattice. In other words, retaining dependency of each view to its smallest sized parent view is most beneficial as the smallest sized parent will in turn have maximum benefit with respect to the view above it. Thus these smallest sized parent views have a high likelihood of being selected for materialization.

To understand the basis of the heuristic for the reduced lattice better, consider the lattice shown in Figure 18 (a). Now as per the heuristic the dependent view V_3 retains its direct dependency only with its smallest parent view, i.e. the view V_1, resulting in the reduced lattice shown in Figure 18 (b).

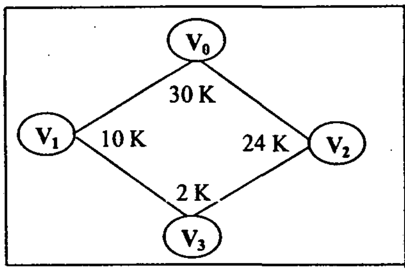


Figure 18 (a): Complete Lattice

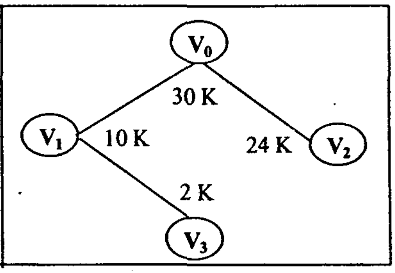


Figure 18 (b): Reduced Lattice

Again as per the heuristic, the view V_1, by virtue of being smaller in size, has a higher likelihood of being materialized as compared, to the other view V_2 from the same level of the lattice. This assumption behind the heuristic is proven to be right, as the first view to be selected for materialization from this lattice is indeed the view V_1, having the highest benefit. As a result of this for the next iteration, its dependent view V_3 will consider this view V_1, as its nearest materialized ancestor view. Thus, by adjusting dependencies as per the reduced lattice heuristic the same decision can be made, in terms of selecting the most beneficial views for materialization, without having to evaluate all the other dependencies from the lattice, like the dependency between views V_3 and its other parent view V_2.

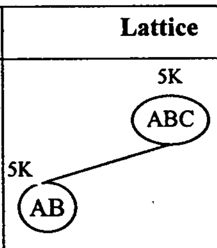
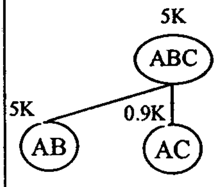
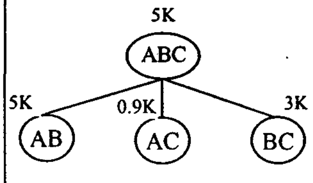
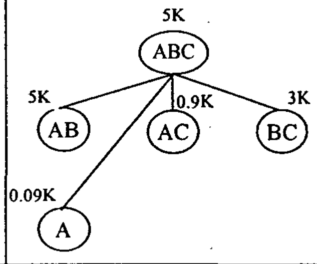
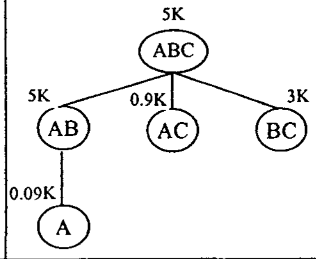
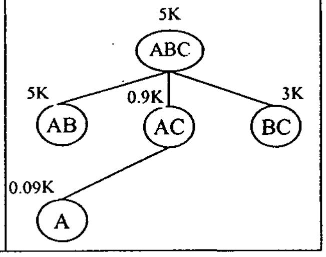
The reduced lattice comprises nodes, corresponding to views to be materialized, and edges, expressing dependencies, from each node to its smallest sized parent node. The method to arrive at reduced lattice from the set of dependencies among views is given in Figure 19, followed by a worked out example based on it.

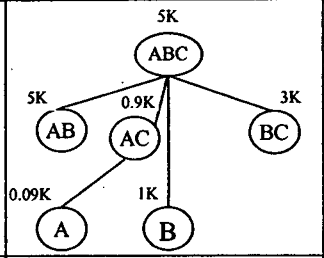
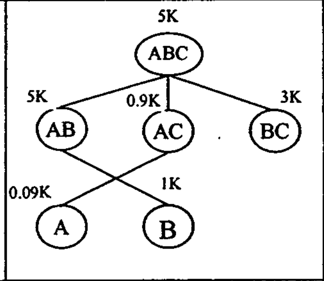
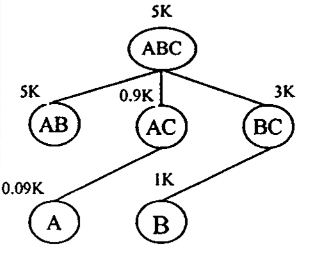
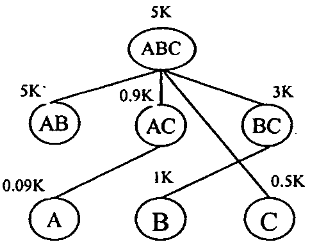
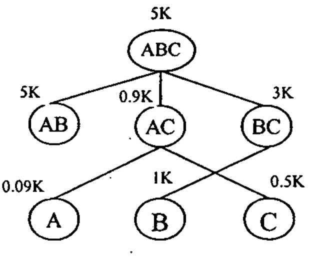
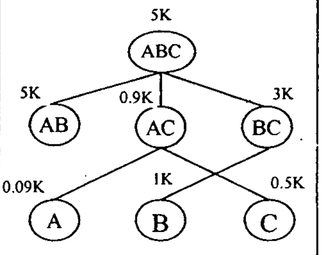
```
Input:   Dependencies D among views V and size of the views
Output:  Reduced Lattice L
Method:
    L:= $Node_{ROOT}$ corresponding to Root View
    FOR (Every Dependency $d_{IJ}$ from D between two views $V_I$, $V_J$ from V, such that $V_I \rightarrow V_J$)
    1. Get nodes $Node_I$ and NodeJ corresponding to views $V_I$ and $V_J$
    2. IF( SIZE($Node_J$) < SIZE($Node_I$.ParentNode) OR $Node_I$.ParentNode = NULL)
        a. Remove Edge between $Node_I$ and $Node_I$.ParentNode
        b. Set $Node_I$.ParentNode := $Node_J$
        c. Add Edge between $Node_I$ and $Node_J$ in case there is no path (set of edges) connecting $Node_I$ to $Node_J$
        d. IF($Node_J$ is not the same as $Node_{ROOT}$)
            Add Edge between $Node_J$ and $Node_{ROOT}$ in case there is no path (set of edges) connecting them
    Return L
```

Figure 19: Method ConstructReducedLattice

Example 4: Arrive at a reduced lattice for the dependencies among views of Example 2.

The stepwise construction of the reduced lattice is given below

I	Dependency	Execution Steps	Lattice
First Iteration	AB→ ABC	1. Get nodes for views ABC and AB. Node$_{ABC}$ for view ABC exists. Node$_{AB}$ = Create new node for view AB 2. Node$_{AB}$ has so far not been assigned a ParentNode. So, a. N.A. b. Set Node$_{AB}$.ParentNode := Node$_{ABC}$ c. Add an edge between Node$_{AB}$ and root node Node$_{ABC}$ d. N.A.	
Second Iteration	AC → ABC	1. Get nodes for views ABC and AC. Node$_{ABC}$ for view ABC exists. Node$_{AC}$ = Create new node for view AC 2. Node$_{AC}$ has so far not been assigned a ParentNode. So, a. N.A. b. Set Node$_{AC}$.ParentNode := Node$_{ABC}$ c. Add an edge between Node$_{AC}$ and root node Node$_{ABC}$ d. N.A.	
Third Iteration	BC → ABC	1. Get nodes for views ABC and BC. Node$_{ABC}$ for view ABC exists. Node$_{BC}$ = Create new node for view BC 2. Node$_{BC}$ has so far not been assigned a ParentNode. So, a. N.A. b. Set Node$_{BC}$.ParentNode := Node$_{ABC}$ c. Add an edge between Node$_{BC}$ and root node Node$_{ABC}$ d. N.A.	
Fourth Iteration	A → ABC	1. Get nodes for views ABC and A. Node$_{ABC}$ for view ABC exists. Node$_{A}$ = Create new node for view A 2. Node$_{A}$ has so far not been assigned a ParentNode. So, a. N.A. b. Set Node$_{A}$.ParentNode := Node$_{ABC}$ c. Add an edge between Node$_{A}$ and root node Node$_{ABC}$ d. N.A.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K A
Fifth Iteration	A → AB	1. Get nodes for views AB and A. Both nodes, Node$_{AB}$ for view AB and Node$_{A}$ for view A exist. 2. SIZE (Node$_{A}$.ParentNode, i.e. Node$_{ABC}$) > SIZE (Node$_{AB}$). So, a. Remove edge between Node$_{A}$ and Node$_{ABC}$. b. Set Node$_{A}$.ParentNode := Node$_{AB}$ c. Add edge between Node$_{A}$ and its parent node Node$_{AB}$. d. Edge (path) exists between nodes Node$_{AB}$ and root node Node$_{ABC}$ so new edge does not have to be added between them.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K A
Sixth Iteration	A → AC	1. Get nodes for views AB and A. Both nodes, Node$_{AC}$ for view AC and Node$_{A}$ for view A exist. 2. SIZE (Node$_{A}$.ParentNode, i.e. Node$_{AB}$) > SIZE (Node$_{AC}$). So, a. Remove edge between Node$_{A}$ and Node$_{AB}$. b. Set Node$_{A}$.ParentNode := Node$_{AC}$ c. Add edge between Node$_{A}$ and its parent node Node$_{AC}$. d. Edge (path) exists between nodes Node$_{AC}$ and root node Node$_{ABC}$ so new edge does not have to be added between them.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K A

I	Dependency	Execution Steps	Lattice
Seventh Iteration	B → ABC	1. Get nodes for views ABC and B. Node$_{ABC}$ for view ABC exists. Node$_{B}$ = Create new node for view B 2. Node$_{B}$ has so far not been assigned a ParentNode. So, a. N.A. b. Set Node$_{B}$.ParentNode := Node$_{ABC}$ c. Add edge between Node$_{B}$ and its parent node Node$_{ABC}$. d. N.A.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K A B
Eighth Iteration	B → AB	1. Get nodes for views AB and B. Both nodes, Node$_{AB}$ for view AB and Node$_{B}$ for view B exist. 2. SIZE (Node$_{B}$.ParentNode, i.e. Node$_{ABC}$) > SIZE (Node$_{AB}$). So, a. Remove edge between Node$_{B}$ and Node$_{ABC}$. b. Set Node$_{B}$.ParentNode := Node$_{AB}$ c. Add edge between Node$_{B}$ and its parent node Node$_{AB}$. d. Edge (path) exists between nodes Node$_{AB}$ and root node Node$_{ABC}$ so new edge does not have to be added between them.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K A B
Ninth Iteration	B→ BC	1. Get nodes for views BC and B. Both nodes, Node$_{BC}$ for view BC and Node$_{B}$ for view B exist. 2. SIZE (Node$_{B}$.ParentNode, i.e. Node$_{AB}$) > SIZE (Node$_{BC}$). So, a. Remove edge between Node$_{B}$ and Node$_{AB}$. b. Set Node$_{B}$.ParentNode := Node$_{BC}$ c. Add edge between Node$_{B}$ and its parent node Node$_{BC}$. d. Edge (path) exists between nodes Node$_{BC}$ and root node Node$_{ABC}$ so new edge does not have to be added between them.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K A B
Tenth Iteration	C → ABC	1. Get nodes for views ABC and C. Node$_{ABC}$ for view ABC exists. Node$_{C}$ = Create new node for view C 2. Node$_{C}$ has so far not been assigned a ParentNode. So, a. N.A. b. Set Node$_{C}$.ParentNode := Node$_{ABC}$ c. Add edge between Node$_{C}$ and its parent node Node$_{ABC}$. d. N.A.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K 0.5K A B C
Eleventh Iteration	C → AC	1. Get nodes for views AC and C. Both nodes, Node$_{AC}$ for view AC and Node$_{C}$ for view C exist. 2. SIZE (Node$_{C}$.ParentNode, i.e. Node$_{ABC}$) > SIZE (Node$_{AC}$). So, a. Remove edge between Node$_{C}$ and Node$_{ABC}$. b. Set Node$_{C}$.ParentNode := Node$_{AC}$ c. Add edge between Node$_{C}$ and its parent node Node$_{AC}$. d. Edge (path) exists between nodes Node$_{AC}$ and root node Node$_{ABC}$ so new edge does not have to be added between them.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K 0.5K A B C
Twelfth Iteration	C→ BC	1. Get nodes for views BC and C. Both nodes, Node$_{BC}$ for view BC and Node$_{C}$ for view C exist. 2. Since SIZE (Node$_{C}$.ParentNode, i.e. Node$_{AC}$) < SIZE (Node$_{BC}$), the steps 2a – 2c are not executed.	 5K ABC 5K 0.9K 3K AB AC BC 0.09K 1K 0.5K A B C

RLGA uses the reduced lattice to select the beneficial views greedily similar to HRU algorithm. The RLGA algorithm is given in Figure 20.

```
Input:   Dependencies D among Views V
Output: Top T views
Method:
Step 1: //Arrive at a reduced lattice L
        L=ConstructReducedLattice(D)
Step 2: //Select Top T Views
    1. Count :=0; MatViewList:=NULL;
    // Select Views for Materialization
    1. For every view N from the reduced lattice L, compute:
        a.  $NoDependents_N$ := COUNT (Number of views appearing at a lower level in the lattice
            and having $View_N$ as an ancestor)
        b.  $BenefitValView_N$ := ($Size_{ROOT}$ - $Size_N$) * $NoDependents_L$
        c.  Prepare $OrderedListBeneficialViews_L$ =
            ORDER_D ESC ($BenefitValView_N$ * $NoDependents_N$)
    2. WHILE (Count < T) DO
        a.  Select top view ($V_{TOP}$) from OrderedListBeneficialViews and add $V_{TOP}$ to MatViewsList
        b.  // Update benefit value and no of dependents
                i.  For every view $V_J$ from a higher level in the lattice than $V_{TOP}$, s.t. $V_{TOP} \rightarrow V_J$
                    Adjust $NoDependents_J$ := $NoDepedents_J$ - $NoDependents_{TOP}$
               ii.  For every view $V_M$ from a lower level in lattice than $V_{TOP}$, s.t. $V_M \rightarrow V_{TOP}$
                    Adjust $BenefitValView_M$ = ($Size_{TOP}$ - $Size_M$) * $NoDependents_M$
        c.  Remove $V_{TOP}$ from OrderedListBeneficialViews
        d.  Re-compute OrderedListBeneficialViews values
        e.  Count++
    3. Return MatViewList
```

Figure 20: Algorithm RLGA

RLGA takes a set of dependencies among views as input and gives the top T beneficial views for materialization as output. The algorithm comprises two stages:

1. Arrive at a reduce lattice from the set of dependencies among views
2. Using the reduced lattice, select top T views greedily.

In the first stage the set of dependencies given as input are evaluated one by one, to arrive at a reduced lattice. As discussed above only those dependencies are retained in the lattice that contributes to selecting beneficial views.

Next top T beneficial views are selected greedily using the reduced lattice. The benefit value of each view is computed similar to the way it is computed by the HRU algorithm i.e. by taking the product of number of its dependents with the cost (size) difference from its nearest materialized ancestor view. In the initial iteration, the benefit value for all views are computed with respect to the root view, which is assumed materialized. The view having the highest benefit value among them is then selected for materialization. The benefit values of all views that were dependent on the view selected for materialization are accordingly recomputed. However, unlike the HRU algorithm that operates over a complete lattice, the RLGA operates over a reduced lattice thereby substantially reducing the number of nodes dependent upon the view selected for materialization. This results in fewer nodes requiring re-computation of their benefit values thereby leading to reduction in the number of re-computations.

Example 5: Select the top 4 beneficial views using HRU and RLGA for the dependencies among views given in Example 2.

The complete lattice and the reduced lattice arrived at from the set of dependencies (given in Example 2) among views is shown in Figure 21.

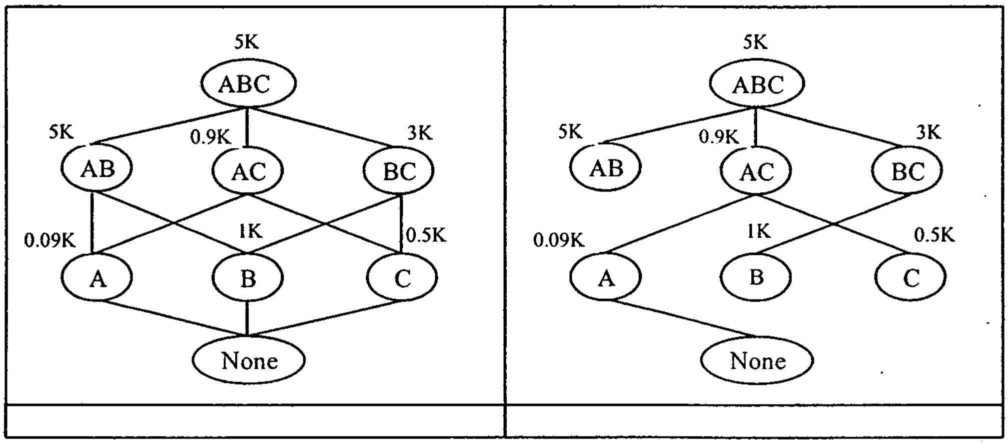


Figure 21: Complete and Reduced Lattices

The selection of top 4 views using HRU algorithm is shown in Figure 22.

	View (V)	**Size of View (Size$_V$)**	**Size of Materialized Ancestor Node (Size$_{NMA}$)**	**Number of Dependents (D)**	**Benefit (Size$_{NMA}$ - Size$_V$) × D**
First Iteration	AB	5K	5K {ABC}	3 {AB, A, B}	(5-5) × 3 = **0**
	AC	0.9K	5K {ABC}	3 {AC, A, C}	(5-0.9) × 3 = **12.3**
	BC	3K	5K {ABC}	3 {BC, B, C}	(5-3) × 3 = **6**
	A	0.09K	5K {ABC}	1 {A}	(5-0.09) × 1 = **4.91**
	B	1K	5K {ABC}	1 {B}	(5-1) × 1 = **4**
	C	0.5K	5K {ABC}	1 {C}	(5-0.5) × 1 = **4.5**

	View (V)	**Size of View (Size$_V$)**	**Size of Materialized Ancestor Node (Size$_{NMA}$)**	**Number of Dependents (D)**	**Benefit (Size$_{NMA}$ - Size$_V$) × D**
Second Iteration	AB*	5K	5K {ABC}	2 {AB, B}	(5-5) × 2 = **0**
	BC*	3K	5K {ABC}	2 {BC, B}	(5-3) × 2 = **4**
	A*	0.09K	0.9K {AC}	1 {A}	(0.9-0.09) × 1= **0.81**
	B	1K	5K {ABC}	1 {B}	(5-1) × 1 = **4**
	C*	0.5K	0.9K {AC}	1 {C}	(0.9-0.5) × 1 = **0.4**

* Recomputed

	View (V)	**Size of View (Size$_V$)**	**Size of Materialized Ancestor Node (Size$_{NMA}$)**	**Number of Dependents (D)**	**Benefit (Size$_{NMA}$ - Size$_V$) × D**
Third Iteration	AB*	5K	5K {ABC}	1 {AB}	(5-5) × 1 = **0**
	A	0.09K	0.9K {AC}	1 {A}	(0.9-0.09) × 1 = **0.81**
	B*	1K	3K {BC}	1 {B}	(3-1) × 1 = **2**
	C	0.5K	0.9K {ABC}	1 {C}	(0.9-0.5) × 1 = **0.4**

* Recomputed

	View (V)	**Size of View (Size$_V$)**	**Size of Materialized Ancestor Node (Size$_{NMA}$)**	**Number of Dependents (D)**	**Benefit (Size$_{NMA}$ - Size$_V$) × D**
Fourth Iteration	AB	5K	5K {ABC}	1 {AB}	(5-5) × 1 = **0**
	A	0.09K	0.9K {AC}	1 {A}	(0.9-0.09) × 1 = **0.81**
	C	0.5K	0.9K {AC}	1 {C}	(0.9-0.5) × 1 = **0.4**

Figure 22: Selection of top 4 views using HRU

The selection of top 4 views using RLGA algorithm is shown in Figure 23.

	View (V)	Size of View (Size$_V$)	Size of Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit (Size$_{NMA}$ - Size$_V$) × D
First Iteration	AB	5K	5K {ABC}	1 {AB}	(5-5) × 1 = **0**
	AC	0.9K	5K {ABC}	3 {AC, A, C}	(5-0.9) × 3 = **12.3**
	BC	3K	5K {ABC}	2 {BC, B}	(5-3) × 2 = **4**
	A	0.09K	5K {ABC}	1 {A}	(5-0.09) × 1 = **4.91**
	B	1K	5K {ABC}	1 {B}	(5-1) × 1 = **4**
	C	0.5K	5K {ABC}	1 {C}	(5-0.5) × 1 = **4.5**

	View (V)	Size of View (Size$_V$)	Size of Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit (Size$_{NMA}$ - Size$_V$) × D
Second Iteration	AB	5K	5K {ABC}	1 {AB}	(5-5) × 1 = **0**
	BC	3K	5K {ABC}	2 {BC, B}	(5-3) × 2 = **4**
	A*	0.09K	0.9K {AC}	1 {A}	(0.9-0.09) × 1= **0.81**
	B	1K	5K {ABC}	1 {B}	(5-1) × 1 = **4**
	C*	0.5K	0.9K {AC}	1 {C}	(0.9-0.5) × 1 = **0.4**

* Recomputed

	View (V)	Size of View (Size$_V$)	Size of Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit (Size$_{NMA}$ - Size$_V$) × D
Third Iteration	AB	5K	5K {ABC}	1 {AB}	(5-5) × 1 = **0**
	A	0.09K	0.9K {AC}	1 {A}	(0.9-0.09) × 1 = **0.81**
	B*	1K	3K {BC}	1 {B}	(3-1) × 1 = **2**
	C	0.5K	0.9K {ABC}	1 {C}	(0.9-0.5) × 1 = **0.4**

* Recomputed

	View (V)	Size of View (Size$_V$)	Size of Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit (Size$_{NMA}$ - Size$_V$) × D
Fourth Iteration	AB	5K	5K {ABC}	1 {AB}	(5-5) × 1 = **0**
	A	0.09K	0.9K {AC}	1 {A}	(0.9-0.09) × 1 = **0.81**
	C	0.5K	0.9K {AC}	1 {C}	(0.9-0.5) × 1 = **0.4**

Figure 23: Selection of top 4 views using RLGA

From the Figure 22, it can be seen that the top 4 beneficial views selected by the HRU algorithm are AC, BC, B and A. RLGA also selects the same top 4 beneficial views, AC, BC, B and A, for materialization as shown in Figure 23. The number of re-computations required in each iteration for the HRU and the RLGA algorithm are shown in Figure 24.

Iteration	Number of Re-computations in HRU	Number of Re-computations in RLGA
1	0	0
2	4	2
3	2	1
4	0	0
Total	**6**	**3**

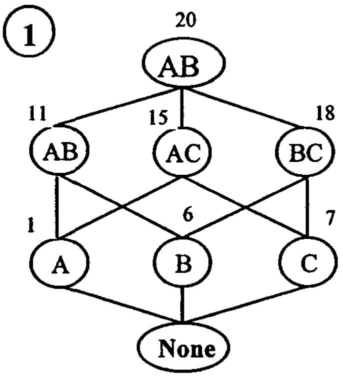
Figure 24: Number of re-computations in HRU and RLGA

It can be noted that the RLGA requires fewer (3) re-computations as compared to algorithm HRU (6). This difference is attributed to the fewer dependencies in the reduced lattice leading to fewer nodes requiring re-computation. Although the top 4 views selected by both algorithms are same, the re-computations required in case of RLGA is lesser than that of HRU. Therefore, it can be said that the reduced lattice is able to provide the same result as the complete lattice but with fewer re-computations. Further it needs to be shown that the algorithm RLGA gives results that are nearly as good as HRU for all possible reduction of a given lattice. This is illustrated with the help of an example given next.

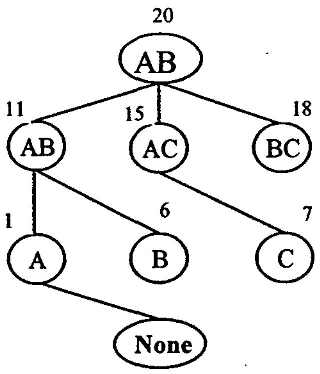
Example 6: Consider a three-dimensional lattice, arrive at all possible reductions of this lattice by varying the size of the views accordingly. Select the beneficial views for materialization using HRU and RLGA.

The 24 possible reductions of a given three dimensional lattice and the beneficial views selected by the two algorithms, HRU using complete lattice and RLGA using reduced lattice, is given below:

1

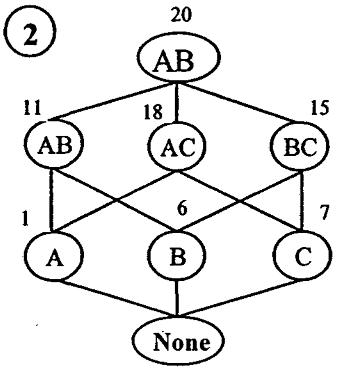


View	Benefit
AB	27
C	13
A	10
AC	5
B	5
BC	2

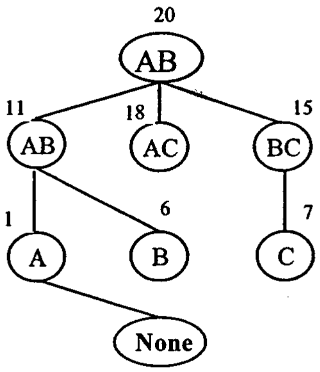


View	Benefit
AB	27
C	13
A	10
AC	5
B	5
BC	2

2

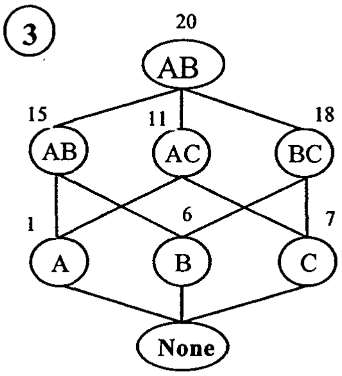


View	Benefit
AB	27
C	13
A	10
BC	5
B	5
AC	2

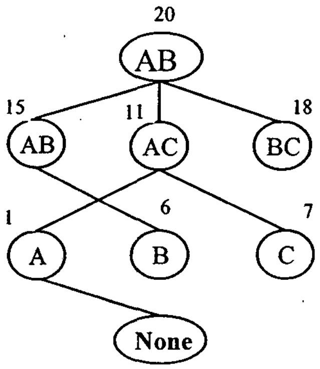


View	Benefit
AB	27
C	13
A	10
BC	5
B	5
AC	2

3

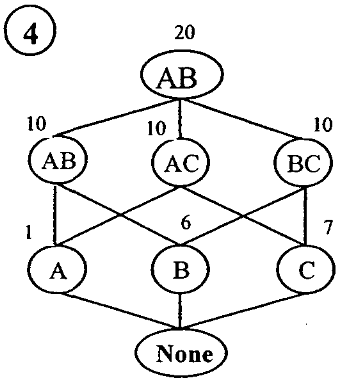


View	Benefit
AC	27
B	13
A	10
AB	5
C	5
BC	2

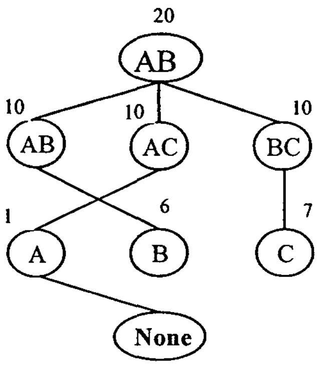


View	Benefit
AC	27
B	13
A	10
AB	5
C	5
BC	2

4

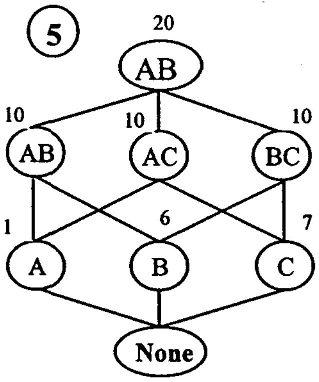


View	Benefit
AB	30
BC	20
AC	10
A	9
B	4
C	3

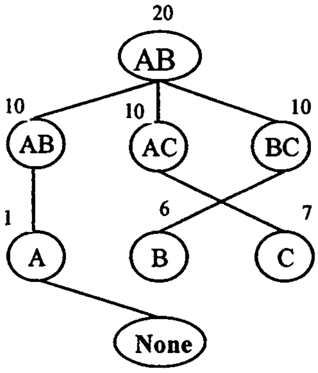


View	Benefit
AB	20
BC	20
AC	20
A	9
B	4
C	3

5

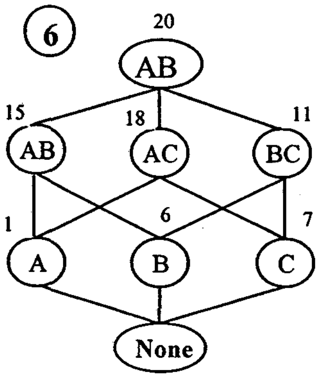


View	Benefit
AB	30
BC	20
AC	10
A	9
B	4
C	3

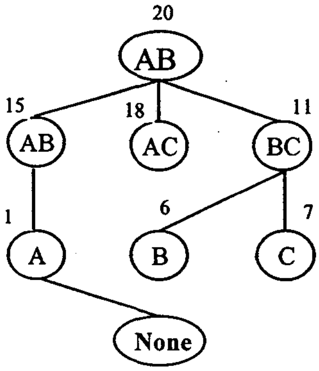


View	Benefit
AB	20
BC	20
AC	20
A	9
B	4
C	3

6

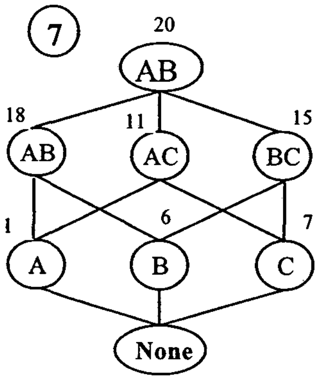


View	Benefit
BC	27
A	19
AB	5
B	5
C	4
AC	2

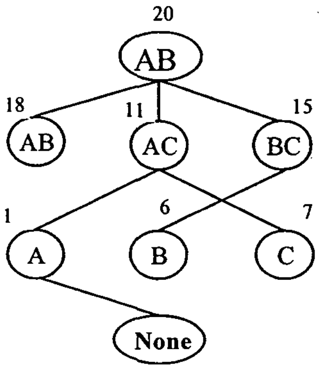


View	Benefit
BC	27
A	19
AB	5
B	5
C	4
AC	2

7

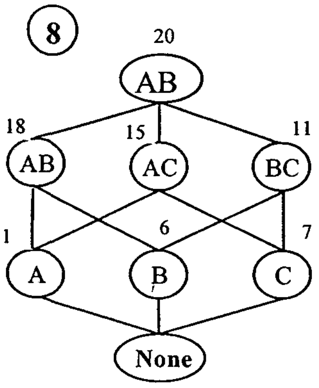


View	Benefit
AC	27
B	13
A	10
BC	5
C	5
AB	2

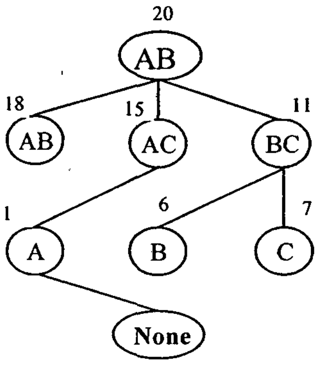


View	Benefit
AC	27
B	13
A	10
BC	5
C	5
AB	2

8

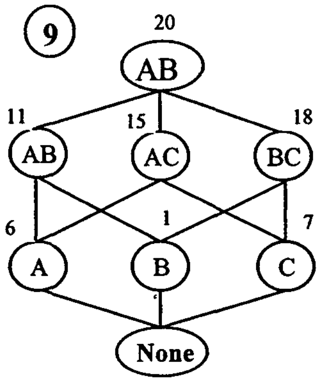


View	Benefit
BC	27
A	19
AC	5
B	5
C	4
AB	2

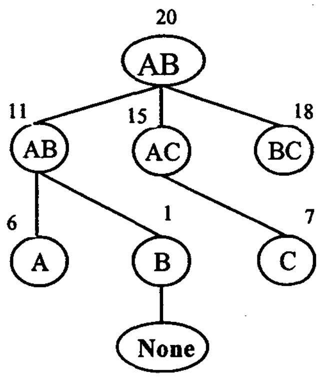


View	Benefit
BC	27
A	19
AC	5
B	5
C	4
AB	2

9

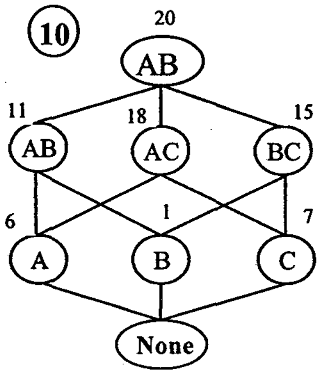


View	Benefit
AB	27
C	13
B	10
AC	5
A	5
BC	2

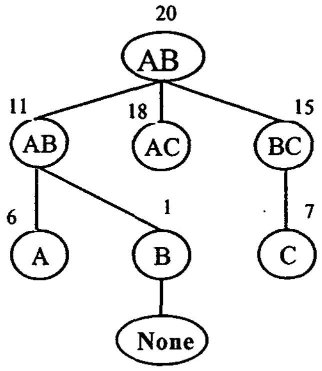


View	Benefit
AB	27
C	13
B	10
AC	5
A	5
BC	2

10

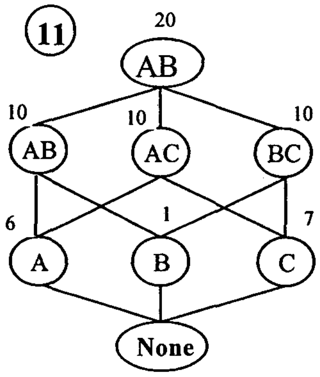


View	Benefit
AB	27
C	13
B	10
BC	5
A	5
AC	2

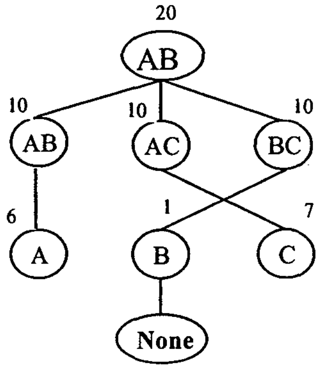


View	Benefit
AB	27
C	13
B	10
BC	5
A	5
AC	2

11

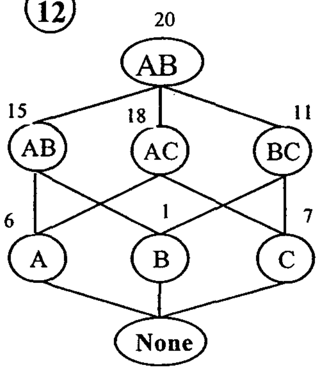


View	Benefit
AB	30
BC	20
AC	10
B	9
A	4
C	3

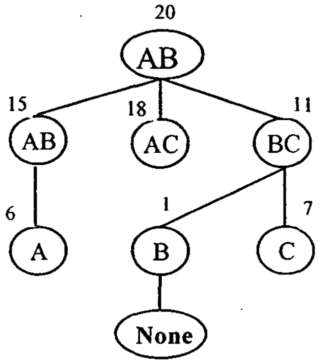


View	Benefit
AB	20
BC	20
AC	20
B	9
A	4
C	3

12

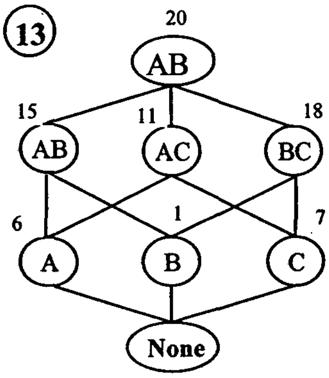


View	Benefit
BC	27
A	14
B	10
AB	5
C	4
AC	2

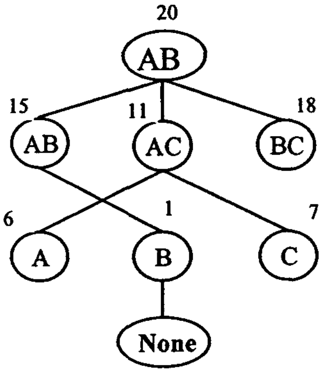


View	Benefit
BC	27
A	14
B	10
AB	5
C	4
AC	2

13

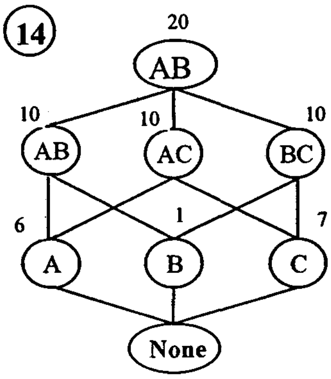


View	Benefit
AC	27
B	19
AB	5
A	5
C	4
BC	2

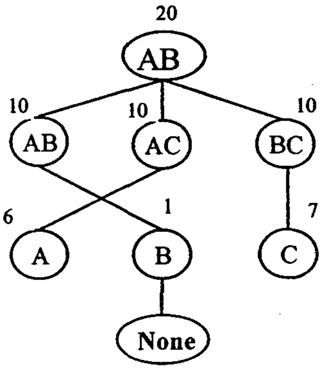


View	Benefit
AC	27
B	19
AB	5
A	5
C	4
BC	2

14

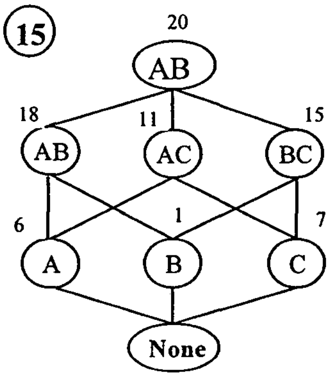


View	Benefit
AB	30
BC	20
AC	10
B	9
A	4
C	3

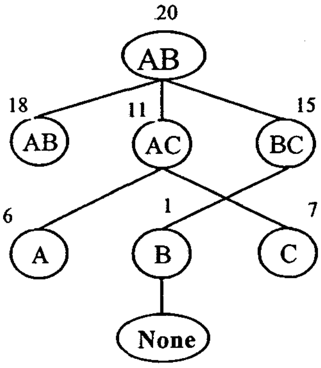


View	Benefit
AB	20
BC	20
AC	20
B	9
A	4
C	3

15

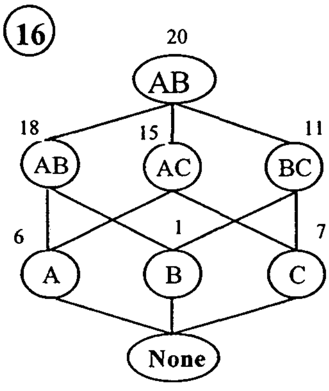


View	Benefit
AC	27
B	19
BC	5
A	5
C	4
AB	2

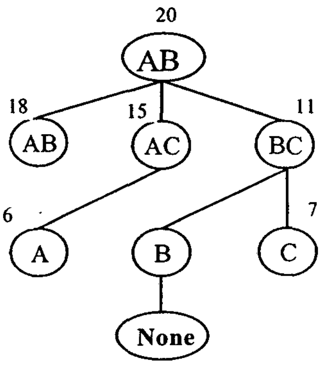


View	Benefit
AC	27
B	19
BC	5
A	5
C	4
AB	2

16

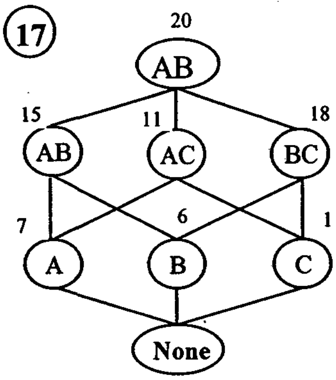


View	Benefit
BC	27
A	14
AB	10
AC	5
C	4
AB	2

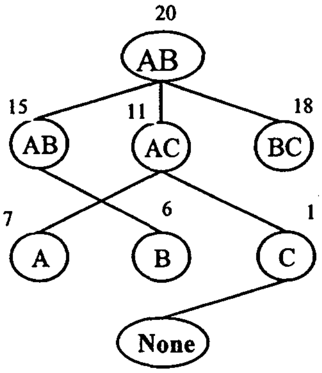


View	Benefit
BC	27
A	14
AB	10
AC	5
C	4
AB	2

17

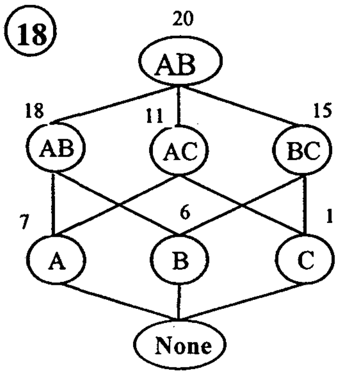


View	Benefit
AC	27
B	14
C	10
AB	5
A	4
BC	2

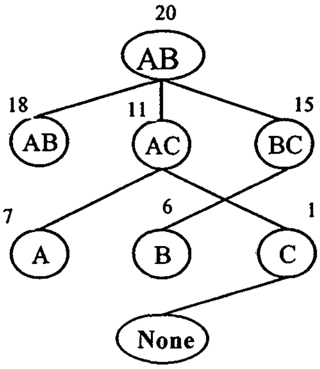


View	Benefit
AC	27
B	14
C	10
AB	5
A	4
BC	2

18

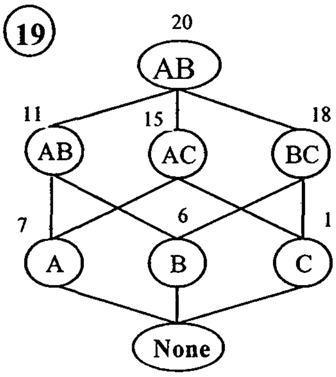


View	Benefit
AC	27
B	14
C	10
BC	5
A	4
AB	2

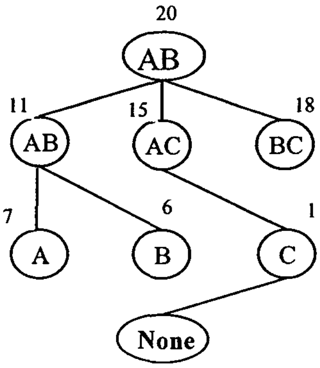


View	Benefit
AC	27
B	14
C	10
BC	5
A	4
AB	2

19

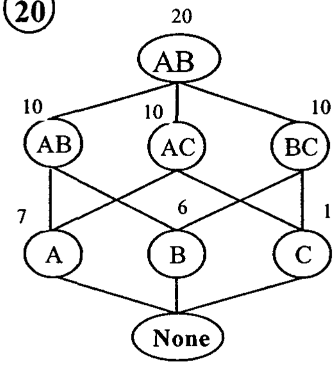


View	Benefit
AB	27
C	19
AC	5
B	5
A	4
BC	2

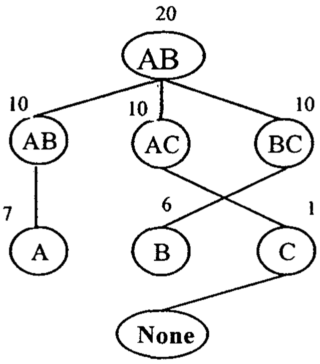


View	Benefit
AB	27
C	19
AC	5
B	5
A	4
BC	2

20

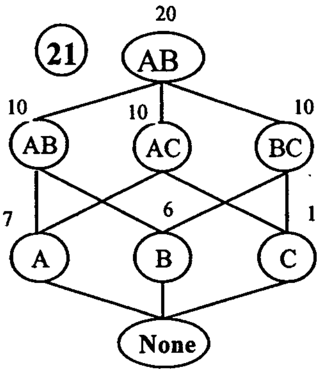


View	Benefit
AB	30
BC	20
AC	10
C	9
B	4
A	3

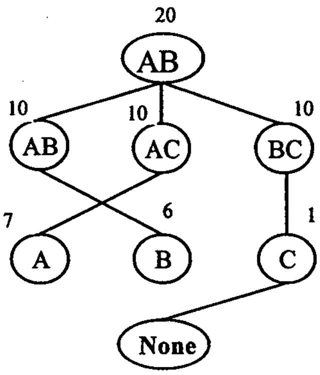


View	Benefit
AB	20
BC	20
AC	20
C	9
B	4
A	3

21

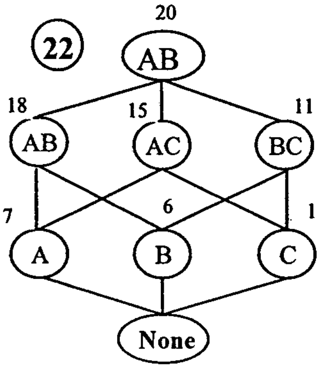


View	Benefit
AB	30
BC	20
AC	10
C	9
B	4
A	3

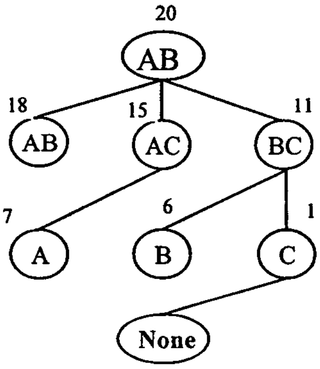


View	Benefit
AB	20
BC	20
AC	20
C	9
B	4
A	3

22

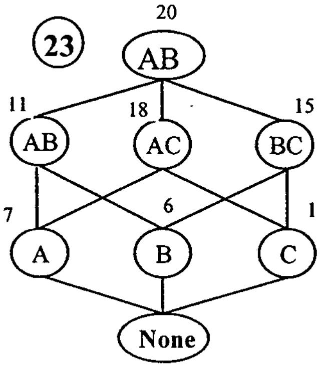


View	Benefit
BC	27
A	13
C	10
AC	5
B	5
AB	2

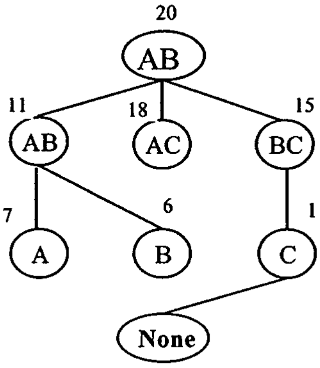


View	Benefit
BC	27
A	13
C	10
AC	5
B	5
AB	2

23

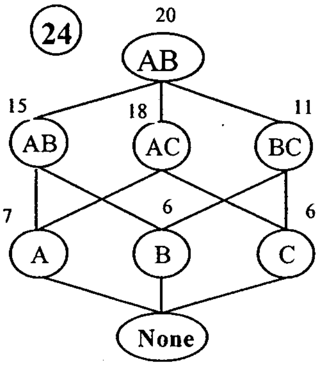


View	Benefit
AB	27
C	19
BC	5
B	5
A	4
AC	2

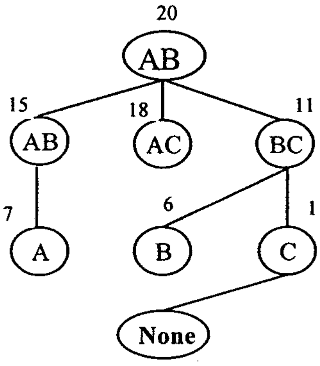


View	Benefit
AB	27
C	19
BC	5
B	5
A	4
AC	2

24



View	Benefit
BC	27
A	13
C	10
AB	5
B	5
AC	2



View	Benefit
BC	27
A	13
C	10
AB	5
B	5
AC	2

It can be observed from the 24 possible cases of lattice reduction that the same views are selected, in each case, by the two algorithms HRU, using complete lattice, and RLGA, using reduced lattice. In the cases 4, 5, 11, 14, 20 and 21, the top three views AB, BC and AC have the same benefit value in the first iteration and can be selected in any order by either of the algorithms. Therefore, in these cases any possible order of selecting the top three views is equally beneficial. Further in these six cases, the top views selected by HRU and RLGA are similar, they only differ with respect to the benefit value, which does not impact the selection of the beneficial views.

It can be said from above that in all the 24 cases, both algorithms select the top beneficial views. In the six cases mentioned above, they may select views in slightly different order since more than one view has the same benefit value. Whereas, in the other eighteen cases, they select the beneficial views in the same order. Thus, it can be concluded that RLGA is equally effective as HRU with respect to selecting the top beneficial views. The complexity analysis of algorithm RLGA is given next.

2.2.1 Complexity Analysis - RLGA

As discussed above, the RLGA heuristic is based upon the reduced lattice. The reduced lattice retains only the dependencies between a node and its smallest parent. As a result each node of the reduced lattice, except the root node, only contains one edge to its immediate parent. Thus, for a total of N nodes (views) in the reduced lattice, there are N-1 dependencies between each node and its immediate smallest parent. Conversely, looking at the dependencies from a parent to its child nodes, in the worst case, the maximum possible direct dependents on a node of the reduced lattice is N/2-1, for a node at a level below the root node. Similarly, in the worst case the maximum number of dependents of a node at two levels below the root node, can have a maximum of N/4-1 dependents. This feature of the reduced lattice can be clearly seen in the Figure 25 showing the reduced lattice for a data containing 4 dimensions.

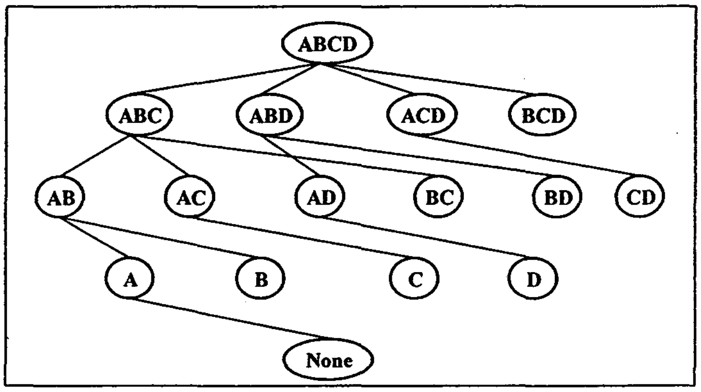


Figure 25: Reduced Lattice

In the reduced lattice of Figure 25, view ABC at a level below the root node of the lattice, has (D-1), i.e. 3 direct dependents. Similarly the view AB at two levels below the root node, has (D-2), i.e. 2 direct dependents. In this way thus, the maximum number of dependents on a node can be computed to be:

$$(N/2 -1) + (N/4 -1) +\dots (N/N -1) = N - \log_2 N$$

Further, in each of the iteration of the RLGA algorithm, a view is selected for materialization. As a result of this selection, the benefit value for all dependent views needs to be recomputed. Therefore, in the worst case, at the most the benefit value for $(N - \log_2 N)$ nodes would have to be recomputed in a single iteration of the algorithm. Therefore, the complexity of the RLGA algorithm for selecting top K views, by executing K iterations of the algorithm, can be computed as $O(K)* O(N - \log_2 N) = O(KN)$.

In order to compare the performance of RLGA with respect to HRU, both the algorithms are implemented and run on the data sets with varying dimensions. The performance based comparisons of HRU and RLGA is given in section 2.2.2.

2.2.2 Comparisons (HRU Vs. RLGA)

The RLGA and HRU algorithms were compared by conducting experiments on an Intel based 2.3 GHz PC having 2 GB RAM. The two algorithms were compared on parameters like number of re-computations and the execution time. First the total number of re-computations was captured against the number of dimensions. The graph is shown in Figure 26.

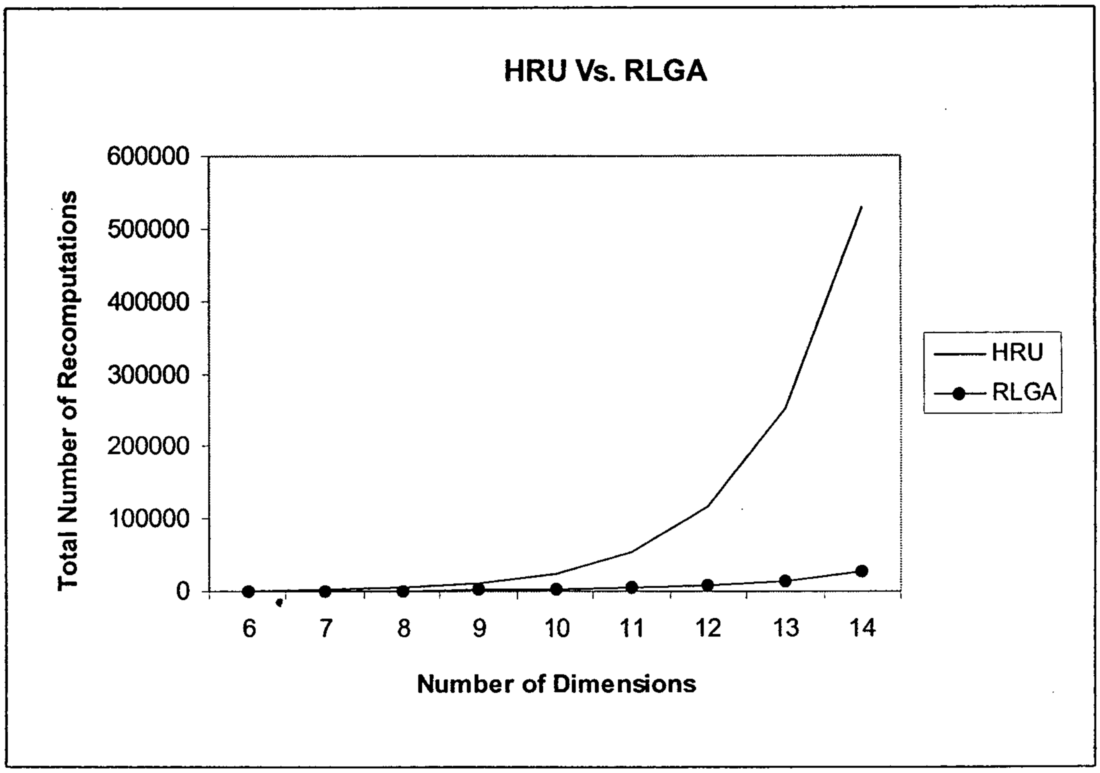


Figure 26: HRU Vs. RLGA: Total Number of Re-computations Vs. Number of Dimensions

It can be note from the graph the number of re-computations required for RLGA is lower than that for HRU. As the number of dimensions increase, this difference becomes significant. This lower number of re-computations in the case of RLGA algorithm can be attributed to the lower number of dependencies that exists within the reduced lattice. To better understand the difference, the number of re-computations was plotted individually for the dimensions 8, 10, 12 and 14 against the iteration of the algorithms. These graphs are shown in Figures 27 - 30.

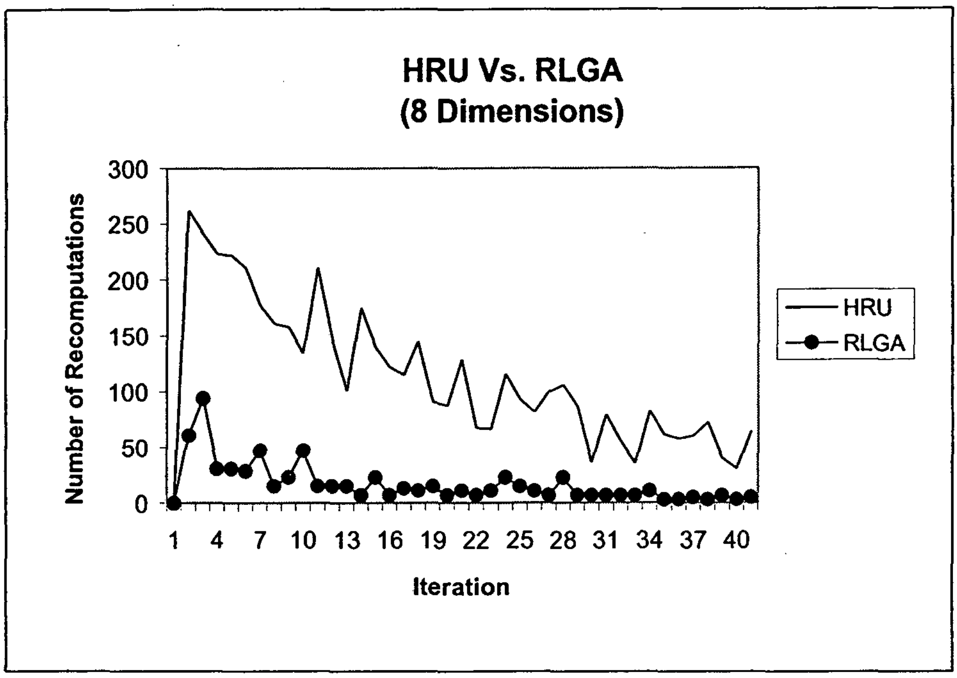


Figure 27: HRU Vs. RLGA: Number of Re-computations Vs. Iterations: 8 Dimensions

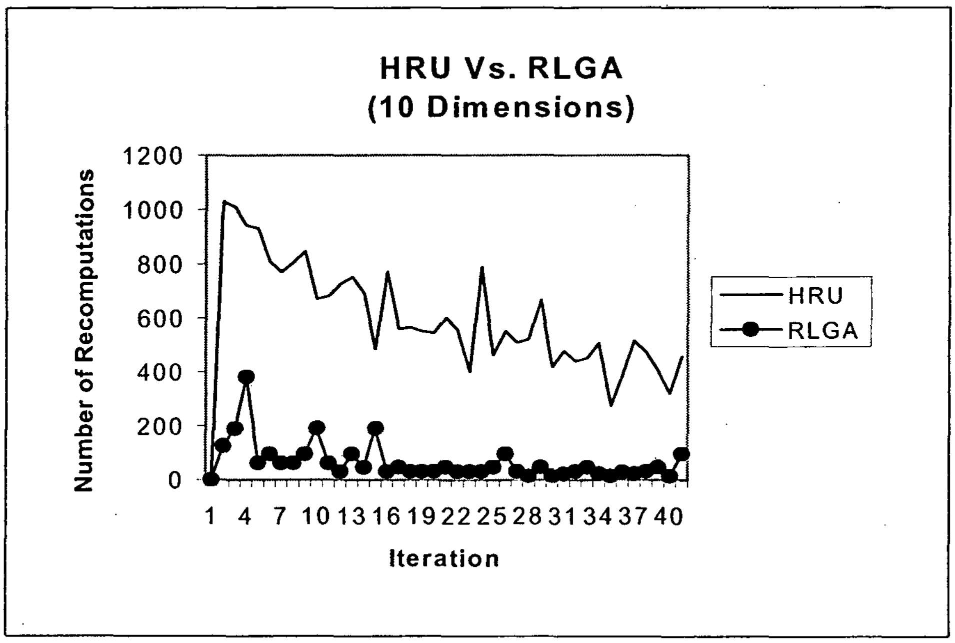


Figure 28: HRU Vs. RLGA: Number of Re-computations Vs. Iterations: 10 Dimensions

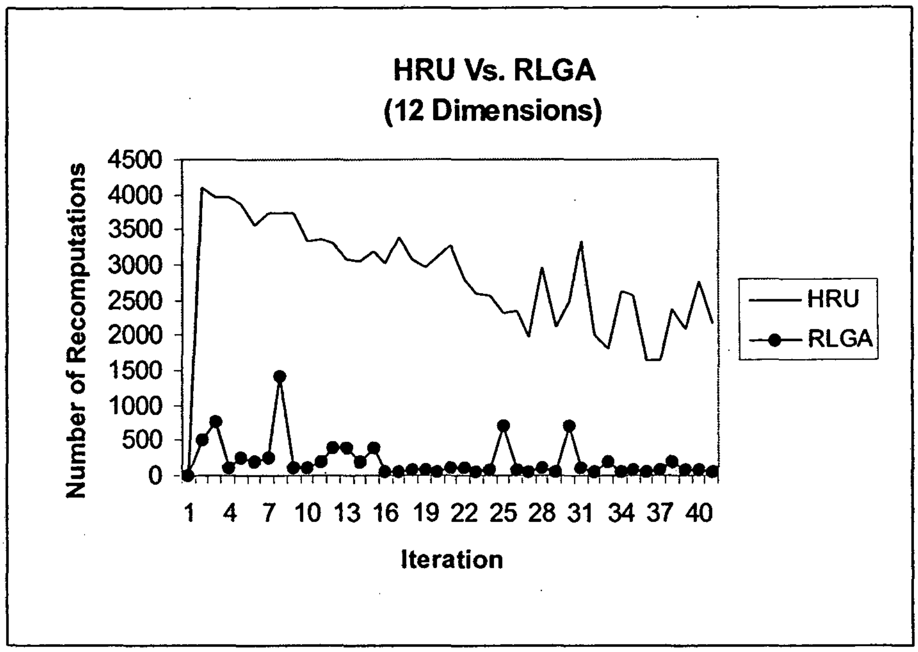


Figure 29: HRU Vs. RLGA: Number of Re-computations Vs. Iterations: 12 Dimensions

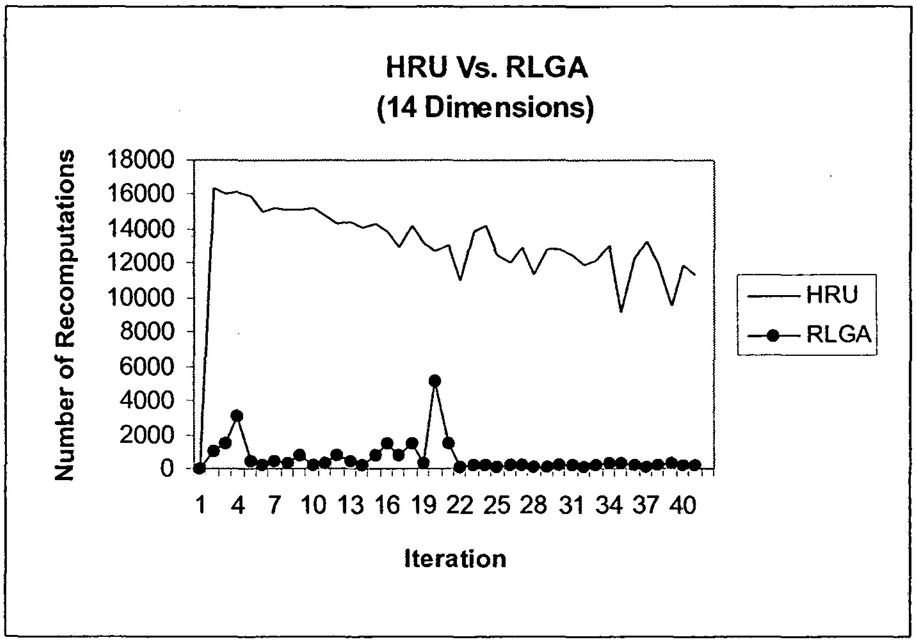


Figure 30: HRU Vs. RLGA: Number of Re-computations Vs. Iterations: 14 Dimensions

In each of these graphs, it can be clearly seen that numbers of re-computations for RLGA is far lower then that of HRU. These graphs comprehensively establish the first claim with regard to reducing lattices that it would result in substantially lower number of re-computations. Another observation that can be made from these graphs is that the number of re-computations in HRU decreases in the latter iterations.

Next, graphs were plotted to establish the second claim with regard to RLGA i.e. with the reduction in the number of dependencies, the execution time can be improved. The graph showing the execution time against the number of dimensions is shown in Figure 31.

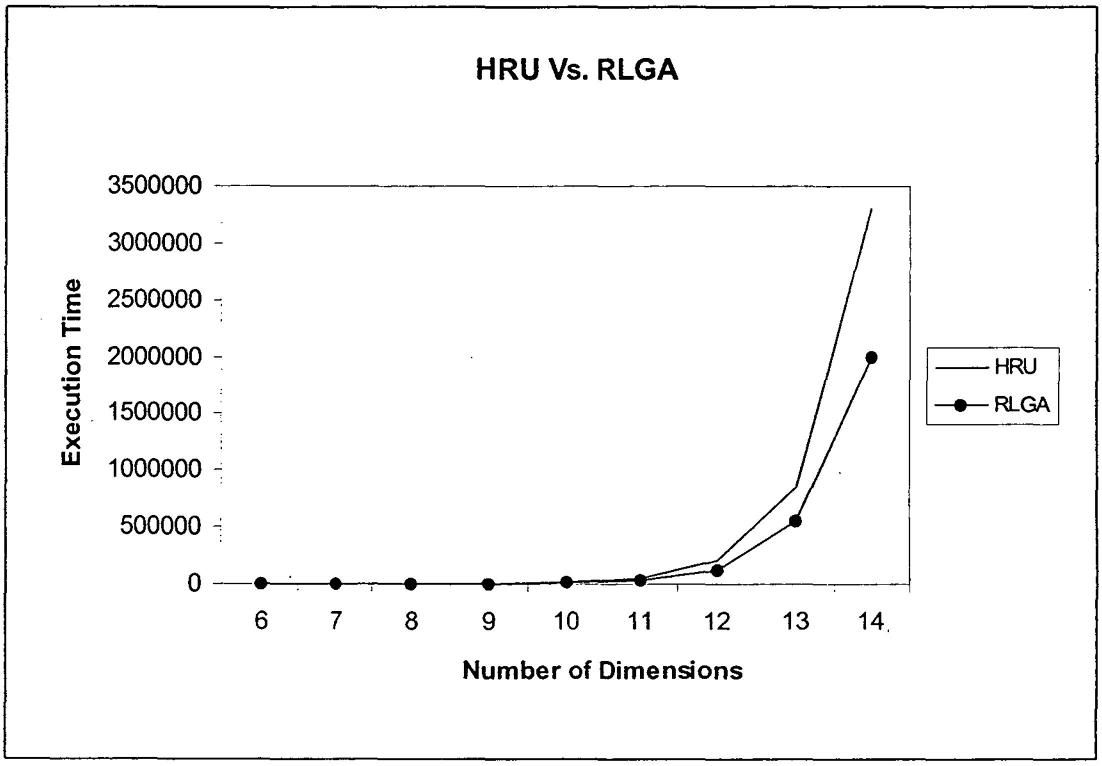


Figure 31: HRU Vs. RLGA: Execution Time Vs. Number of Dimensions

It can be observed that the execution time for RLGA is lower than that of HRU. This can be further explained by the Execution Time versus Iteration graphs for individual dimensions. These graphs are shown in Figures 32 – 35.

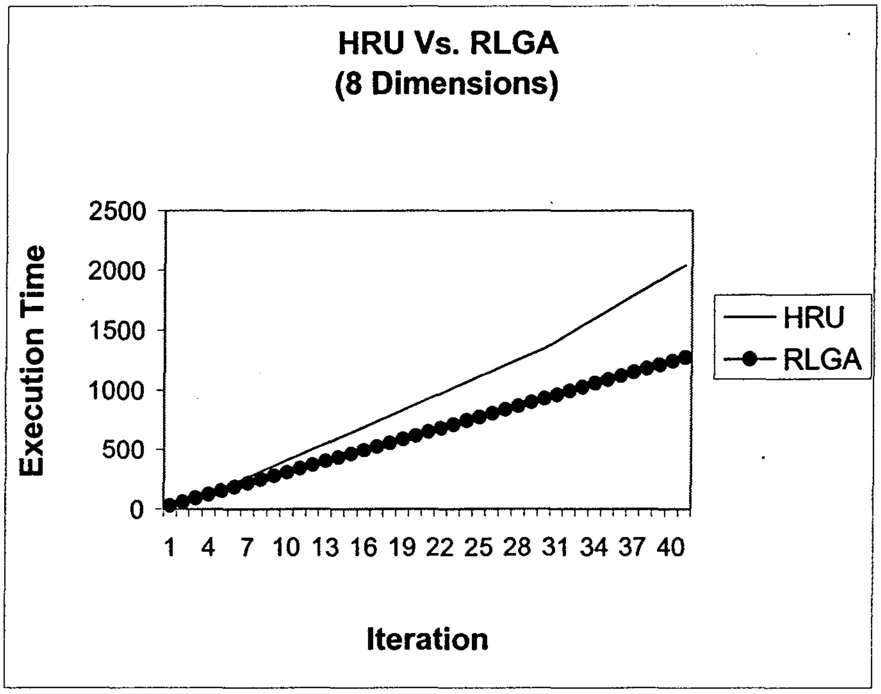


Figure 32: HRU Vs. RLGA: Execution Time Vs. Iteration: 8 Dimensions

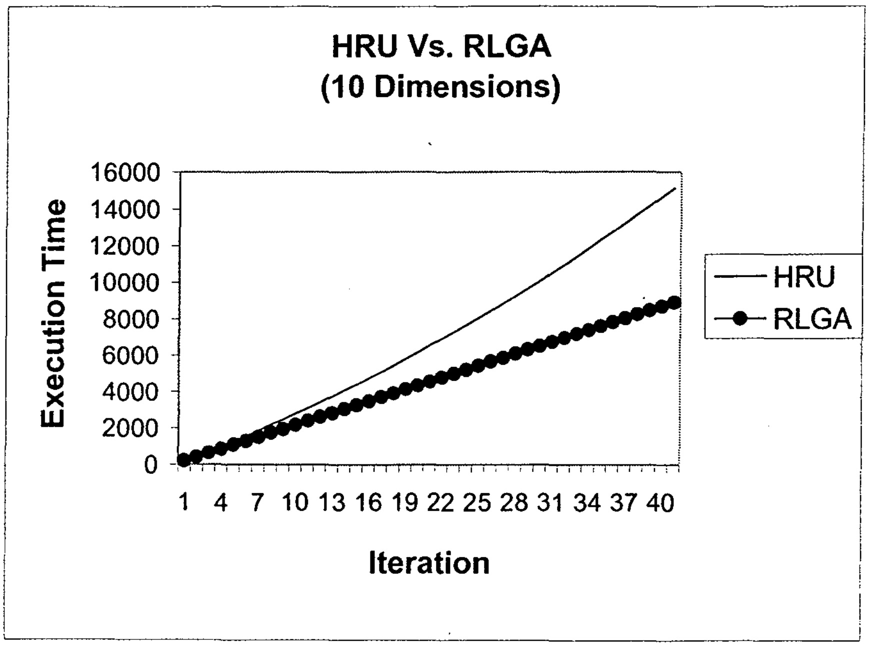


Figure 33: HRU Vs. RLGA: Execution Time Vs. Iteration: 10 Dimensions

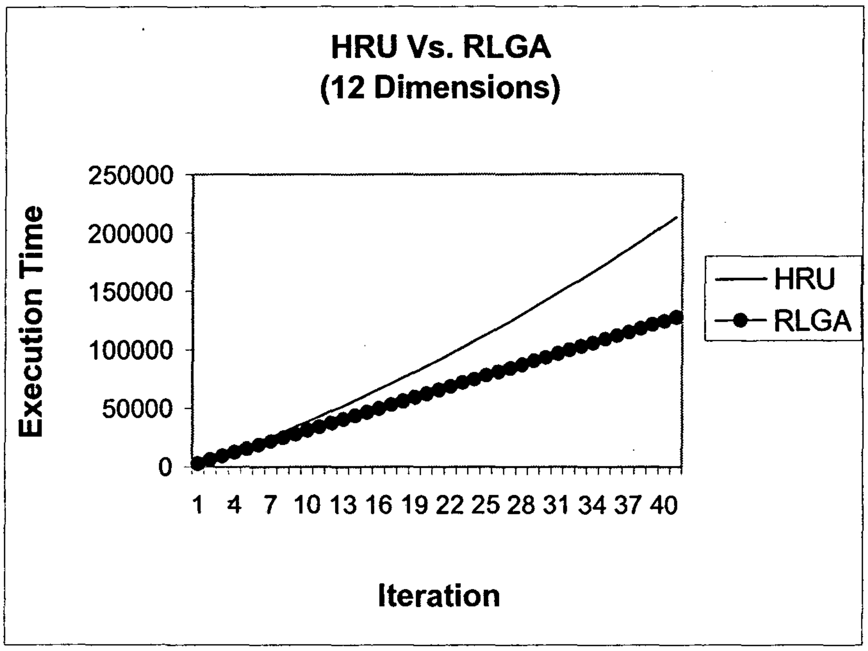


Figure 34: HRU Vs. RLGA: Execution Time Vs. Iteration: 12 Dimensions

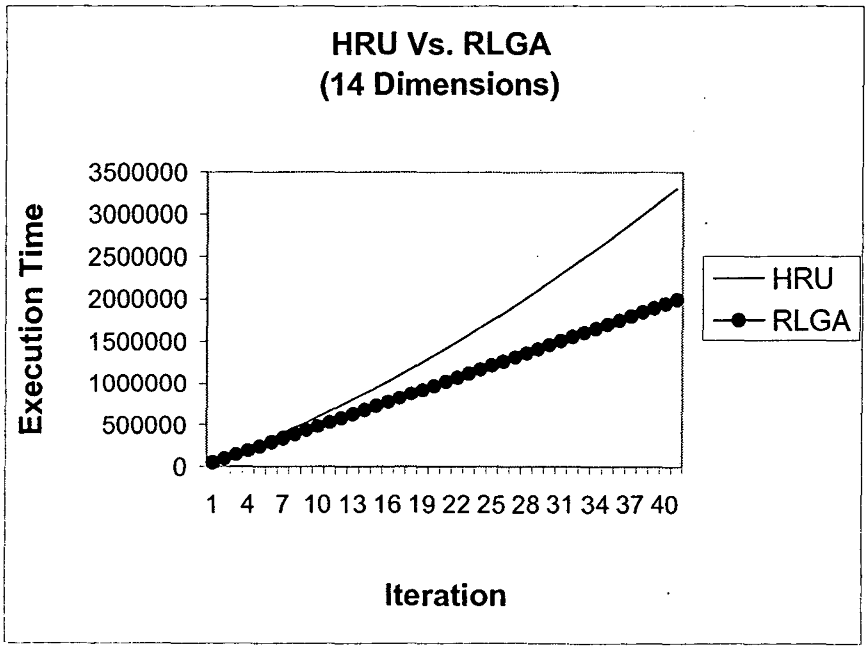


Figure 35: HRU Vs. RLGA: Execution Time Vs. Iteration: 14 Dimensions

2.3 Reduced Candidate Greedy Algorithm (RCGA)

The HRU algorithm discussed above has an exponential runtime complexity primarily because in each of the iteration of execution of the algorithm, benefit values of nearly all views of the lattice have to be evaluated. This exponentially high runtime complexity makes the HRU algorithm infeasible for high dimensional data sets as the execution time of the HRU algorithm becomes phenomenally high. An attempt is made to utilize the heuristic, used in RLGA, to improve on the scalability aspect of the view selection problem. For this, a Reduced Candidate Greedy Algorithm (RCGA) is proposed. The algorithm RCGA comprises of two stages namely candidate views recommendation and materialized view selection.

In the candidate views recommendation phase, candidate views are recommended for selection based on the same lines as the heuristic used for lattice reduction in section 2.2. The heuristic is defined as follows:

For each view, the smallest sized parent is retained for view selection

The basis of this heuristic is that in HRU algorithm, the view having the smallest size, among all views at a level of the lattice, has the maximum benefit at that level with respect to all the views on which it is dependent. This implies that picking the smallest sized view can maximize the benefit with respect to views above it in a lattice. In other words, retaining the smallest sized parent view is most beneficial, as the smallest sized parent will in turn have the maximum benefit with respect to the view above it. Thus these smallest sized beneficial parent views have a high likelihood of being selected for materialization and are therefore added to the candidate set in the recommendation stage of the algorithm.

The recommendation starts at bottom of the lattice. In the first iteration, as per the heuristic the smallest sized parent view from the bottom of the lattice, which has not yet been recommended, is added to the candidate set. Next, its smallest sized parent view which has not yet been recommended is added to the candidate set. This process of adding views to the candidate set continues till the root view is reached. These views in the candidate set are then recommended for the selection phase of the algorithm.

In the selection phase, the benefit value of the recommended views is computed using the benefit function of [HRU96] given below:

$$\text{Benefit} = (\text{Size}_{NMA} - \text{Size}_V) \times D_V$$

where,

Size_{NMA} = Size of the nearest materialized ancestor node of V

Size_V – Size of the view V

D = Number of dependents of V

The view, among the recommended views, having the highest benefit is selected for materialization.

The next iteration again starts from the bottom of the lattice by adding the smallest sized parent view, which has not yet been recommended, to the new candidate set. The smallest size parent view's addition to the candidate set is done as in the first iteration till the root node is reached. The most beneficial view from the set of recommended views in the candidate set is then selected for materialization.

The iterations continue as above till the top T views are selected for materialization. The algorithm RCGA is given in Figure 36. An example worked out using this algorithm is given next after Figure 36.

```
Input:   Dependencies D among Views V
Output: Top T views
Method:
    1. // Construct Lattice of Views using Dependencies D among them
       L=ConstructLattice(D)
       Count :=0; MatViewsList:=NULL;
    1. WHILE (Count < T) DO
        a. // Candidate Views Recommendation
           R:= $\text{Node}_{\text{NONE}}$; W:={} // Set of candidate views
        b. WHILE (More nodes to consider at a level above R)
            i. FOR (Each Node from one level above R that has not yet been recommended)
               Identify smallest node $r_{\text{SMALLEST}}$
            ii. W:= W U { $r_{\text{SMALLEST}}$ }
            iii. R:= $r_{\text{SMALLEST}}$
        c. // Selection of Views
           For every view N, from set of candidate views W, compute the following using L:
            i. $\text{NoDependents}_N$ := COUNT (Number of views appearing at a lower level in
               the lattice and having $\text{View}_N$ as an ancestor)
            ii. $\text{BenefitValView}_N$ := ($\text{Size}_{\text{ROOT}}$ - $\text{Size}_N$) * $\text{NoDependents}_N$
            iii. $\text{CommonView}_{NM}$ : = Identify the direct depedent views common with every
                 other view M, from the same level in the lattice as N
            iv. Prepare $\text{OrderedListBeneficialViews}_N$ =
                ORDER_DESC ($\text{BenefitValView}_N$ * $\text{NoDependndnts}_N$)
        d. Select top view ($V_{\text{TOP}}$) from OrderedListBeneficialViews and add $V_{\text{TOP}}$ to
           MatViewsList
        e. // Update benefit value and no of dependents
            i. For every view $V_K$ from the same level in the lattice as $V_{\text{TOP}}$,
               Having a common node $V_{\text{COMMON}}$ := $\text{CommonView}_{\text{KTOP}}$
               Adjust $\text{NoDependents}_K$ := $\text{NoDependents}_K$ - $\text{NoDependents}_{\text{COMMON}}$
            ii. For every view $V_J$ from a higher level in the lattice than $V_{\text{TOP}}$, s.t. $V_{\text{TOP}} \rightarrow V_J$
                Adjust $\text{NoDependents}_J$ := $\text{NoDepenendents}_J$ - $\text{NoDependents}_{\text{TOP}}$
            iii. For every view $V_M$ from a lower level in lattice than $V_{\text{TOP}}$, s.t. $V_M \rightarrow V_{\text{TOP}}$
                 Adjust $\text{BenefitValView}_M$ = $\text{Size}_{\text{TOP}}$ - $\text{Size}_M$
            iv. Remove $V_{\text{TOP}}$ from OrderedListBeneficialViews
            v. Count ++
    2. Return MatViewList
```

Figure 36: Algorithm RCGA

Example 7: Consider the dependencies among views of Example 2 and the corresponding lattice in Example 3. Arrive at the top 3 materialized views using algorithm RCGA.

The iteration wise recommendation and selection of views using RCGA is given in Figure 37. The top 3 views selected by RCGA algorithms are AC, BC and B. These are the same views as selected by HRU over the same lattice. A more detailed comparison of the algorithm RCGA against HRU is given in section 2.3.2.

	Views Recommended (V)	Size of the View (Size$_V$)	Size of the Nearest Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit = (Size$_{MAN}$- Size$_V$) × D
First Iteration	A	0.09	5K {ABC}	1 {A}	(5-0.09) * 1 = **4.91**
	AC	0.9	5K {ABC}	3 {AC, A, C}	(5-0.9) * 3 = **12.3**
	ABC	5	5K {ABC}	7 {AB, AC, BC, AC, A, B, C}	(5-5) * 7 = **0**

	Views Recommended (V)	Size of the View (Size$_V$)	Size of the Nearest Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit = (Size$_{MAN}$- Size$_V$) × D
Second Iteration	C	0.5	0.9K {AC}	1 {C}	(0.9 – 0.5) * 1 = **0.4**
	BC	3	5K {ABC}	2 {BC, C}	(5-3) * 2 = **4**

	Views Recommended (V)	Size of the View (Size$_V$)	Size of the Nearest Materialized Ancestor Node (Size$_{NMA}$)	Number of Dependents (D)	Benefit = (Size$_{MAN}$- Size$_V$) × I
Third Iteration	B	1	3K {BC}	1 {B}	(3 – 1) * 1 = **2**
	AB	5	5K {ABC}	2 {AB, B}	(5-5) * 2 = **0**

Figure 37: Selecting views for materialization using algorithm RCGA

The complexity analysis of algorithm RCGA is given next.

2.3.1 Complexity Analysis - RCGA

The RCGA algorithm initially recommends D views, for D dimensions of the data set. The recommendation of D views starts from the bottom of the lattice. Consider the lattice in Figure 38. First, the smallest parent view of the bottom most view of the lattice is recommended. In other words, a view from Level-1 of the lattice is recommended. It can seen from the Figure 38, that there are D views, V_1, V_2,..,V_D at Level-1 of the lattice. Identifying the smallest view, V_S, out of these D views can be performed by one pass over the D views, making it an O(D) operation. Next, the smallest parent view of the view V_S is identified from the Level-2, two levels above the bottom most node, of the lattice.

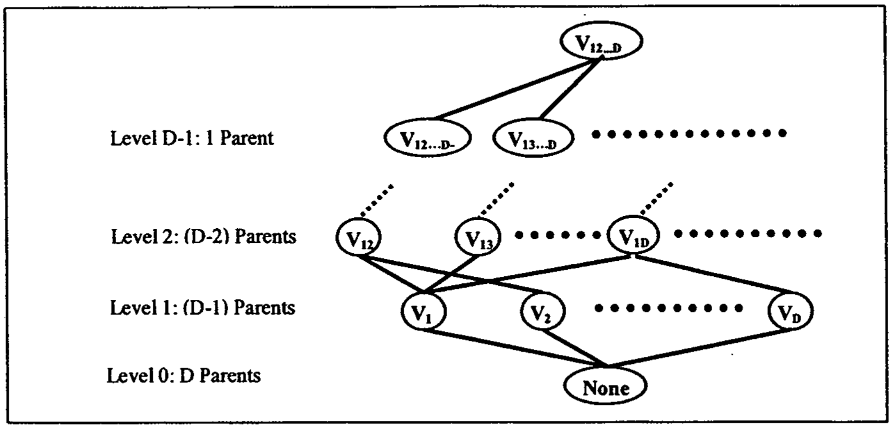


Figure 38: Number of parents at each level of a D-Dimensional Lattice

It can again be seen from the lattice in Figure 38, that a view from the Level-1 can form (D-1) aggregations with other (D-1) dimension of data at Level-1. Therefore, a view present at Level-1 of the lattice contains at most (D-1) parents. Identifying the smallest view from the (D-1) parent views of view V_S is an O(D-1) operation. Continuing this way, the smallest parent views from each of the level up to the root node of the lattice are recommended. The total numbers of such parent views that need to be evaluated are:

$$D + (D-1) + (D-2) + \ldots + 1 = D(D+1)/2$$

Therefore, the time complexity of the recommendation stage is $O(D^2)$.

From the D recommended views, the most beneficial view is selected for materialization. Identifying the most beneficial view requires one pass over each of the D recommended views, making it a O(D) time complexity operation.

So, the overall time complexity of an iteration of the RCGA algorithm is $O(D^2) + O(D)$, i.e. $O(D^2)$. For selecting K top views, therefore the overall time complexity works out to be $O(KD^2)$.

The performance comparison of RCGA with HRU and RCGA with PGA is given in the following two sub sections 2.3.2 and 2.3.3 respectively.

2.3.2 Comparisons (HRU Vs. RCGA)

The HRU and RCGA algorithms were compared by conducting experiments on an Intel based 2.3 GHz PC having 2 GB RAM. The two algorithms were compared on parameters like the execution time and the benefit value.

First, the two algorithms were compared on execution time versus the number of dimensions. The corresponding graph is shown in Figure 39.

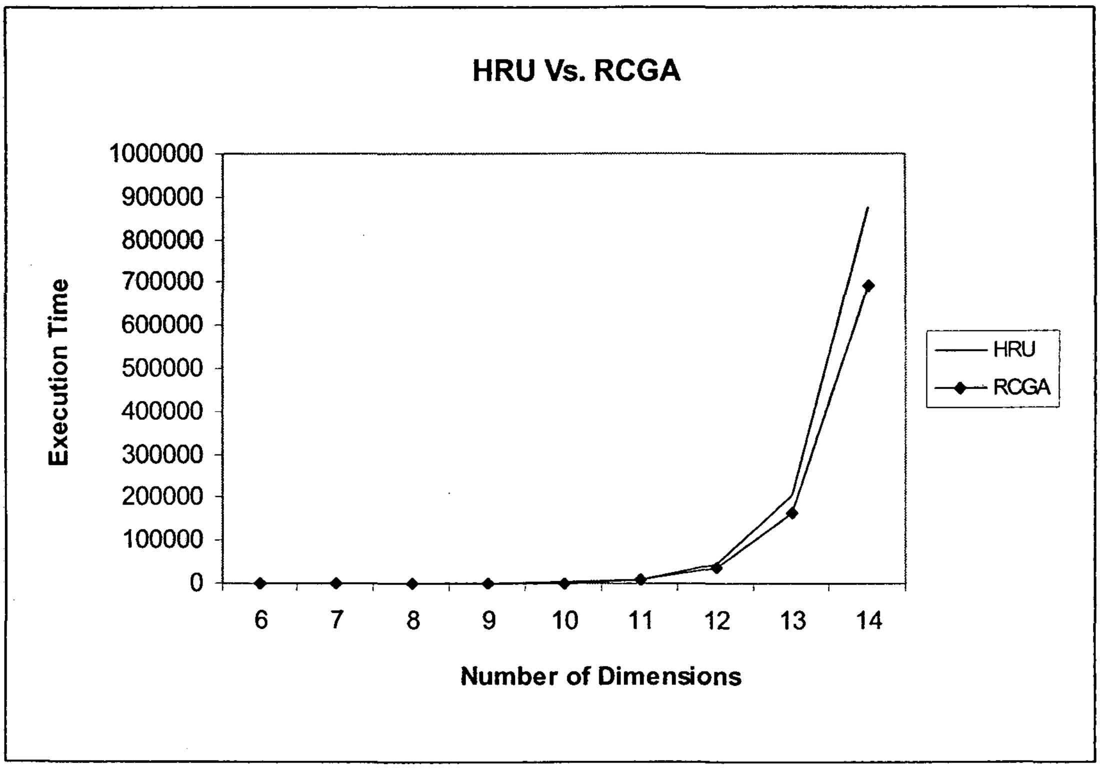


Figure 39: HRU Vs. RCGA: Execution Time Vs. Number of Dimensions

The graph shows that the algorithm RCGA has a much improved execution time as compared to HRU. This substantiates the claim that the heuristic used in RCGA will result in a much improved execution time. Graphs were also plotted for the Execution Time versus the Iteration for the dimensions 8, 10, 12 and 14. These graphs, shown in Figures 40- 43, also show that RCGA has a much better execution time than HRU.

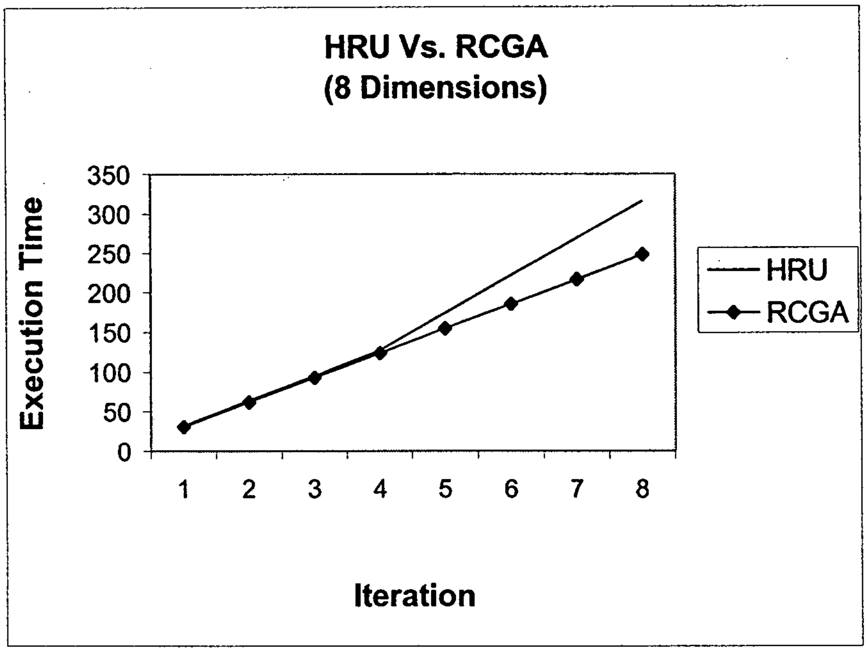


Figure 40: HRU Vs. RCGA: Execution Time Vs. Iteration: 8 Dimensions

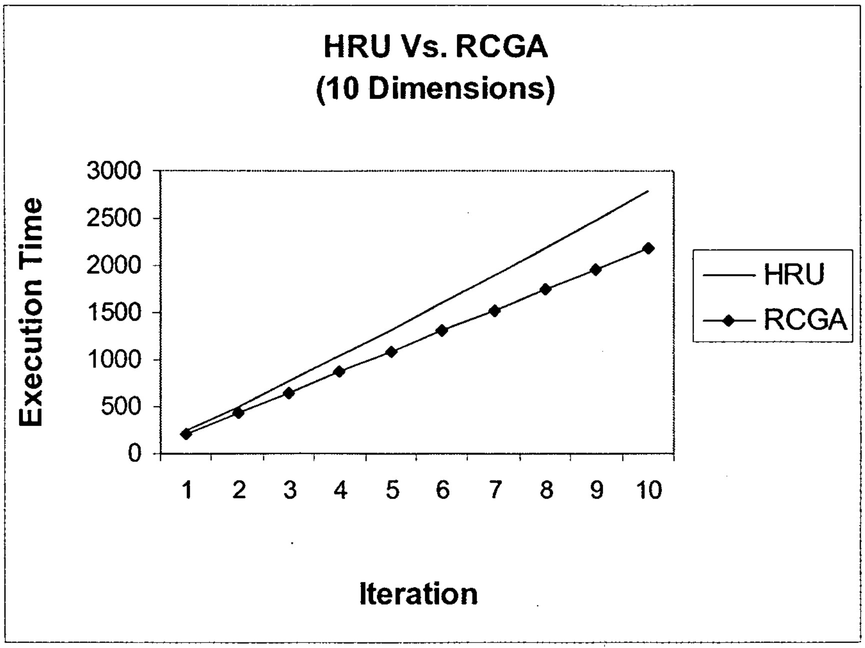


Figure 41: HRU Vs. RCGA: Execution Time Vs. Iteration: 10 Dimensions

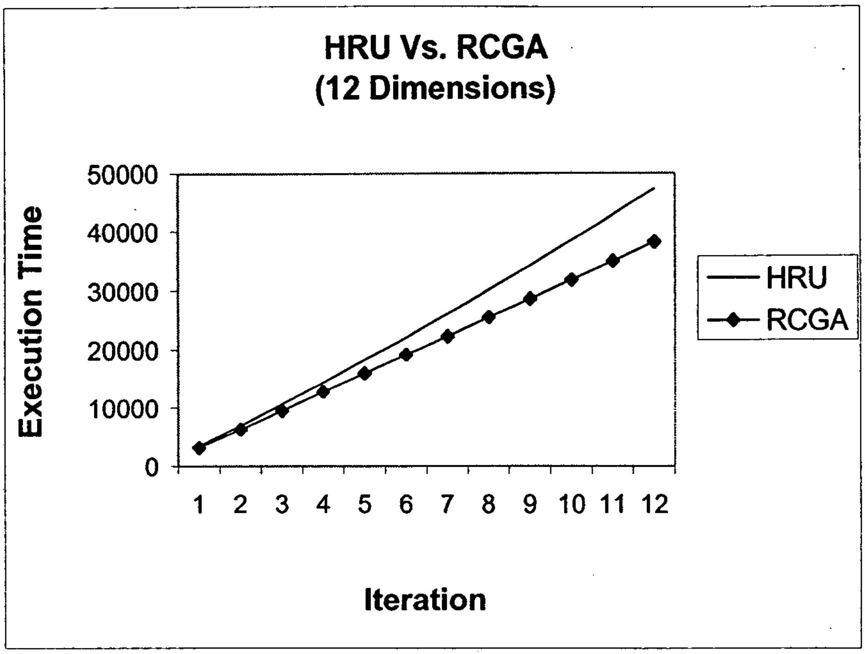


Figure 42: HRU Vs. RCGA: Execution Time Vs. Iteration: 12 Dimensions

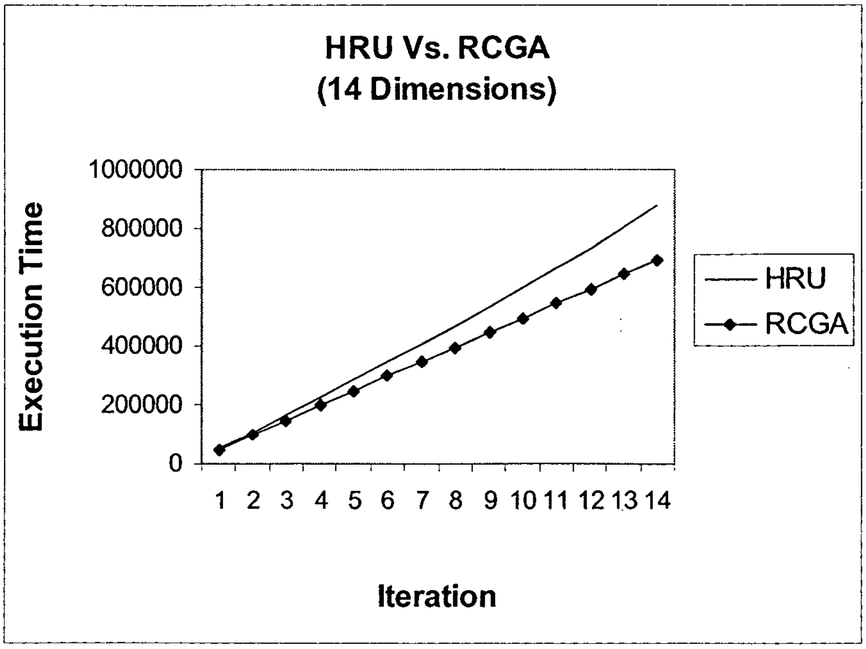


Figure 43: HRU Vs. RCGA: Execution Time Vs. Iteration: 14 Dimensions

Next, the two algorithms HRU and RCGA were compared on the benefit of the views selected with respect to dimensions. The benefit value versus the number of dimensions graph is shown in Figure 44.

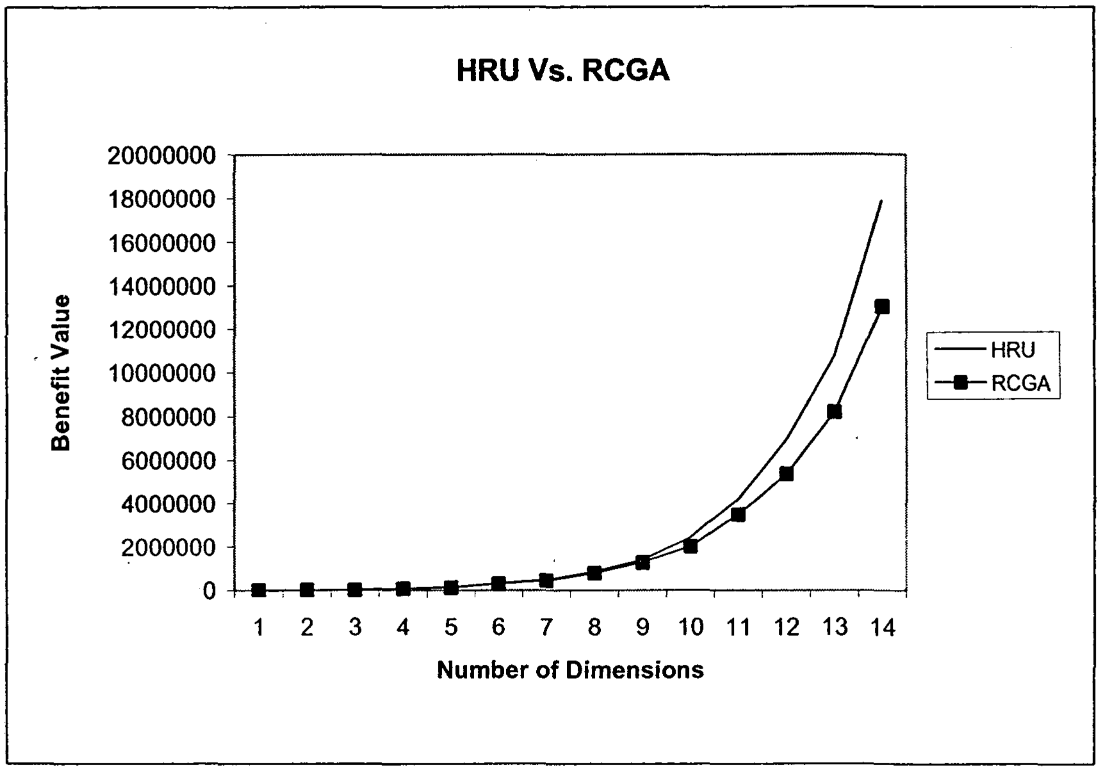


Figure 44: HRU Vs. RCGA: Benefit Value Vs. Number of Dimensions

The graph shows that the benefit value of RCGA is lower than that of HRU. This is due to heuristic followed in RCGA that trades benefit value for an improvement in scalability with respect to the dimensions.

Further, in order to observe how the benefit value change with respect to iteration for the two algorithms, graphs were plotted for dimension 8, 10, 12 and 14 as shown in Figures 45-48. The graphs show that the top views selected by the two algorithms have nearly similar benefit values, with HRU holding a slight edge over RCGA.

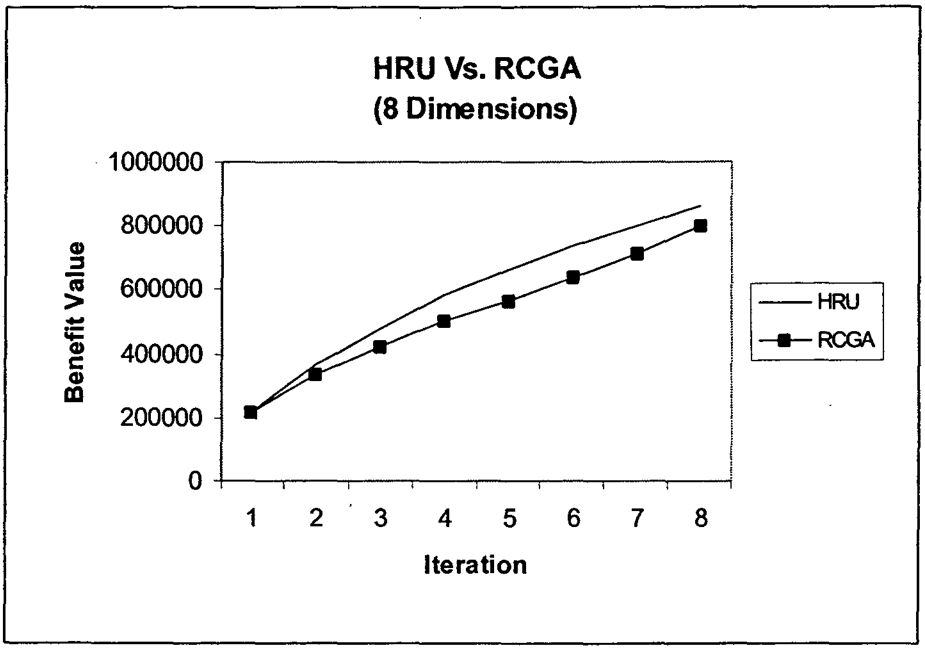


Figure 45: HRU Vs. RCGA: Benefit Value Vs. Iteration: 8 Dimensions

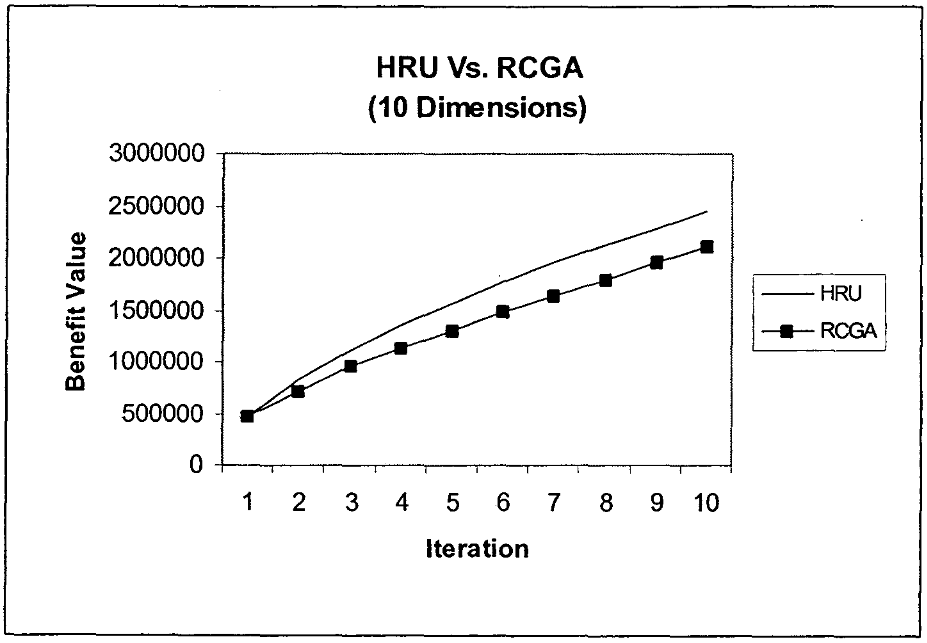


Figure 46: HRU Vs. RCGA: Benefit Value Vs. Iteration: 10 Dimensions

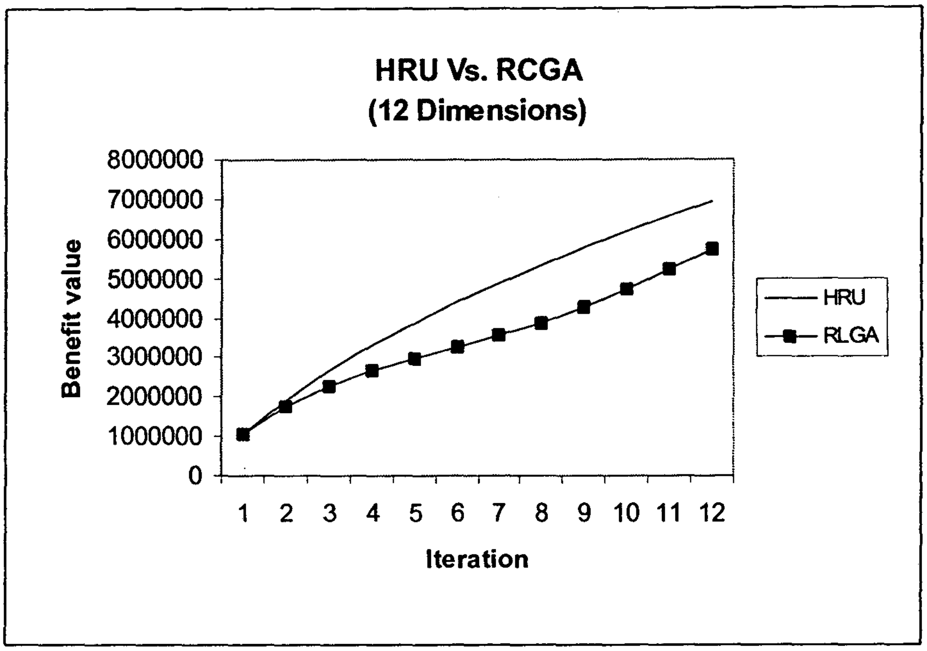


Figure 47: HRU Vs. RCGA: Benefit Value Vs. Iteration: 12 Dimensions

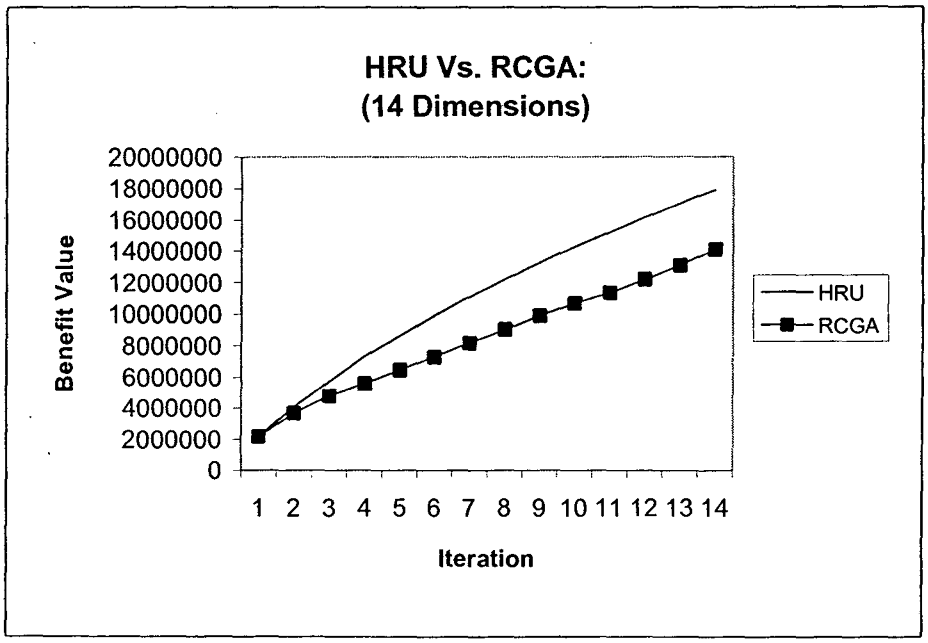


Figure 48: HRU Vs. RCGA: Benefit Value Vs. Iteration: 14 Dimensions

2.3.3 Comparisons (PGA Vs. RCGA)

RCGA was also compared with the algorithm PGA by conducting experiments on an Intel based 2.3 GHz PC having 2 GB RAM. The two algorithms were compared on parameters like the execution time and the benefit value.

The two algorithms were compared on the execution time versus the number of dimensions. The corresponding graph is shown in Figure 49.

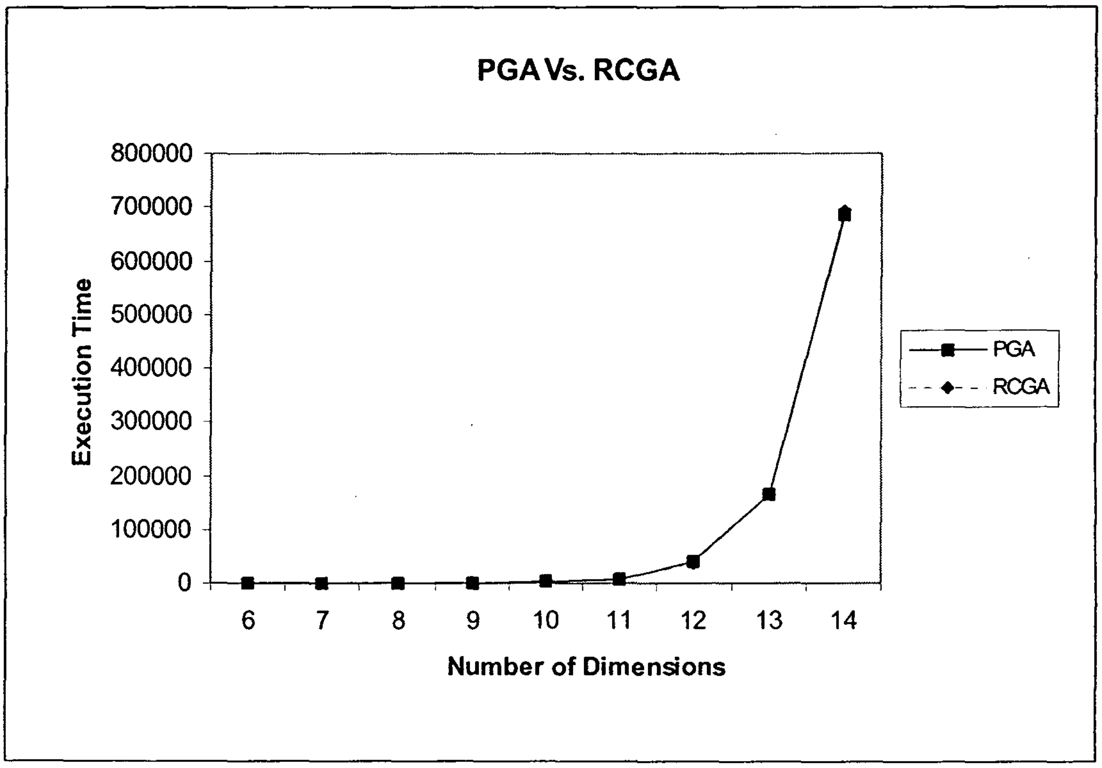


Figure 49: PGA Vs. RCGA: Execution Time Vs. Number of Dimensions

It can be seen from the graph that both the algorithms have nearly identical execution time, as depicted by the overlapping curves of RCGA and PGA.

Next, the execution time versus iteration graphs for the dimensions 8, 10, 12 and 14 were plotted for the two algorithms. These graphs, shown in Figures 50 – 53, also depict the same overlapping trends in their corresponding curves.

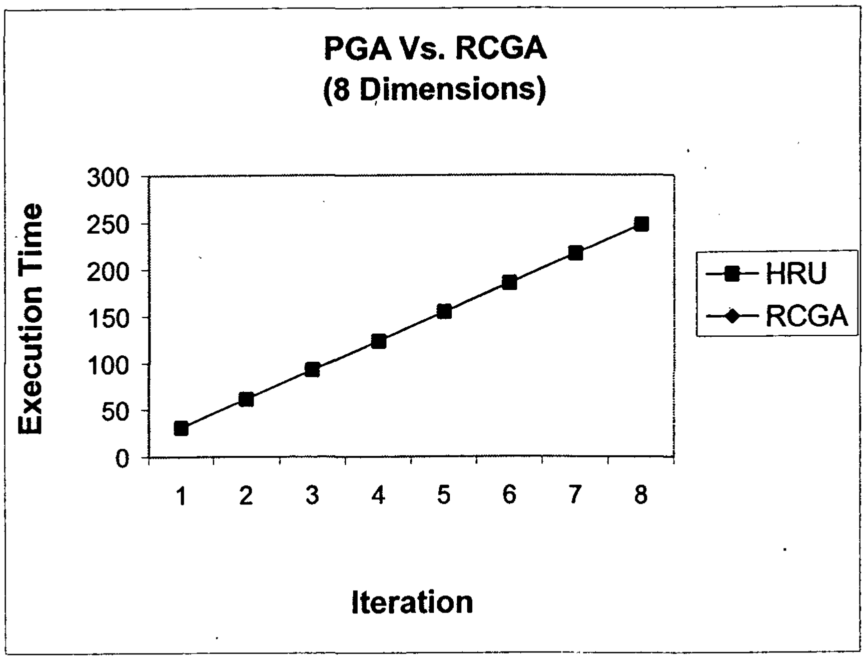


Figure 50: PGA Vs. RCGA: Execution Time Vs. Iteration: 8 Dimensions

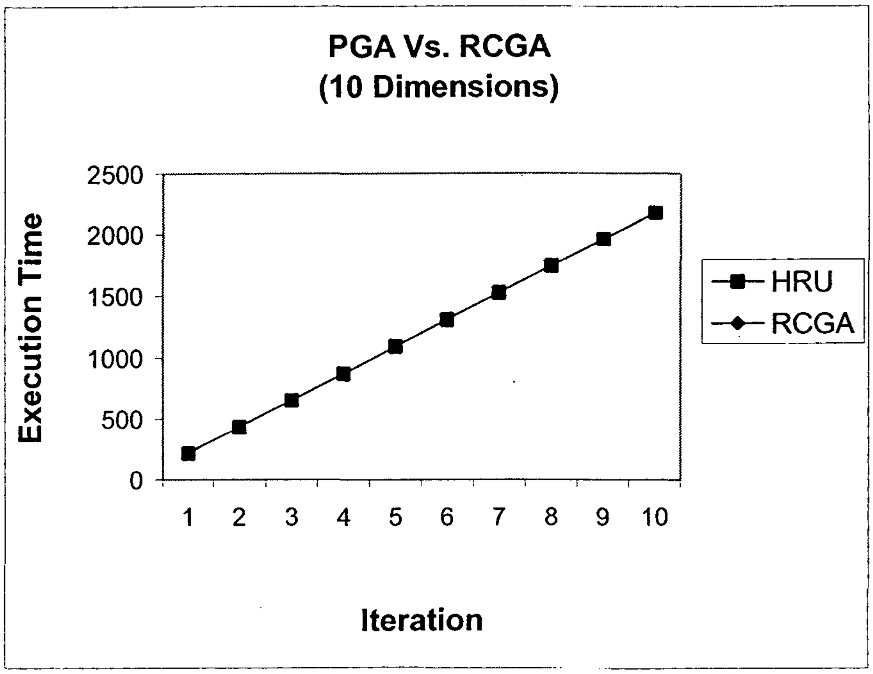


Figure 51: PGA Vs. RCGA: Execution Time Vs. Iteration: 10 Dimensions

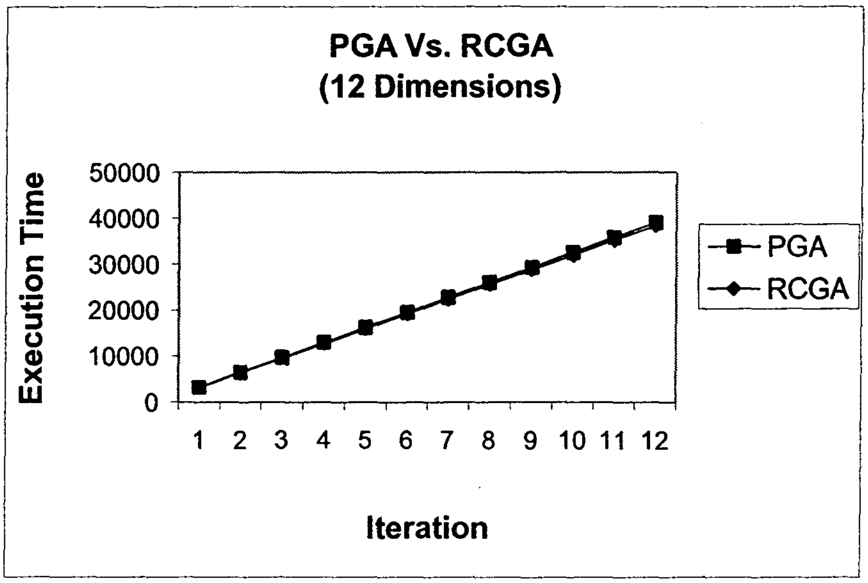


Figure 52: PGA Vs. RCGA: Execution Time Vs. Iteration: 12 Dimensions

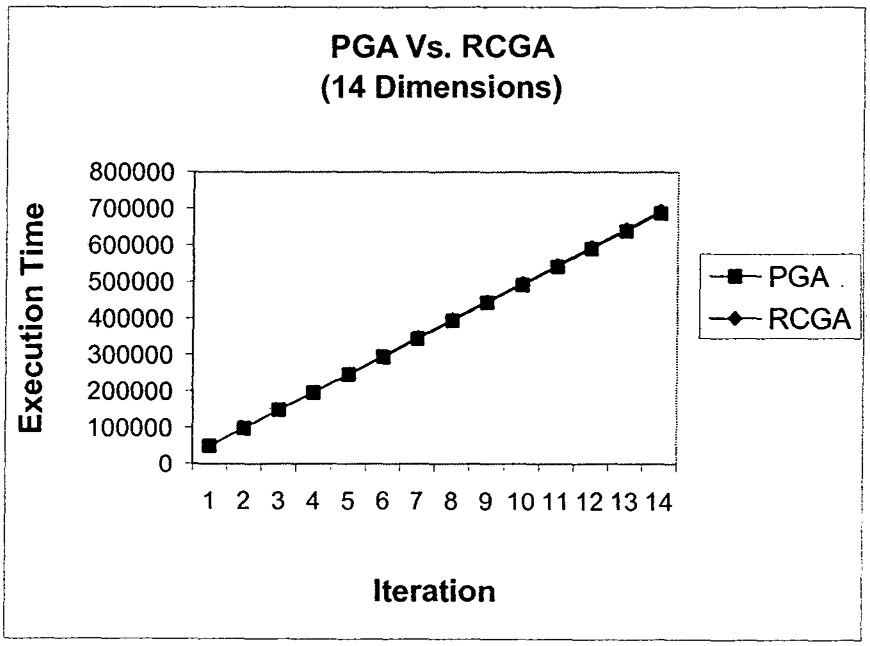


Figure 53: PGA Vs. RCGA: Execution Time Vs. Iteration: 14 Dimensions

In order to compare the benefit values of view selected by the two algorithms RCGA and PGA, a graph was plotted between benefit value and the number of dimensions. The graph is shown in Figure 54.

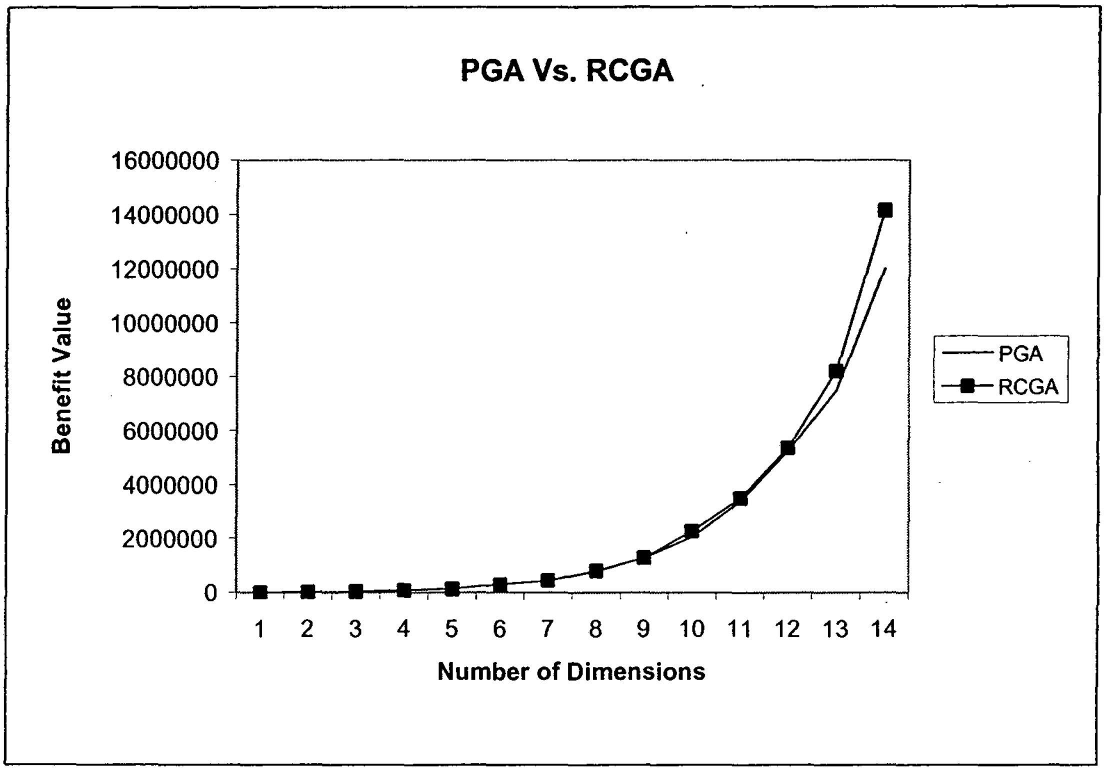


Figure 54: PGA Vs. RCGA: Benefit Value Vs. Number of Dimensions

As per the graph, there is very little difference between the benefit values of the view selected by the two algorithms for lower dimensions. However, for higher dimensions the algorithm RCGA holds a slight advantage over the algorithm PGA. In order to understand this difference better, the benefit values versus iteration graph were plotted for the dimensions 8, 10, 12 and 14 for the two algorithms. These graphs are shown in Figures 55 – 58.

The graphs show that for initial iterations both algorithms have a similar benefit value. However, in the later iterations, the RCGA shows a slightly higher benefit value as compared to PGA.

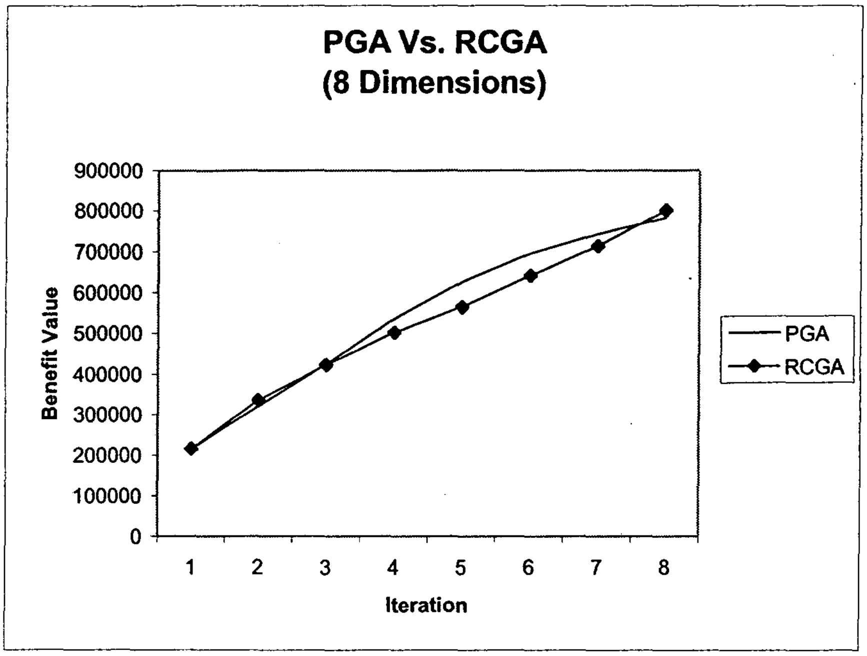


Figure 55: PGA Vs. RCGA: Benefit Value Vs. Iteration: 8 Dimensions

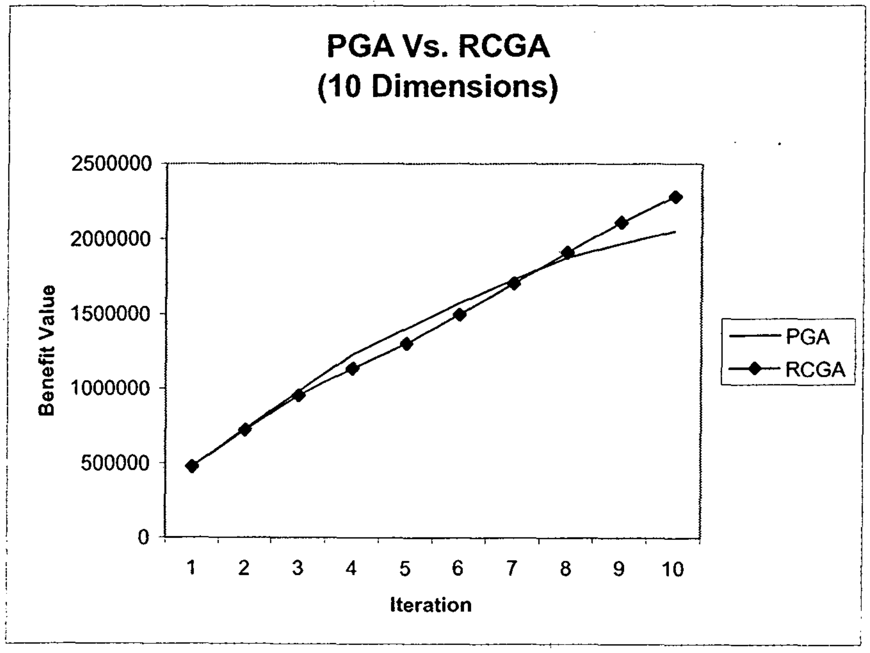


Figure 56: PGA Vs. RCGA: Benefit Value Vs. Iteration: 10 Dimensions

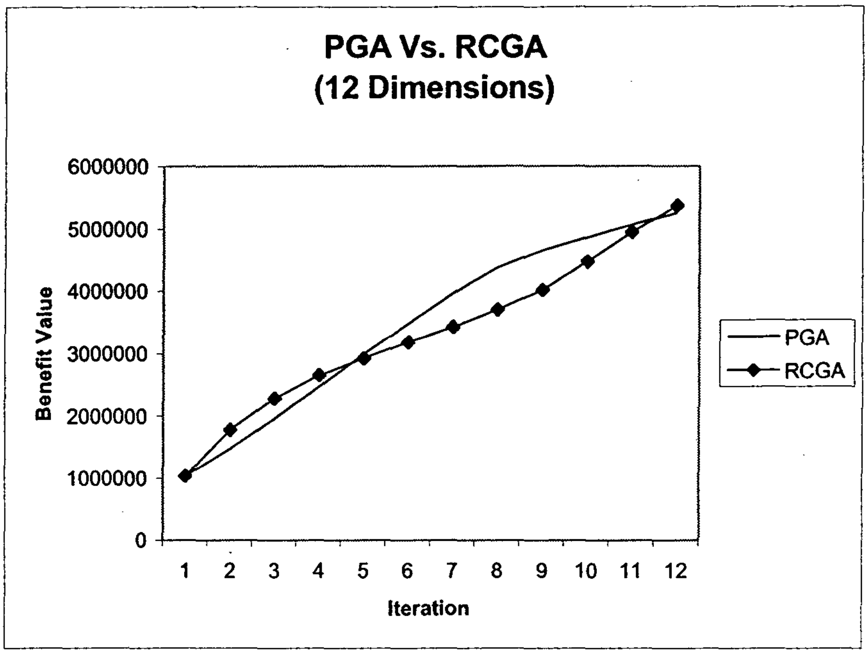


Figure 57: PGA Vs. RCGA: Benefit Value Vs. Iteration: 12 Dimensions

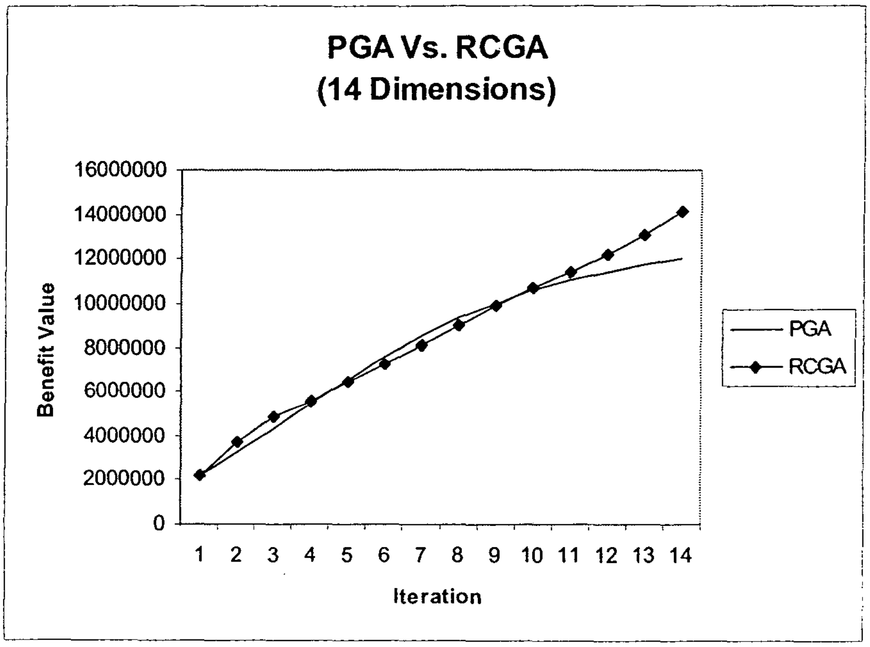


Figure 58: PGA Vs. RCGA: Benefit Value Vs. Iteration: 14 Dimensions

CHAPTER 3

CONCLUSIONS

This dissertation work has focused on the view selection issue of view materialization. A literature survey on algorithms for view selection has been carried out with special emphasis on greedy based algorithms. Some issues related to most fundamental greedy based algorithm HRU have been identified and addressed by the proposed algorithms RLGA and RCGA. The following have been accomplished as part of the work:

- The existing HRU and PGA algorithms have been discussed and compared on performance. The comparison shows that PGA is more scalable with respect to dimensions, as compared to HRU. However, the views selected for materialization using PGA may not be as beneficial as those selected by HRU.
- One reason for the high run time complexity of HRU algorithm has been found to be the frequent re-computations of benefit values after every selection of view for materialization. An algorithm RLGA has been proposed that addresses this issue by reducing the number of re-computations with every selection of beneficial view. The algorithm RLGA uses a heuristic based criteria to reduce the number of dependencies

among views. This results in a reduced lattice, with respect to the dependencies, from which beneficial views are selected greedily. The proposed algorithm RLGA has been compared with algorithm HRU on the parameters like the number of re-computations and the execution time. The results show that the algorithm RLGA requires a lower number of re-computations as compared to algorithm HRU, resulting in an improved execution time.

Another issue with algorithm HRU is that the number of possible views is exponential with respect to the number of dimensions, making it infeasible for high dimensional data sets as its execution time becomes phenomenally high. An algorithm RCGA, on the lines of existing algorithm PGA, has been proposed that improves the scalability with respect to number of dimensions. The algorithm RCGA recommends views, using the heuristic used in RLGA, followed by greedily selecting the most beneficial view out of the recommended views. The performance of algorithm RCGA has been compared with algorithms HRU and PGA on parameters like the execution time and the benefit value. The results show that the algorithm RCGA is more scalable as compared to the algorithm HRU with a much improved execution time. However, the benefit value of views selected by RCGA is found to be slightly lower than those selected by HRU. Further, the comparison with algorithm PGA shows that both algorithms have similar execution time with respect to number of dimensions. Though algorithm RCGA has a slight edge in terms of benefit value for higher dimensional data sets.

REFERENCES

[ACN00] Agrawal,S., Chaudhuri, S. and Narasayya, V.: Automated Selection of Materialized Views and Indexes for SQL Databases, 26th VLDB Conference, Cairo, Egypt; 2000

[AD98] Abiteboul, S. And Duschka, O.M.: Complexity of Answering Queries Using Materialized Views, ACM PODS, Seattle, WA, USA; 1998

[BB97] Brassard, G. and Bratley, P.: Fundamentals of Algorithms, Prentice Hall of India; 1997

[BLT86] Blakeley, J. A., Larson, P. and Tompa, F.W.: Efficiently Updating Materialized Views, ACM SIGMOD Management of Data, pp. 61-71, Washington DC; May 1986

[BPT97] Baralis, E., Paraboshi, S. and Teniente, E.: Materialized view selection in a multidimensional database. 23rd VLDB Conference, pp. 156-165, 1997

[CD01] Ciferri, C.D de A and deSouza, F. da F.: Materialized Views in Data Warehousing Environments, IEEE International Conference of the Chilean Computer Science Society (SCCC'01), p. 0003; 2001

[CFP99] Ceri, S., Fraternali, P. and Paraboschi, S.: Data-Driven One-to-One Web Site Generation for Data-Intensive Applications, 25th VLDB Conference, Scotland; 1999

[CHS01] Chirkova, R., Halevy, A. Y., and Suciu,D.: A formal perspective on the view selection problem, 27 VLDB Conference, Italy; 2001

[CLF01] Chan, G.K.Y. and Li, Q. and Feng, L.: Optimized Design of Materialized Views in a Real-Life Data Warehousing Environment, International Journal of Information Technology, 7 (1). pp. 30-54; 2001

[CP03] Carrano, F.M. and Prichard, J.J.: Data Abstraction and Problem Solving with Java, Walls and Mirrors, Addison Wesley; 2003

[EM07] Encinas, M.T.S. and Montano, J.A.H.: Alorithms for selection of materialized views: based on a cost model, Eighth Mexican International Conference on Current Trends in Computer Science, IEEE; 2007

[G97] Gupta, H.: Selection of Views to Materialize in a Data Warehouse, ICDT '97, Delphi, Greece, Springer, pp. 98-112; January 1997

[GHRU97] Gupta, H., Harinarayan, V., Rajaraman, A., and Ullman, J.D.: Index Selection for OLAP, 13th ICDE Conference, pp. 208-219; April 1997

[GM99] Gupta, H. and Mumick, I. S.: Selection of Views to Materialize Under a Maintenance Cost Constraint, 7th Intl. Conf. on Database Theory, pp. 453-470; 1999

[GM05] Gupta, H. and Mumick, I. S.: Selection of Views to Materialize in a Data Warehouse, IEEE; 2005

[GYCL03] Gou, G., Yu, J.X., Choi, C.H. and Lu, H: An Efficient and Interactive A*-Algorithm with Pruning Power- Materialized View Selection Revisited, Eighth International DASFAA'03 Conference; 2003

[H87] Hanson, E.R.: A Performance Analysis of View Materialization Strategies, ACM SIGMOD Management of Data, pp. 440-453; May 1987

[HCLK99] Horng, J.T., Chang, Y.J., Liu, B.J. and Kao, C.Y.: Materialized View Selection Using Genetic Algorithms in a Data Warehouse System, IEEE CEC; July 1999

[HCL03] Horng, J.T., Chang, Y.J. and Liu, B.J.: Applying Evolutionary Algorithms o Materialized View Selection in a Data Warehouse, Springer-Verlag Soft Computing; 2003

[HLJ06] Habich, D., Lehner, W. and Just, M.: Materialized Views in the Presence of Reporting Functions, 18th SSDBMB'06 Conference; 2003

[HRU96] Harinarayan, V., Rajaraman, A. and Ullman, J: "Implementing Data Cubes Efficiently," Proc. ACM SIGMOD Int'l Conf. Management of Data, 1996

[I00] Indulska, M., Shared Result Identification for Materialized View Selection, 11th Australasian Database Conference, ADC; 2000

[I03] Inmon, W.H,: Building the Data Warehouse, Third Edition, Wiley Dreamtech; 2003

[KR01] Kotidis, Y. and Roussopoulos, N.: A Case of Dynamic View Management, ACM Transactions on Database Systems, pp. 388-423; 2001

[KSCP04] Karenos, K., Samaras, G., Chrysanthis, P. K. and Pitoura, E.: Mobile Agent-Based Services for View Materialization, ACM SIGMOBILE Mobile Computing and Communications Review, Volume 8, Number 3; 2004

[L06] Lawrence, M.: Multiobjective Genetic Algorithms for Materialized View Selection in OLAP Data Warehouses, ACM GECCO'06, US; July 2006

[L07] Luo, G.: Partial Materialized Views, Int. Conf. on Data Engineering (ICDE'07), Istanbul, Turkey; Apr. 2007

[LB06] Loureiro, J and Belo, O.: An Evolutionary Approach to the Selection and Allocation of Distributed Cubes, IEEE IDEAS'06; 2006

[LR99] Labrinidis, A. and Roussopoulos, N.: On the Materialization of Web Views, ACM SIGMOD Workshop on the Web and Databases (WebDB'99), Philadelphia, USA; June 1999

[LR00] Labrinidis, A. and Roussopoulos, N.: Web View Materialization, ACM SIGMOD Conference, Dallas, Texas, USA; May 2000

[LR01a] Labrinidis, A. and Roussopoulos, N.: Adaptive Web View Materialization ACM SIGMOD Workshop on the Web and Databases (WebDB'2001), California, USA; June 2001

[LR01b] Labrinidis, A. and Roussopoulos, N.: Update Propagation Strategies for Improving the Quality of Data on the Web, 27th VLDB Conference, Rome, Italy; September 2001

[LR02] Labrinidis, A. and Roussopoulos, N.: Online View Selection for the Web, University of Maryland, Computer Science Department, Technical Report, CS-TR-4343; 2002

[LR04] Labrinidis, A. and Roussopoulos, N.: Exploring the tradeoff between performance and data freshness in database-driven Web servers, VLDB Journal; Aug 2004

[LWL06] Li, J., Wang, Y. and Liu, R.: Selection of Materialized View Based on Information Weight and Using Huffman-Tree on Spatial Data Warehouse, IEEE ICICIC, pp. 71-74; 2006

[LYSW07] Lin, Z., Yang, D., Song, G. and Wang, T.: User-oriented Materialized View Selection, 7th IEEE International Conference on Computer and Information Technology; 2007

[MMM02] Meccan, G., Mendelzon, A.O. and Merialdo, P.: Efficient Queries over Web Views, IEEE Transactions on Knowledge and Data Engineering, vol. 14, no.6; Nov/ Dec 2002

[MSRK99] Mohania, M., Samtani, S., Roddick, J, and Kambayshi, Y.: Advances and Research Directions in Data Warehousing Technology, Australian Journal of Information Systems, Vol 7, No 1; 1999

[NT02] Nadeua, T.P., Teorey, T.J.: Achieving Scalability in OLAP Materialized View Selection, DOLAP '02, pp. 28–34, ACM; 2002

[R97] Roussopoulos, N.: Materialized Views and Data Warehouse, 4th Workshop KRDB-97, Athens, Greece; August 1997

[RK86] Roussopoulos, N and Kang, H.: Principles and Techniques in the Design of ADMS+/-, COMPUTER; December 1986

[RSS96] Ross, K. A., Srivastava, D. and Sudarshan, S.: Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time, ACM SIGMOD International Conference on Management of Data, pp. 447-458, Montreal; June 1996

[SF89] Segev, A. And Park, J.: Updating Distributed Materialized Views, IEEE Transactions on Knowledge and Data Engineering; June 1989

[SF90] Segev, A. and Fang, W.: Currency-Based Updated to Distributed Materialized Views, IEEE; 1990

[SKRP01] Schrefl, M.; Kapsammer, E.; Retschitzegger, W.; Proll, B.: Self-Maintaining Web Pages - An Overview, 12th Australasian Database Conference (ADC); 2001

[SP89a] Segev, A. and Park, J.: Maintaining Materialized Views in Distributed Databases, IEEE; 1898

[SP89b] Segev, A. and Park, J.: Updating Distributed Materialized Views, IEEE Transaction on Knowledge and Data Engineering, vol. 1, no.2, pp. 173 -184; June 1989

[SRR06] Shah, B., Ramachandaran, K and Raghavan, V.: A Hybrid Approach for Data Warehouse View Selection, Intl. Journal of Data Warehousing and Mining; 2006

[SSA07] Saidi, S., Slimani, Y. and Arour, K.: WebView Selection from User Access Patterns, PIKM'07, Lisboa, Portugal; November 2007

[TU] Teschke, M. and Ulbrich, A.: Using Materialized Views to Speed Up Data Warehousing

[TB00] Theodoratos, D., Bouzeghoub, M.: A General Framework for the View Selection Problem for Data Warehouse Design and Evolution, 3rd ACM Intl. Workshop on Data Warehousing and On-Line Analytical Processing (DOLAP'00), Washington, DC., U.S.A., ACM Press, pp. 1-8; Nov. 2000

[URT99] Uchiyama, H., Runapongsa, K. and Teorey, T.J.: A Progressive View Materialization Algorithm, ACM DOLAP, Kansas city, USA; 1999

[VGP07] Velinov, G., Gligoroski, D. and Popovska, M.K.: Hybrid Greedy and Genetic Algorithms for Optimization of Relational Data Warehouses, 25th IASTED International Multi-Conference, AI and Applications, Austria; Feb. 2007

[VVK02] Valluri, .R., Vadapalli, S. and Karlapalem, K.: View Relevance Driven Materialized View Selection in Data Warehousing Environment, 13th Autralasian Database Conference (ADC2002), Melbourne, Autralia; 2002

[WW99] Weber, J.L. and Wutka, M.: Using Java 2 Platform; Que Corp.; 1999

[YKL97] Yang, J., Karlapalem, K. and Li, Q.: Algorithms for Materialized View Design in Data Warehousing Environment, 23rd VLDB, pp. 136-145; 1997

[YYL07] Yin, G., Yu, X and Lin, L.: Strategy of Selecting Materialized Views Based on Cache-updating,: IEEE International Conference on Integration Technology, ICIT '07; 2007

[YAE05] Yousri, N.A.R., Ahmed, K.M. and El-Makky, N.M.: Algorithms for Selecting Materialized Views in a Data Warehouse, IEEE 2005 International Conference on Computer Systems and Applications; 2005

[YFIV00] Yagoub, K., Florescu, D., Issarny, V. and Valduriez, P.: Caching Strategies for Data-Intensive Web Sites, 26th VLDB Conference, Cairo, Egypt; 2000

[ZLGD07] Zhou, J., Larson, P., Goldstein, J. and Ding, L.: Dynamic Materialized Views, IEEE 23rd International Conference on Data Engineering, ICDE; 2007

[ZLE07] Zhou, J., Larson, P. and Elmongui, H.G.: Lazy Maintenance of Materialized Views, ACM VLDB'07, Autria; Sep. 2007

[ZYY99] Zhang, C., Yao, X. And Yang, J.: Evolving Materialized Views in a Data Warehouse, IEEE CEC; July 1999