INVESTIGATION OF EVOLUTIONARY MULTI-OBJECTIVE ALGORITHMS IN SOLVING VIEW SELECTION PROBLEM

SEYED HAMID TALEBIAN

THESIS SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR
2013
Name of Candidate:  Seyed Hamid Talebian  (I.C/Passport No:  R10764969)

Registration/Matric No: WHA070020

Name of Degree: Doctor of Philosophy

Investigation of Evolutionary Multi-Objective Algorithms in Solving View Selection Problem

Field of Study:

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya (“UM”), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate’s Signature               Date:

Subscribed and solemnly declared before,

Witness’s Signature               Date:

Name: 
Designation:
Abstract

Today’s huge volumes of data are maintained in conventional database systems. The data distributed in these systems sometimes come in inconsistent formats. On the other hand, data analysts require an environment so that they are able to obtain the required information. However, the distributed and heterogeneous structure of these systems prevents them from taking advantage of these data for analytical purposes.

In order to overcome this weakness the data warehouse concept was introduced (Inmon, 1992). The main idea is such that, incompatible data spread over heterogeneous systems are extracted and after transformations to a unified form are loaded to a central and separate database for analytical purposes.

Since analytical queries are complex and take a long time to be processed under normal circumstance, there is a need for a strategy to improve the speed of such queries. One of the ways for resolving this problem is by using pre-computed results of queries. In this approach, results of possible queries are computed in advance and whenever a user submits a query, instead of referring to the main table with enormous numbers of records, a proper pre-computed result is fetched and used for answering the query.

The results of each query can be a logical table which is derived from the base tables. Such tables are called views in database terminology. Once the records of a view are stored on disk, the view is called a materialized view.

Another important issue resulting from materializing view is updating the views. If during periodical reload from the conventional database systems new records are inserted to the fact table, the views that have been derived from the fact table need to be updated. The process of updating views in response to changes to base tables is called view update or view maintenance (Kotidis, 2002). This process is expensive because it
is time consuming. In today’s systems availability is one goal and this is achieved by minimizing the update window during which the system is down.

Although using materialized views for answering queries reduces the query response time but at the same time it increases the view update time. Selecting a subset of views which gives the best compromise between minimizing query response time and minimizing total update time is known as the view selection problem. This is considered a multi-objective problem because the problem involves optimizing more than one problem simultaneously subject to constraint(s).

Evolutionary multi-objective algorithms are considered as good candidate for solving multi-objective optimization problems and have been applied to variety of problems in different areas.

In this research, we showed how evolutionary multi-objective algorithms can be used to solve the view selection problem and its advantage over classical optimization problems were described. As a comparative study, the performance of the algorithms was evaluated based on various standard metrics. In addition to the normal metrics, the computational time for executing each algorithm was also measured and compared.

Our results show that algorithms which use elitism feature are superior to other algorithms in most of the metrics. At the same time implementing elitism feature increases the computational complexity of the algorithm. Furthermore, niching strategies in some algorithms play an important role in delivering a diverse set of solutions.

Generally, it can be said that two algorithms SPEA-II and NSGA-II perform better than other algorithms in terms of convergence to the optimal solution and diversity.
Acknowledgment

This thesis would not be completed without the continuous support of my supervisor, Associative Professor Dr. Sameem Abdul Kareem. Her encouragement was invaluable and her tolerance during period of research was remarkable. I also would like to thank University of Malaya for support for this research.
# Table of Contents

## Chapter 1. Introduction ........................................................................................................ 1

1.1 Background ......................................................................................................................... 1

1.2 Motivation ............................................................................................................................. 3

1.3 Problem Statement .............................................................................................................. 3

1.3.1 Total Query Response Time .......................................................................................... 4

1.3.2 Update Time .................................................................................................................... 4

1.3.3 Total Disk Space .............................................................................................................. 5

1.3.4 View Selection Problem .................................................................................................. 5

1.3.5 Multi-Objective View Selection Problem ......................................................................... 6

1.4 Evolutionary Multi-Objective Optimization ......................................................................... 6

1.5 Research Objectives ............................................................................................................ 10

1.6 Research Scope .................................................................................................................... 10

1.7 Research Methodology ...................................................................................................... 11

1.8 Thesis Outline ..................................................................................................................... 13

## Chapter 2. Materialized View Selection ................................................................................. 15

2.1 Background .......................................................................................................................... 15

2.2 The Data Warehouse Architecture ..................................................................................... 17

2.3 Terminology ........................................................................................................................ 20

2.4 Dimensional Modeling ....................................................................................................... 21

2.5 The Star Schema ................................................................................................................. 23

2.6 Dimension Hierarchies ....................................................................................................... 25

2.7 OLAP Query format .............................................................................................................. 26
3.3.2 Elitism .................................................................................................................. 74
3.3.3 Non-Elitist Algorithms ....................................................................................... 75
  3.3.3.1 Weight Based Genetic Algorithm................................................................. 75
  3.3.3.2 Vector Evaluated Genetic Algorithm............................................................ 76
  3.3.3.3 Non-dominated Sorting Genetic Algorithm.................................................... 77
  3.3.3.4 Niched Pareto Genetic Algorithm................................................................ 78
  3.3.3.5 Multiple Objective Genetic Algorithm......................................................... 79
3.3.4 Elitist Algorithms .................................................................................................. 80
  3.3.4.1 Non-dominated Sorting Genetic Algorithm II............................................... 81
  3.3.4.2 Strength Pareto Evolutionary Algorithm....................................................... 82
  3.3.4.3 Strength Pareto Evolutionary Algorithm II.................................................... 84
3.3.5 Constraint Handling ............................................................................................ 87
3.3.6 Applications of Evolutionary Multi-Objective Algorithms in Other Areas........ 88
3.4 Performance Metrics (indicators) ........................................................................ 89
  3.4.1 Set Coverage (C) .............................................................................................. 91
  3.4.2 Spacing (SP) .................................................................................................... 92
  3.4.3 Maximum Spread (MS) .................................................................................. 93
  3.4.4 Hypervolume (HV) .......................................................................................... 95
3.5 Summary .................................................................................................................. 96

Chapter 4. Methodology ............................................................................................... 97
4.1 Introduction .............................................................................................................. 97
4.2 Object Oriented Architecture ............................................................................... 98
  4.2.1 Objects in Problem Domain .......................................................................... 99
  4.2.2 Objects in Method Domain ............................................................................ 104
    4.2.2.1 Core Objects ............................................................................................. 105
    4.2.2.2 Shell Objects ............................................................................................. 113
  4.2.3 Performance Evaluation .................................................................................. 114
4.3 Parameter Setting ................................................................................................... 114
4.4 Performance Metrics Used .................................................................................... 117

List of Figures

Figure 1.1 Basic Evolutionary Algorithm’s Flowchart ............................................................. 8
Figure 1.2 Different Phases of Research Methodology ............................................................. 13
Figure 2.1 Data Warehouse Architecture .................................................................................. 18
Figure 2.2 Example of Normalized Entity Relationship Model ................................................. 23
Figure 2.3 Star-Like Modeling of Data Warehouse ..................................................................... 24
Figure 2.4 Star Schema for Sales System .................................................................................. 24
Figure 2.5 Fact Table and Dimensional Tables .......................................................................... 25
Figure 2.6 Dimension Hierarchies for Sales System ................................................................. 26
Figure 2.7 Roll Up And Drill Down Queries ............................................................................. 26
Figure 2.8 Picking Up a Hierarchy Level from Each Dimension Hierarchy to Form a Group-By Query .................................................................................................................. 27
Figure 2.9 A Dependency Lattice Organized From All Possible Views ...................................... 29
Figure 2.10 Dependency Lattice Organized From All Possible Group by Queries ................. 29
Figure 2.11 Dependency Lattice for Sales System .................................................................... 30
Figure 2.12 A Sample Dependency Lattice with a Current Set of Materialized Views .......... 32
Figure 2.13 Full, Partial and No Materialization in a 2D space ................................................ 35
Figure 2.14 Selecting the Best Trade-Off View Selection Problem Solution ........................... 35
Figure 2.15 Benefit Calculation ............................................................................................... 38
Figure 3.1 Multi-Objective and Conflictive Optimization Problem ......................................... 54
Figure 3.2 Evolutionary Algorithms Branches .......................................................................... 56
Figure 3.3 Decision Variable Space Versus Objective Function Space .................................... 58
Figure 3.4 A Set of 7 Solutions in Objective Space .................................................................. 60
Figure 3.5 Classification of Non-Dominated and Dominated Set for Example in Figure 3.4 ........................................................................................................................................ 62
Figure 3.6 Pareto Front Curves for Two Different Search Spaces ........................................62
Figure 3.7 Non-Dominated Sorting For Solution Set of Figure 3.4 ......................................63
Figure 3.8 Evolution Cycle for Genetic Algorithm .................................................................65
Figure 3.9 Real World Space versus Genetic Space ...............................................................66
Figure 3.10 A Sample Representation ......................................................................................67
Figure 3.11 A sample random initial population with 10 members ........................................68
Figure 3.12 Genetic Operators: Selection, Crossover and Mutation .......................................70
Figure 3.13 Single point crossover ..........................................................................................71
Figure 3.14 Two Points Crossover ..........................................................................................71
Figure 3.15 Mutation Operator ...............................................................................................72
Figure 3.16 Schematic Procedure of VEGA ..........................................................................77
Figure 3.17 NPGA Selection Mechanism ..............................................................................79
Figure 3.18 Crowding Distance Calculation ..........................................................................82
Figure 3.19 Schematic of NSGA-II procedure ......................................................................82
Figure 3.20 SPEA Fitness Assignment For a Set of Solutions ................................................84
Figure 3.21 SPEA-II Strength And Raw Fitness for a Set of Solutions .....................................85
Figure 3.22 Search Space, Feasible and Infeasible Regions ....................................................87
Figure 3.23 Convergence and Diversity .................................................................................90
Figure 3.24 Ideal Value for Coverage Metric ........................................................................91
Figure 3.25 Distances between Neighboring Solutions in Set of Obtained Solutions.................92
Figure 3.26 Ideal Value for Spacing Metric ..........................................................................93
Figure 3.27 Maximum Spread for A Set of Solutions ............................................................94
Figure 3.28 Ideal Value for Maximum Spread Metric .............................................................94
Figure 3.29 Hypervolume for a Set of Non-Dominated Solutions ...........................................95
Figure 3.30 Ideal Value For Hypervolume Metric .................................................................96
Figure 4.1 Classifications of Methods and Problem ..................................................................97
Figure 4.2 The UML Class Diagram ................................................................. 99
Figure 4.3 Hierarchy Defined Within Each Dimension Table ......................... 101
Figure 4.4 View Dependency Lattice Calculated Based On Hierarchies in Figure 4.2 101
Figure 4.5 The Lattice Object ........................................................................... 102
Figure 4.6 Core And Shell Objects .................................................................... 105
Figure 4.7 Inheritance In Method Domain ........................................................... 106
Figure 4.8 Evolutionary Multi-Objective Optimization Interface ....................... 114
Figure 4.9 Calculation of distance in 2D objective space .................................... 115
Figure 4.10 Defined Reference Point for Hypervolume Metric ......................... 116
Figure 4.11 View Selection Problem Encoding .................................................. 119
Figure 4.12 Calculating the Size of Search Space .............................................. 119
Figure 4.13 Screenshot of the Tool for Defining the View Selection Problem Instance ........................................................................................................ 123
Figure 4.14 Creating VSP Instance ..................................................................... 123
Figure 4.15 The Star Schema for the Supplier-Parts Database ............................ 123
Figure 4.16 Dimension Hierarchies for VSP₁ ..................................................... 125
Figure 4.17 Dependency Lattice for VSP₁ .......................................................... 127
Figure 4.18 Dimension Hierarchies for VSP₂ ..................................................... 128
Figure 4.19 Dependency Lattice for VSP₂ .......................................................... 129
Figure 5.1 A sample box plot ............................................................................. 133
Figure 5.2 Non-Dominated Front Obtained By Each Evolutionary Algorithm Solving VSP₁ ........................................................... 142
Figure 5.3 Non-Dominated Front Obtained By Each Evolutionary Algorithm Solving VSP₂ ........................................................... 143
List of Tables

Table 2.1 Differences Between OLTP and OLAP Systems ........................................... 16
Table 2.2 Single Objective View Selection Problem Variations ..................................... 37
Table 2.3 Different Works for Single Objective View Selection Problem ...................... 48
Table 2.4 Different Works for Solving Single Objective View Selection and Their
    Performance ............................................................................................................. 49
Table 3.1 List of some other Applications of Evolutionary Algorithms ......................... 89
Table 4.1 View Class ..................................................................................................... 100
Table 4.2 Lattice Class .................................................................................................. 102
Table 4.3 VSP Class ...................................................................................................... 103
Table 4.4 VSP Phenotype Class ..................................................................................... 104
Table 4.5 GA Class ....................................................................................................... 106
Table 4.6 Individual Class ............................................................................................. 107
Table 4.7 Assignment of Values for Objectives and Constraint .................................... 108
Table 4.8 Population Class ............................................................................................ 109
Table 4.9 PopulationSet Class ...................................................................................... 111
Table 4.10 Crossover Class ........................................................................................... 111
Table 4.11 Mutation Class ............................................................................................ 112
Table 4.12 Selection Class ............................................................................................ 113
Table 4.13 Evolutionary Multi-Objective Algorithm Interface ...................................... 113
Table 4.14 Performance Evaluation Class ...................................................................... 114
Table 4.15 GA Parameter Setting .................................................................................. 116
Table 4.16 Ideal Values for Metrics Used ..................................................................... 118
Table 4.17 List of Views For VSP$^1$ .............................................................................. 126
Table 4.18 List Of Views for VSP$^2$ .............................................................................. 128
Table 5.1 Checking 5.1 Condition for *SPEA-II* ................................................................. 134
Table 5.2 Mean Values of Two Set Coverage Metric for $VSP_1$ ............................................. 135
Table 5.3 Mean Values of Two Set Coverage Metric for $VSP_2$ ............................................. 135
Table 5.4 Multiple Comparison of *Coverage Metric* for $VSP_1$. ............................................. 135
Table 5.5 Multiple Comparison of *Coverage Metric* for $VSP_2$. ............................................. 135
Table 5.6 Ranking of the Algorithms Based on *Two Set Coverage Metric* and for $VSP_1$ ............................................................................................................................... 136
Table 5.7 Ranking of the Algorithms Based on *Two Set Coverage Metric* With Respect to $VSP_2$ ............................................................................................................................... 137
Table 5.8 Multiple Comparison of *Hypervolume* for $VSP_1$ ....................................................... 137
Table 5.9 Multiple Comparison of *Hypervolume* for $VSP_2$ ....................................................... 137
Table 5.10 Mean and variance values of the *Hypervolume* metric for $VSP_1$ ......................... 138
Table 5.11 Mean and Variance Values of the *Hypervolume* Metric for $VSP_2$ ................. 138
Table 5.12 Multiple Comparison of *Spacing* for $VSP_1$ ............................................................. 139
Table 5.13 Multiple Comparison of *Spacing* for $VSP_2$ ............................................................. 139
Table 5.14 Mean and Variance Values for *Spacing* Metric for $VSP_1$ ............................................ 140
Table 5.15 Mean and Variance Values for *Spacing* Metric for $VSP_2$ ............................................ 140
Table 5.16 Multiple Comparison for *Maximum Spread* Metric for $VSP_1$ .............................. 141
Table 5.17 Multiple Comparison for *Maximum Spread* Metric for $VSP_2$ .............................. 141
Table 5.18 Mean and Variance values of the *Maximum Spread* Metric for $VSP_1$ .............. 141
Table 5.19 Mean and Variance Values of the *Maximum Spread* Metric for $VSP_2$ .............. 141
Table 5.20 Multiple Comparisons for Computational Time Metric for $VSP_1$ ....................... 144
Table 5.21 Multiple Comparisons for Computational Time Metric for $VSP_2$ ....................... 144
Table 5.22 Mean of Computational Time (in Second) for $VSP_1$ ................................................. 145
Table 5.23 Mean of Computational Time (in Second) for $VSP_2$ ................................................. 145
## List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$</td>
<td>The set of all possible queries</td>
</tr>
<tr>
<td>$Q(M)$</td>
<td>Total query response time in the presence of $M$</td>
</tr>
<tr>
<td>$Q(M \cup v)$</td>
<td>Query response time for all queries when view $v$ would be added to the $M$</td>
</tr>
<tr>
<td>$f_q$</td>
<td>The frequency by which query $q$ is issued</td>
</tr>
<tr>
<td>$f_v$</td>
<td>The frequency by which view $v$ is updated</td>
</tr>
<tr>
<td>$t(q,M)$</td>
<td>The query cost for answering a query $q$ using $M$</td>
</tr>
<tr>
<td>$v$</td>
<td>A particular view</td>
</tr>
<tr>
<td>$M$</td>
<td>The set of materialized views</td>
</tr>
<tr>
<td>$U(M)$</td>
<td>The total time required for updating $M$</td>
</tr>
<tr>
<td>$t(v,M)$</td>
<td>The required time of updating the materialized view $v$ in the presence of the set of $M$ materialized views.</td>
</tr>
<tr>
<td>$DS(M)$</td>
<td>The total disk space required for materializing the $M$</td>
</tr>
<tr>
<td>$DS(v)$</td>
<td>Size of the disk space required for storing view $v$</td>
</tr>
<tr>
<td>$DS$</td>
<td>The size of the allocated disk space for saving views</td>
</tr>
<tr>
<td>$h_i$</td>
<td>The number of hierarchy levels in dimension table $i$</td>
</tr>
<tr>
<td>$dt$</td>
<td>The number of dimension tables</td>
</tr>
<tr>
<td>$V$</td>
<td>The set of all possible views</td>
</tr>
<tr>
<td>$</td>
<td>V</td>
</tr>
<tr>
<td>$Q_{\text{max}}$</td>
<td>The maximum query response time</td>
</tr>
<tr>
<td>$Q_{\text{min}}$</td>
<td>The minimum query response time</td>
</tr>
<tr>
<td>$Q(\phi)$</td>
<td>Query response time when no view is materialized</td>
</tr>
<tr>
<td>$Q(V)$</td>
<td>Query response time when all possible views are materialized</td>
</tr>
<tr>
<td>$Q(M)$</td>
<td>The Total query response time after materializing the $M$</td>
</tr>
<tr>
<td>$Q_{\text{interval}}$</td>
<td>History of incoming views within a certain period of time</td>
</tr>
<tr>
<td>$F$</td>
<td>Fact Table</td>
</tr>
<tr>
<td>$</td>
<td>F</td>
</tr>
<tr>
<td>$\text{Ancestors}(v)$</td>
<td>Ancestor views of view $v$ in the lattice</td>
</tr>
<tr>
<td>$U_{\text{min}}$</td>
<td>Minimum view update time</td>
</tr>
<tr>
<td>$U_{\text{max}}$</td>
<td>Maximum view update time</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>The amount of change has been made to a view since last update of fact table</td>
</tr>
<tr>
<td>$</td>
<td>v</td>
</tr>
<tr>
<td>$</td>
<td>v</td>
</tr>
<tr>
<td>$</td>
<td>D_i</td>
</tr>
<tr>
<td>$D_i$</td>
<td>Dimension table $i$</td>
</tr>
<tr>
<td>$B(v,M)$</td>
<td>Query Benefit of view $v$ in presence of $M$. Defined as decrease in the Query response time caused by adding view $v$, $M$</td>
</tr>
<tr>
<td>$S$</td>
<td>A set of solutions</td>
</tr>
<tr>
<td>$N$</td>
<td>A non-dominated set of solutions</td>
</tr>
</tbody>
</table>
$w_i$ The weight coefficient for objective function $i$
$f_i(x)$ Objective function $i$
$h_i(x)$ equality constraints
$g_i(x)$ inequality constraints
$p$ The number of equality constraints
$m$ The number of inequality constraints
$k$ The number of objective functions
$P_c$ Crossover probability
$P_m$ Mutation probability
$F(j)$ Fitness value of individual $i$
$S(i)$ Strength of individual $i$
$\overline{popsize}$ The size of external population
$\overline{popsize}$ Population Size
$P_t$ The main population at generation $t$
$\overline{P}_t$ The archive population at generation $t$
$V(x)$ The overall violation of solution $x$
$D(i)$ Density for individual $i$
$v_i(x)$ The violation of solution $x$ with respect to constraint $i^{th}$
$P(x)$ The penalty function for solution $x$
$C(A,B)$ Coverage of set $A$ with respect to $B$
$t_{dom}$ The number of individuals in the comparison set for NPGA
$|\mathcal{Q}|$ A set of solutions
$\sigma_{share}$ the niche radius
$B(M)$ Query benefit which is yielded after materializing all views in $M$. 
### List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>International Business Machines</td>
</tr>
<tr>
<td>Mac</td>
<td>Macintosh</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>OLTP</td>
<td>Online Transaction Processing</td>
</tr>
<tr>
<td>OLAP</td>
<td>Online Analytical Processing</td>
</tr>
<tr>
<td>WBGA</td>
<td>Weight-Based Genetic Algorithm</td>
</tr>
<tr>
<td>VEGA</td>
<td>Vector Evaluated Genetic Algorithm</td>
</tr>
<tr>
<td>NSGA</td>
<td>Non-Dominated Sorting Genetic Algorithm</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>Non-Dominated Sorting Genetic Algorithm-II</td>
</tr>
<tr>
<td>SPEA</td>
<td>Strength Pareto Evolutionary Algorithm</td>
</tr>
<tr>
<td>SPEA-II</td>
<td>Strength Pareto Evolutionary Algorithm-II</td>
</tr>
<tr>
<td>MOGA</td>
<td>Multiple Objective Genetic Algorithm</td>
</tr>
<tr>
<td>NPGA</td>
<td>Niched Pareto Genetic Algorithm</td>
</tr>
<tr>
<td>E-R</td>
<td>Entity Relationship</td>
</tr>
<tr>
<td>LRU</td>
<td>Least Recently Used</td>
</tr>
<tr>
<td>PBS</td>
<td>Pick By Size</td>
</tr>
<tr>
<td>A*</td>
<td>A Star</td>
</tr>
<tr>
<td>BPUS</td>
<td>Benefit Per Unit Space</td>
</tr>
<tr>
<td>PGA</td>
<td>Polynomial Genetic Algorithm</td>
</tr>
<tr>
<td>SA</td>
<td>Simulated Annealing</td>
</tr>
<tr>
<td>II</td>
<td>Iterative Improvement</td>
</tr>
<tr>
<td>MA</td>
<td>Memetic Algorithm</td>
</tr>
<tr>
<td>IP</td>
<td>Integer Programming</td>
</tr>
<tr>
<td>RLGA</td>
<td>Reduced Lattice Greedy Algorithm</td>
</tr>
<tr>
<td>GLS</td>
<td>Genetic Local Search</td>
</tr>
<tr>
<td>MOEA</td>
<td>Multi-Objective Evolutionary Algorithm</td>
</tr>
<tr>
<td>HLGA</td>
<td>Hajela &amp; Lin Genetic Algorithm</td>
</tr>
<tr>
<td>MS</td>
<td>Maximum Spread</td>
</tr>
<tr>
<td>HV</td>
<td>Hypervolume</td>
</tr>
<tr>
<td>SP</td>
<td>Spacing</td>
</tr>
<tr>
<td>VSP</td>
<td>View Selection Problem</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>EA</td>
<td>Evolutionary Algorithm</td>
</tr>
<tr>
<td>GLList</td>
<td>Gene List</td>
</tr>
<tr>
<td>GD</td>
<td>Generational Distance</td>
</tr>
<tr>
<td>ER</td>
<td>Error Ratio</td>
</tr>
<tr>
<td>VSP₁</td>
<td>View Selection Problem Instance 1</td>
</tr>
<tr>
<td>VSP₂</td>
<td>View Selection Problem Instance 2</td>
</tr>
<tr>
<td>TPC</td>
<td>Transaction Processing Performance Council</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>CSP</td>
<td>Constraint Satisfaction Problem</td>
</tr>
<tr>
<td>NP</td>
<td>Non-Deterministic Polynomial Time</td>
</tr>
<tr>
<td>NP-Hard</td>
<td>Non-Deterministic Polynomial-Time Hard</td>
</tr>
</tbody>
</table>
Chapter 1. Introduction

1.1 Background

Today’s large volume of data is kept in conventional database (or operational) systems. These systems are useful in performing the routine tasks for organizations. On the other hand, business analysts realized the importance of these data. Due to competitive nature of business they are motivated to take advantage of the data for the purpose of decision making. However, the data which reside in conventional database systems are not in a form suitable for analysis for the following reasons: the data is distributed over multiple independent sources; the data is represented in inconsistent formats. The type of software and hardware architecture differs from one source to another. In addition, operational systems or OnLine Transaction Processing (OLTP) systems are designed to efficiently respond to regular queries. The regular queries in operational systems are simple queries accessing small numbers of records. In fact, the goal of OLTP systems is to maximize transaction throughput. In contrast to OLTP queries the analytical queries are complex and involved in aggregating large number of records which are accumulated over several years. In order to overcome those weaknesses the Data Warehouse concept was introduced (Inmon, 1992). Data warehouse is a service by which incompatible and heterogeneous data are extracted from different operational sources, transformed into a unified form and then loaded into a central and huge repository. The aim of Data Warehouse is to easily and efficiently support decision making and business analysis (Agrawal, 2005; Ponniah, 2001).

As mentioned the analytical queries are complex queries and therefore take a considerably long time to be answered under normal circumstance. For example, a
query that asks for the sum of sales of a particular product category in a particular country in the last decades may result in the calculation of millions of records. Moreover, the executives of the organization require the result of analytical queries in a short time so that they are able to make fast and proper decisions. As a result, improving the performance of queries in a data warehouse environment is an essential need. One of the common techniques for speeding up queries is utilizing pre-computed result. The results are called *materialized views* and calculated in advance to submitting queries by end users. In the relational database terminology *view* is a derived table computed from a set of base tables on the fly (Teorey, Lightstone, Nadeau, & Jagadish, 2005). In the materialized view approach, the query response time is saved up since instead of calculating the complex queries through a large number of records the ready result is returned in significantly shorter time. For example, a query which may be processed in one day under the normal situation can be answered less than one hour through materialized views.

The ideal choice would be pre-computing the results for all the possible queries and storing them on disk (which is called materializing) so that each possible query can be responded quickly. However, that is not a practical choice. Materializing all possible views requires a great size of available disk space which is not supported by some computer systems (Kumar & Ghoshal, 2009). Moreover, loading new data from operational sources causes changes to the base table from which the views are already computed. In order to avoid inconsistency between views and base table the views need to be re-calculated. The task of computing the changes made to the base table and applying them to the previously saved views is called *view update* or *view maintenance*. The update window interferes with the working time of the system. Therefore, in order to increase availability, minimizing the update time is desired. Materializing all the possible views causes the update time to increase because the update process takes a
long time since all the views may need to be updated. Therefore because of the mentioned limitations we are forced to select only a subset of view to be saved on disk. Selecting the right set of materialized views is a crucial decision in designing a data warehouse (Shah, Ramachandran, & Raghavan, 2006).

1.2 Motivation

The problem of selecting the right views to be stored on disk space is well studied in the single objective form (see Table 2.3). Several methods like greedy, genetic algorithm, simulated annealing have been applied to solve different variations of the problem. All of these variations share a common characteristic which is the consideration of only one objective function. The objective can be minimizing the query response time, update time or even the weighted sum of these two primitive objectives. However, in the real-world, the data warehouse designer might be interested in minimizing two problem objectives simultaneously. This variation of the problem is called the multi-objective materialized view selection problem (Dhote & Ali, 2009).

So far, few studies (Lawrence, 2006) have addressed the multi-objective view selection problem. In this research, we investigate the materialized view selection problem especially pertaining to the multi-objective variation of the problem. We study the application of evolutionary multi-objective optimization algorithms in solving the two objectives view selection problem.

1.3 Problem Statement

This research is about finding a set of solutions which gives a good trade-off between two conflicting goals, that is: minimizing the total query response time and minimizing the total update time subject to the constraints of available hard disk space for saving the views. The following sub-topics discuss the problem statement in greater details and
describe the objectives and constraints of the problem. The research problem is formally stated in Section 1.3.5.

1.3.1 Total Query Response Time

Having a set of materialized views; $M$, the total time required for evaluating all possible queries in the system; $Q(M)$ can be calculated as below (Lawrence & Rau-Chaplin, 2006):

$$Q(M) = \sum_{q\in Q} f_q \cdot t(q, M) \quad 1.1$$

where $Q$ is set of all queries, $q$ is a particular query, $f_q$ is the frequency of the query $q$. This equation applies, as long as some queries are posed often while some other queries occur rarely. The $t(q, M)$ denotes the time needed for processing the query $q$ in the presence of $M$.

1.3.2 Update Time

Materialized views work like a cache system in the topic of memory management. In a cache system, the frequently accessed data in main memory are duplicated in fast and small capacity storage. Any future request for the data that has already been cached is done through the cache rather than referring to the main memory. This reduces the access time (Silberschatz, 1998).

In the cache technique, whenever the original data in main memory are modified, the cached data needs to be also updated. Similarly, in the materialized view selection, whenever the data warehouse is loaded, and the new records are inserted to the fact table and cause changes, the views which have been derived before from the fact table needs to be updated as well. Normally, in order to reduce disruption to the working window of a data warehouse, the update process is done when the system is idle or at night (Liang, Wang, & Orlowska, 2001; Theodoratos & Bouzeghoub, 2000).
Furthermore, when companies extend their operation hours, the update window becomes shorter.

The total update time for a set $M$, of materialized views can be calculated as below (Lawrence & Rau-Chaplin, 2006):

$$U(M) = \sum_{v \in M} f_v t(v, M)$$  \hspace{1cm} 1.2

where $f_v$ is the frequency of updating view $v$, $t(v, M)$ is the required time of updating the materialized view $v$ in the presence of $M$.

1.3.3 Total Disk Space

The total size of disk space required for saving a subset of views considered for materialization; $DS(M)$ can be calculated as below (Hung, Huang, Yang, & Hsueh, 2007):

$$DS(M) = \sum_{v \in M} DS(v)$$  \hspace{1cm} 1.3

where $DS(v)$ is the size of the disk space required for storing view $v$.

1.3.4 View Selection Problem

The problem of choosing a subset of views to be stored on disk is known as the materialized view selection problem. In general, the view selection problem involves minimizing one or two goal functions possibly subject to one or more constraints. The view selection problem may be considered as a constrained optimization problem (Dhote & Ali, 2009; Gou, Yu, & Lu, 2006) and it has been proven to be an NP-Hard (non-deterministic polynomial-time hard) problem (Gupta, Harinarayan, & Rajaraman, 1997). NP-Hard problems are a class of problems at least as difficult as NP problems. NP Problems, themselves, are a type of problem that is very difficult to solve (Wang, Chang, & Cheng, 2009). According to the literatures (Hanusse, Maabout, & Tofan, 2009; Harinarayan, Rajaraman, & Ullman, 1996; Liang et al., 2001; Lin & Kuo, 2004; Zhou, Wu, & Ge, 2008), there are many variations of the view selection problems and
they have been well studied. All of them can be classified in two main categories; namely, a single objective view selection category or a multi-objective view selection category.

1.3.5 Multi-Objective View Selection Problem

When two objective functions needs to be minimized simultaneously, we are dealing with the multi-objective view selection problem (Dhote & Ali, 2009). While single objective view selection problems received significant attention in the past and several heuristic methods have been proposed for solving this class of problems (see Table 2.3) the multi-objective view selection is rarely addressed in the literature and introduces a broad area of research. The multi-objective view selection problem that is investigated in this research is defined as below:

Select a subset, $M$, of views among a set of all views, $V$ such that:

\[
Q(M) \text{ and } U(M) \text{ minimized}
\]

and:

\[
DS(M) < DS \text{ is satisfied}
\]

This problem can be stated as minimizing both the total query response time, $Q(M)$ and the total view update time, $U(M)$ such that total disk space, $DS(M)$ for storing all views is less than $DS$, a pre-defined size of the disk. In fact, $DS$, is the size of the allocated disk space for saving all the views.

1.4 Evolutionary Multi-Objective Optimization

Evolutionary Algorithms are a type of heuristics which imitates biological evolution in nature and are based on the Darwinian Principle of “natural selection”. In nature, at a given time, several organisms co-exist and compete for obtaining limited available resources. However, only strong and highly fit organisms will win the competition, survive and reproduce. Evolutionary algorithms are a research area in computer science
and are increasingly applied to many complex optimization problems in several fields such as: Medicine (Yang, Reinstein, Pai, Xu, & Carroll, 1998), Robotic (Zalzala & Fleming, 1997), Engineering (Bowden, 1992) and Image Processing (Chen, Nakao, & Arakaki, 1997). A conventional evolutionary algorithm works as follows: at first a real-world problem solution must be encoded as a computer data structure (often an array of binary values). The encoded solution is called an individual or chromosome. The algorithm starts by random creation of the first generation. Then the individuals within the current generation are evaluated by means of a fitness function to measure how good they are. Each individual is assigned a score called the fitness value which reflects the goodness degree of the individual. Thereafter, a selection operator will select individuals with the highest fitness value among the whole population. Then crossover and mutation operators are applied to the selected individuals as parents to produce the new offspring. The offspring forms part of the new generation and the same procedure will be repeated for the next generation until the termination criteria holds. The termination criteria can be a convergence to a satisfactory solution or exceeding a predetermined number of generations. Figure 1.1 shows a general flowchart for an evolutionary algorithm (Deb, 2001; Haupt & Haupt, 1997; Sivanandam & Deepa, 2009).

The single objective optimization problem is the minimization/maximization problem of only one objective function in the presence of some constraints. So far a large amount of effort has been devoted to the understanding, design (Coley, 1998; Michalewicz, 1996; Sivanandam & Deepa, 2009) and application (Chen et al., 1997; Cohoon, Hegde, Martin, & Richards, 1991; Nordvik & Renders, 1991; Schulze-Kremer, 1994; Yang et al., 1998; Zalzala & Fleming, 1997) of single objective genetic algorithms.
However, in the real-world there exist some problems which naturally involve optimization of multiple goals at the same time. These types of optimization problem are different and are called the multi-objective optimization problem. In this kind of optimization problem all the objectives must be taken into account simultaneously. Furthermore, instead of a single optimal solution which is expected in a single objective
optimization we deal with a set of trade-off solutions each of which can be regarded as
the optimal solution. Most classical approaches for solving the multi-objective problem
transform the inherently multi-objective problem to the parametric single objective
problem and then applying the common methods designed for the single objective
problems. Such reduction ignores the fundamental difference between these two types
of problems and is highly dependent on the parameters chosen. Advantages of
evolutionary algorithms for solving multi-objective algorithms such as view selection
problem are as follows:

- The population based feature of these algorithms allows multiple solution of the
  problem to co-evolve within a single run of the algorithm. This characteristic is
  well suited to multi-objective problems where a set of solutions are desirable
  rather than a single solution.

- Secondly, these algorithms introduce some sense of parallelism in solving the
  problem since in a single run of the algorithm, the evolution process is
  performed for several individuals in the population simultaneously and thus they
  improve the overall performance.

- Evolutionary algorithms do not require much knowledge (such as gradient
  evaluation) about the given problem. All the information they need is only the
  objective function (Coello & Lamont, 2004).
1.5 Research Objectives

The objectives considered for this research are the following:

- Investigate the application of different evolutionary multi-objective algorithms in order to solve the view selection problem.
- To compare the performance of different evolutionary algorithms with respect to total query response time and total view update time subject to the constraint of total disk space.
- To investigate the convergence, diversity and computational time of the evolutionary multi-objective algorithms.

1.6 Research Scope

The data warehouse is a repository of integrated information extracted from several source of operational systems and are made available for analytical queries for the purpose of decision making. This research is focused on the study of the view selection problem in a data warehouse environment only. Given a set of possible queries, and disk space constraint, the goal of selection is to select a subset of views to minimize both the total query response time and the total view update time simultaneously subject to this constraint. There are two well-known schemas that are used as the data structure for modeling data in a warehouse, called the *star schema* and the *snowflake schema*. However, in this research the *star schema* is used as a data warehouse model because it is more popular and efficient in data access (England & Powell, 2007; Han, Kamber, & Pei, 2005). Furthermore it is simpler and more consistent than the snowflake schema (Itl Education Solutions Limited, 2010; Rainardi, 2007). For calculating the query response time using a view the *linear cost model* which will be discussed in Section 2.9 is used.

Dimension hierarchies as logical arrangement of attributes in dimension tables provide a
navigational path for roll up and drill down queries. Some researchers in view selection field ignore the issue of dimension hierarchies in solving the problem while other researchers accommodate them in their problem statement. However, this research investigates the view selection problem in the presence of dimension hierarchies (refer to Section 2.6 for a discussion on dimension hierarchies). Two common structures are found in previous works for logical organization of views. These are AND-OR view graphs (Gupta & Mumick, 2005) and dependency lattice framework (Harinarayan et al., 1996). This research is only based on the dependency lattice framework which is suggested in (Harinarayan et al., 1996). The view selection algorithms can be static or dynamic. In dynamic view selection the pattern of user queries changes over time while it is fixed in static. In this research the investigated algorithm operate merely on the static form of view selection. However, the static selection of views is also useful in the starter phase of any dynamic algorithm. The view selection algorithms investigated in this research will be evaluated against some problem instances. Each problem instance is defined as a set of possible views (which includes view size, view update frequency and query update frequency) as well as the relationships with them. The database for problem instances is populated based on the Transaction Processing Performance Council (TPC) (http://www.tpc.org) proposal which is a database generator and extensively used in decision support research. The data are uniformly populated with zero skew.

1.7 Research Methodology

Our methodology for this research is divided by the three following phases:

A. Problem based phase:
   - To study the view selection problem fundamental and principles.
   - To review different methods used for solving view selection problem.
• To review different variation of the problem addressed in the past.
• To examine different techniques for estimating the size of views.

B. Method Based phase:
• To study evolutionary multi-objective principles and fundamentals
• To identify and investigate well-known evolutionary multi-objective algorithms.
• To study the different performance metrics in order to assess the performance of evolutionary multi-objective optimization algorithms. In particular, we are interested to find out which metrics is suitable for evaluating evolutionary algorithms designed to solve the view selection problem.

C. Development Phase:
• To define an appropriate way for representing the view selection problem solution with respect to the evolutionary algorithm.
• To study different methods for dealing with problem constraints and adopting a proper constraint handler to the problem
• To develop an application to implement the evolutionary algorithms as well as implementing a framework for the definition of view selection problem instances using Visual Basic

D. Analysis
• To assess different convergence, diversity and computational time of the evolutionary algorithms applied to a set of problem instances of view selection problem.

Figure 1.2 shows different phases of our methodology for this research.
1.8 Thesis Outline

The current chapter is a summary of what is intended to be presented to the readers of this thesis. However, the remainder of thesis is organized as the following chapters:

- **Chapter 2: Materialized View Selection**

  This chapter discusses principles and fundamentals related to the view selection problem as well as the related works that has been done in this area. The chapter starts with a background to the data warehouse concept. Then, some principles and definitions in relation to the view selection problem are explained. Thereafter, the single and multi-objective view selection problem is formally defined. The overview of related works is divided into two categories: The works addressed by the single objective view selection problem and the work pertinent to the multi-objective view selection problem.

- **Chapter 3 : Evolutionary Multi-Objective Optimization**

  This chapter gives an introduction to the multi-objective problems principles; evolutionary algorithms fundamentals as well as describing some well-known multi-objective evolutionary algorithms intended to solve the general multi-objective optimization problem. The chapter then continues by presenting the existing
performance metric for the performance assessment of evolutionary multi-objective algorithm.

- **Chapter 4: Methodology**

In Chapter 4, the application of several multi-objective optimization algorithms to the multi-objective view selection problem is discussed. The eight (8) different evolutionary algorithms: *WBGA, VEGA, NSGA, NSGA-II, SPEA, SPEA-II, MOGA, NPGA* is examined over a set of problem instances for the view selection problem.

- **Chapter 5: Results and Discussion**

This chapter presents the experimental result obtained by applying the algorithms over the problem instances.

- **Chapter 6: Conclusion**

This chapter is our conclusion of the work done in this research. The chapter gives the recommendation for the most suitable algorithms which outperforms all the other algorithms experimented with in solving the multi-objective view selection problem.
Chapter 2. Materialized View Selection

2.1 Background

Over the years, huge amounts of data has been collected in conventional database systems in the form of relational tables, spreadsheets, documents, flat files or even external data (Ponniah, 2001). This data has been scattered over multiple, independent and heterogeneous data sources with different types of software or even hardware. For example, one system may use the IBM hardware architecture while the other system is based on the Mac hardware design. The Database Management System (DBMS) may differ from Oracle to the SQL Server between these two different sources. Furthermore, often, the data is stored in different databases and may include some inconsistencies and incompatibilities. For example, in one source the length of measurement may be based on the metric while in another system, the measurement may be based on the imperial system. Again, encoding and naming conventions may differ.

Traditional database systems perform the normal daily operations of an organization. For example, they generate invoice, print payrolls and bills, and carry out transactions on bank accounts. In fact, the Online transaction processing (OLTP) systems have been effective systems for the requirements they have been designed for and, organizations are extremely dependent on this type of systems without which the wheel of business will not turn (Ponniah, 2001).

On the other hand, business analysts realized the importance of the large volume of data that has been collected on a regular basis. These data is useful for efficient decision making (Lin & Kuo, 2004). These professional users are interested in detecting the
business trends in these data. For instance, the analyst may look for an answer to the question: “why the total sales for the specific city and specific product have not been as expected during last decade”. The information that analysts require is called strategic information (Ponniah, 2001), and are used for the purpose of efficient decision making by managers and executives. An example of a decision can be, establishing a new store, or decreasing the price of a specific product.

However, the different nature and aims of OLTP systems may prevent analysts from easily retrieving such kinds of information and thus they require a central, coherent, integrated and homogeneous environment to perform their analytical queries.

As a promising response to this weakness, the data warehouse concept (Inmon & Kelley, 1993) and On-line Analytical Processing (OLAP) systems was introduced. The main idea of the data warehouse concept is to extract heterogeneous and inconsistent data scattered over several operational databases, transform them into a consistent and homogeneous form and load them to the central and standalone repository for the purpose of decision making. Table 2.1 lists some of the key difference between OLTP and OLAP systems (Ponniah, 2001).

Table 2.1 Differences Between OLTP and OLAP Systems

<table>
<thead>
<tr>
<th></th>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Content</strong></td>
<td>Current values</td>
<td>Historical values</td>
</tr>
<tr>
<td><strong>Data structure</strong></td>
<td>Optimized for transactions</td>
<td>Optimized for complex queries</td>
</tr>
<tr>
<td><strong>Access type</strong></td>
<td>Read-Write</td>
<td>Read Only</td>
</tr>
<tr>
<td><strong>Response time</strong></td>
<td>Sub-seconds</td>
<td>Several hours to days</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>MB-GB</td>
<td>GB-TB</td>
</tr>
</tbody>
</table>

“A Data Warehouse is a subject oriented, integrated, nonvolatile, and time variant collection of data in support of the management’s decisions.” (Inmon, 2005, p. 29).

Unlike conventional operational systems in which the data stored is based on a particular application, data in the data warehouse is oriented to major business subjects.
Examples of business subjects can be stores, products or customers as illustrated in Figure 2.4 (Bhansali, 2009; Ponniah, 2001). Data warehouse is built by integrating heterogeneous data from multiple operational systems (Han et al., 2005; Ponniah, 2001). The data warehouse is nonvolatile because in contrast to operational systems in which records are deleted, modified or added, the data in the data warehouse is read-only and the only changes that would occur in the data warehouse is the insertion of new rows to the base table during periodical load from operational sources. As a result, the data warehouse repository is always growing (Rob & Coronel, 2007). In order to retain the history of the data, the previous records remain unchanged (Bhansali, 2009).

In operational systems, the value of a specific record reflects the current information. If the value is updated by a transaction, the old value may be lost. For example, the balance of a banking account implies the customer’s balance as of that moment and not necessarily the balance of one week ago. But often, analysts need past information in order to discern the trends. For example, an analyst may be interested in knowing the buying pattern of a group of customer within a specific time frame and requires a history of purchases that have been made by these customers in that period of time. The data in the data warehouse consists of a series of snapshots that may be taken during a period of 15 years, instead of a 3 months basis which is customary in business operational systems (Khan, 2003), and thus, tend to be very large and grow over time. These data provide a historical perspective to analytical users. The time variant feature is considered as a significant element of a data warehouse (Ponniah, 2001).

### 2.2 The Data Warehouse Architecture

The data warehouse building process starts by extracting autonomous data from different data sources such as operational databases, flat files and webpages. Thereafter these data are cleansed and filtered; any data inconsistencies are resolved. Examples of
inconsistencies can be the difference in encoding, naming conventions and units of measurement. Thereafter, they are subsequently transformed to a common format, the data are then loaded to a separate dedicated central database (Hobbs & Hillson, 1999). This database will then be available for:

- End users
- Data mining tools
- Reporting tasks

The three main steps of building a common data warehouse called the Extract, Transform, Load (ETL) process can be summarized as below:

- **Extract**: gathering raw data through several operational sources with diverse formats
- **Transform**: cleaning, resolving inconsistencies and converting to the uniform format
- **Load**: move the processed data to the central database.

Figure 2.1 shows the general procedure for building a common data warehouse.

![Figure 2.1 Data Warehouse Architecture](image)

As a result of the periodical ETL processes, a central and integrated repository with a huge amount of historical data collected during several years are made available to the
analysts allowing them to issue analytical queries efficiently and in a more convenient manner (Ponniah, 2001).

The following are some of the advantages of a data warehouse (Limaye, 2009; Ye, Gu, Yang, & Liu, 2005):

- A data warehouse provides decision makers with a consolidated environment to access data which were difficult to obtain previously.
- By isolating decision support systems from operational systems, local processing at OLTP systems remain un-affected
- A data warehouse can operate even when operational sources are unavailable temporarily

The disadvantages of the data warehouse are as follows (Błażewicz, Kubiak, Morzy, & Rusinkiewicz, 2003):

- Since the data warehouse stores large amount of data from multiple sources during several years separate from the operational database, a big capacity of storage is required for accommodating these data.
- After they have been loaded to the central database of a data warehouse, the data in the operational sources are liable to change which will cause data inconsistency. In order to keep the data in the data warehouse consistent with the source data, periodical updates are performed. The frequency of the update is decided by the administrator. Considering this weakness, one can conclude that the data warehouse is not well suited for users who are interested to access current data.
2.3 Terminology

- **Relational Model**
  The Relational model is a simple and powerful database model. The model represents data in the form of two dimensional tables. Each table represents a real-world object such as a place or a person. In other words, a relational database model is based on a set of tables (Narang, 2006; Shenai & Krishna, 1992). An example of a relational model for a sale system is shown in Figure 2.5.

- **Entity-Relationship Model**
  Introduced by (Chen, 1976) the Entity-Relationship (E-R) model is the most popular conceptual model for designing database (Itl Education Solutions Limited, 2010). The E-R model, as the name suggests views the real world as entities and relationships between them. The entity is an object of interest such as person, place, thing or concept. Figure 2.2 shows an example of an E-R model (Shenai & Krishna, 1992).

- **Query**
  A query is a question asked by the user against the existing relations in the database. For example, asking the total sales for each product type sold in each city in each customer region constitutes a query. A sample query is shown in Section 2.7.

- **Row, Tuple, Record**
  In database terminology, rows, tuples or records are interchangeable terms for addressing a line of data within a table. Throughout this thesis, the terms row, tuple and record are used interchangeably. As an example, each row of the tables in Figure 2.5 is considered as a tuple or record (Norman, 2003; Telles, 2007).

- **View, Pre-computed result or pre-aggregated result**
  In database theory view is a virtual table that is derived from a set of base tables (Teorey et al., 2005). Therefore, the view defines a function from a set of base tables.
to the derived table (Gupta & Mumick, 1995). The rows in view are computed from underlying tables and in contrast to base tables are not necessarily stored in the physical disk (Elmasri & Navathe, 2003; Ramakrishnan & Gehrke, 2002). Thereafter, throughout this thesis, the term *view*, *pre-computed result*, and *pre-aggregated result* are used interchangeably. Examples of views are shown Figure 2.9.

- **Materialized View**

  The *view* is called a *materialized view* if the view’s record is saved on disk (Gupta & Mumick, 1995).

- **Table or Relation**

  A table or relation is a two dimensional structure consisting of rows and columns. For better understanding, the table can be imagined like a spreadsheet. The table consists of all information related to a specific object (Adamski & Finnegan, 2007; Telles, 2007). Throughout this thesis, the terms table and relation are used interchangeably. An example of a table is shown in Figure 2.5

### 2.4 Dimensional Modeling

In operational systems data are commonly represented as an Entity Relationship (E-R) model. Within this model each entity is represented by a table, the attributes of the entities are shown as columns of the tables and the tables are connected together by using the primary/foreign keys. In order to optimize storage in these systems, the normalization procedure is applied to these tables in several forms. By keeping only one copy of data, normalization helps to eliminate data redundancy in tables and hence establish data consistency (Farrell, 2010; Hobbs & Hillson, 1999; Sumathi & Esakkirajan, 2007). As an alternative approach to the popular entity relationship (E-R) modeling mostly used in commercial database systems, and in order to meet the user
requirements of a data warehousing environment, a dimensional modeling approach is used (Hobbs & Hillson, 1999). Dimensional modeling is a technique for the logical design to support user queries in a data warehouse and improving the query performance. Even though E-R modeling is advantageous in online transactional systems where queries are short and simple (Petkovic, 2000) it is not well suited for decision support systems in which the query efficiency and loading data are important (Chaudhuri & Dayal, 1997).

Although normalization is considered as an appropriate technique in OLTP databases it is not sufficient in OLAP systems for following reasons (Hobbs, Hillson, & Lawande, 2003; Nagabhushana, 2008):

- They are too complex to be easily understood

A normalized entity-relationship diagram adds extra tables and relationship and thus increases the complexity of the diagram. Therefore a normalized E-R diagram does not have enough simplicity and is not user friendly. For example, a reader may compare the simple star-like diagram in Figure 2.4 with the normalized E-R model in Figure 2.2.

- Users require Standard Query Language (SQL) knowledge to deal with normalized data structures

Even for simple forms of queries the user needs to know SQL. However, decision makers and senior executives are not expected to learn programming codes.

- Normalized databases are not well optimized for analytical queries.

Analytical queries by their nature involve the aggregation of large numbers of records. Processing such complex queries in normalized structures is slow and inefficient.
Based on the relational model, there are two most common schemas that are used as the data structure for modeling data in a warehouse, called the star schema and the snowflake schema (Ponniah, 2001).

Figure 2.2 Example of Normalized Entity Relationship Model

### 2.5 The Star Schema

The star schema is the most simple and natural way for a logical design of a data warehouse (Parida, 2005). The star schema, consists of two basic objects. One fact table (placed in center of star) and many dimension tables (placed on points of star). The way the fact table and dimension tables are connected together is similar to the star shape (see Figure 2.3). The fact table and dimensional tables are connected together by means of the primary and foreign keys. The primary key of a fact table is a composite key, consisting of the primary keys from each dimension table. For example for the star schema in Figure 2.4 the primary key of the Sales fact table is the composite key consisting of CustomerID, StoreID, ProductID which are the primary keys in the Customer, Store and Product dimension tables respectively. The fact table’s attributes consist of two types; namely, measurements type attributes and the primary keys from the dimensional tables. Often, the fact table is a deep table, that is, it includes large amount of historical records. In contrast to the fact tables, dimensional tables do not have too many records but instead, they are wide tables, which means they have large number of attributes (Ponniah, 2001).
Figure 2.3 Star-Like Modeling of Data Warehouse

Figure 2.4 Star Schema for Sales System.

Figure 2.4 shows a sample star schema for a sales system. It models a sales system where products are sold to customers through a store. The sample fact tables and dimensional tables are shown in Figure 2.5. The dimension tables describe the business subjects such as Customer, Store, Product, while the fact tables store some measurements about dimensions such as the amount of sales. The measurement attribute in the fact table are often in the form of numerical values while the dimensional tables usually include descriptive textual attributes. In a star schema each record in the fact table corresponds to a single record in each dimension surrounded by it. For example, each record in the fact table of Figure 2.5 represents the price of the specific product in
the product dimension table sold to a specific customer in the customer dimension table at a specific store in the store dimension table. However, although the star schema is an easy to understand and implement model, it increases the degree of data redundancy. (Hobbs & Hillson, 1999; Hoberman, 2009; Ponniah, 2001).

**2.6 Dimension Hierarchies**

The attributes in the dimension tables, usually form a hierarchy as a logical structure to facilitate the roll up and drill down operations (see Figure 2.6). Roll up operation is a series of user queries that navigates from detailed results to summarized result. Drill down are the reverse operation of roll up in which a user issues a series of queries to navigate from summarized results to more granular results (see Figure 2.7) (Han et al., 2005). Within each hierarchy, a particular level is connected to more detailed level below and less detailed level above (except the top and bottom level) (Parida, 2005). Each level in the hierarchy indicates a specific granularity degree. Figure 2.6 depicts one hierarchy for each dimension table of the sales system example in Figure 2.4. For instance, consider the store dimension table where the attributes storeID, city, region form a hierarchy as illustrated in Figure 2.6. Going up from storeID towards the Region the data is summarized and vice versa. The notation All in Figure 2.6 indicates aggregation of all records in the corresponding dimension.
2.7 OLAP Query format

A sample SQL query for star schema is as below (Runapongsa, Nadeau, & Teorey, 1999):

```sql
SELECT SUM(Price), Customer.Region, Store.City, Product.Type
FROM Sales, Store, Customer, Product
GROUP BY Customer.Region, Store.City, Product.Type
```

In the `select` clause the calculation made is based on the numeric measurement attribute in the fact table, and in the `groups-by` clause, each attribute is an aggregation level picked from a dimension hierarchy. For example, in the above sample query, in the group by clause, `Region` is selected from the dimension table `Customer`, `City` is selected from the dimension table `Store`, and `Type` is selected from the dimension table `Product`.
from the dimension table *Store* and *Type* is selected from the dimension table *Product* as illustrated in Figure 2.8. For simplicity, hereafter we denote this query as:

\[
\text{Sales (Region, City, Type)}
\]

Figure 2.8 Picking Up a Hierarchy Level from Each Dimension Hierarchy to Form a Group-By Query

If a star schema consists of \(k\) different dimension tables and within dimension \(i\), there exist \(h_i\) different hierarchy levels, then the number of all possible combination of *group-by* queries with this format is calculated as (Ahmed, Agrawal, Nandkeolyar, & Sundararaghavan, 2007):

\[
|Q| = \prod_{i=1}^{k} h_i
\]  

where \(Q\) is the set of all possible *group-by* queries. The result of each of these queries can be considered as a view and thus each query corresponds to a particular view and therefore, the number of all possible views is equal to the number of the *group by queries*, \(|Q| = |V|\). In the example shown in Figure 2.8, since in each dimension hierarchy there are 4 different levels \(|Q| = \prod_{i=1}^{3}(4) = 64\), therefore the number of all possible views/queries is 64. Each time, the user submits a query; the query is one of these 64 possible queries. Hereafter throughout this thesis, it is assumed that whenever
a user requests a view by issuing a specific *group-by* query, the request is for the entire view and not a part of it.

### 2.8 Dependency lattice

The set of all views can be structured as a lattice framework as introduced by (Harinarayan et al., 1996) to display the relationship between different views (see Figure 2.9). In that lattice, the relationship between views are expressed as partial order denoted by . (Lawrence & Rau-Chaplin, 2006). Since there is a corresponding *group-by* query for each view a dependency lattice which is made from equivalent queries of views as depicted in Figure 2.9 can be constructed. This dependency lattice is shown in Figure 2.10.

As an example, consider the dependency lattice for the sales system illustrated in Figure 2.11. We denote the lattice by $G = (ND,E)$ where $ND$ is set of nodes and $E$ is set of directed edges. Each node in this lattice represents a particular *view/group by query*. A directed edge $(v_i,v_j) \in E$ or $v_i \preceq v_j$ if $v_i$ can be computed through $v_j$. For instance, in the sales system lattice of Figure 2.11, $5 \preceq 1$ since the view number 5 can be computed from view number 1. By organizing the views as dependency lattice, the problem of finding the right set of views can be reduced to the problem of finding the proper set of nodes among all possible nodes in the lattice. There is a top and largest view in the lattice, which represents the fact table and by using it every other view in the lattice is computable. The data in the fact table are in the highest level of detail. Similarly, the bottom and smallest view represents a view which includes only one record. This record is the aggregation of all existing records in the fact table and is the
most summarized view. This view can be computed from every other view in the lattice. In fact, the smallest materialized ancestor of a view is used to answer a query unless the corresponding view to that query is materialized.
2.9 Linear Cost Model

The linear cost was proposed by (Harinarayan et al., 1996) who assumed that the time for answering a query using view $v$, has a linear relationship to the number of records in view $v$ (or alternatively the size of the view). i.e.:

$$t(v) = a \times |v| + b$$ \hspace{1cm} (2.2)

where $t(v)$ is query response time for answering a query using view $v$, and $a$ and $b$ are constants. Hereafter, throughout this thesis we use linear cost model for computing the query response time using a specific view. It is to be noted in this research, queries are assumed to access all records in the view rather than the partial view.
2.10 Total query response time

The total query response time is the time for answering all the possible queries as stated earlier in Section 1.3.1. It is to be noted that since query speedup is caused through the help of materialized views, hence, the maximum query response time, occurs when there is no materialized view; \( Q_{\text{max}} = Q(\phi) \). Similarly, when all views are materialized, we have a minimum query response time, \( Q_{\text{min}} = Q(V) \), since for every incoming query there is a pre-computed result.

\[
Q_{\text{min}} \leq Q(M) \leq Q_{\text{max}}
\]

However the required time for answering query, \( q \), in presence of a set of materialized views, \( M \), is calculated as following:

\[
t(q, M) = \begin{cases} 
\min_{v \in M'} |v| & \text{if } M' \neq \phi \\
|F| & \text{otherwise}
\end{cases}
\]

\[
M' = M \cap \text{Ancestors}(v)
\]

where \( |v| \) is the number of records in view \( v \) and \( M' \) is the set of materialized ancestor for query \( q \). \( |F| \) is the number of records in the fact table and \( \text{Ancestors}(v) \) is the set of ancestors for view \( v \) in the dependency lattice (Gou et al., 2006).

- **Example 2.1** the Figure 2.12 shows a dependency lattice with the current set of materialized views (the nodes are shown in gray), \( M = \{F, b, c\} \). The number of records in each view is written in the nodes. Assuming that we intend to calculate the time needed for answering a query which corresponds to view \( d \). The \( t(d, M) \) is calculated as below:

\[
\text{Ancestors}(d) = \{F, a, b\}
\]

\[
M' = \{F, b\}
\]

\[
t(d, \{F, b, c\}) = \min\{100, 45\} = 45
\]
where Ancestors(d) is a set of ancestors for view d in dependency lattice of Figure 2.12.

Note that the computation of \( t(d, M) \) is based on the linear cost model suggested in (Harinarayan et al., 1996).

**2.11 Total View Update Time (or View Maintenance Time)**

As mentioned before in Section 1.3.2 the total update time refers to the time required for updating all the materialized views. It is to be noted that minimum update time \( (U_{\text{min}}) \) happens when there is no materialized view \( (M = \emptyset) \) and thus no update process required. On the other hand, when all possible views are materialized \( (M = V) \), all views may need to be updated and therefore the time for updating views reach the highest value \( (U_{\text{max}}) \).

\[
U_{\text{min}} \leq U \leq U_{\text{max}}
\]  

Generally, there are two update policies and for both of them we use the fact table or the smallest ancestor of views as a source of updating: these are: incremental update and re-computational update (Shah et al., 2006)
2.12 View Size Estimation

The calculation of the query response time and the storage requirement of views need the prior knowledge about the size of views as a parameter. The determination of the exact and actual size of a view requires computing the view from the fact table and is thus, an expensive exercise (Teorey et al., 2005). As a sub-problem of the view selection problem, view size estimation addresses the issue of how the amount of disk space required for storing a view can be predicted without actually computing and saving it on disk. Two main objectives for view size estimation are accuracy and speed of estimation. The methods used to perform the estimation may underestimate or overestimate. Although overestimation is acceptable because they present a conservative approach in managing disk space but underestimation is not desirable (Nadeau & Teorey, 2001). In order to represent the error of estimation, the following formula can be considered as denoted in (Nadeau & Teorey, 2001):

\[ \text{Estimation Error} = \frac{\text{Actual Size} - \text{Estimated Size}}{\text{Actual Size}} \]  

According to (Shukla, Deshpande, Naughton, & Ramasamy, 1996) three different types of methods can be used to estimate view sizes which are analytical methods, linear sampling methods and probabilistic counting method.

2.13 View Selection Problem

In dealing with materialized views the following choices are considered based on (Zhang, Yao, & Yang, 2001):

- **Full materialization**

As an ideal choice, we would like to save all possible views in the system on hard disk. In terms of query response time, we will gain maximum acceleration because, for every
incoming query, there is a pre-computed result which can be used to answer the query. However, in practice, this option is not feasible in some systems because storing all possible views can take a large amount of disk space which may not be supported by these systems. Moreover, materializing all views will cause the update process to take a long time.

- **No Materialization**

By using this option, no views are materialized at all. Although the amount of disk space used for storing the views would thus, be zero and no update process is required, this option has the poorest performance in terms of query response time, since for every incoming query we need to refer to the base table.

- **Partial Materialization**

In partial materialization only a subset of all possible views are selected to be materialized. Hence, a balance may be achievable between the query response time, update time and the size of disk space.

Figure 2.13 shows above three choices in a 2D space.

The question that arises here is, if we are going to select a subset of views to save on disk space, which subset is the most appropriate one? (Horng, Chang, Lin, & Kao, 1999) The answer is, the subset which most optimize our objectives function(s) while satisfying our constraint(s) (Jamil & Modica, 2001). For example, Figure 2.14 shows three view selection problem solutions with corresponding $Q$ and $U$ values. Of these solutions, solution $A$ is considered as the best solution since it has the minimum values of $U$ and $Q$.
View selection problems can be regarded as a search problem where the search space is the set of all possible subset of views and the search goals is a particular subset which minimizes one function(s) subject to constraint(s) (Jamil & Modica, 2001). The problem is important to the design and optimization of data warehouses (Shah et al., 2006; Zhang et al., 2001) and is considered as NP-Hard (Kumar & Ghoshal, 2009).

In order to select a subset of views two approaches are possible as follows (Jamil & Modica, 2001; Talebi, Chirkova, & Fathi, 2009; Zhang & Yang, 1999a):
A) In an exhaustive search strategy all points in the search space needs to be enumerated in order to find the optimal solution (if there exists any).

B) Adopting heuristic algorithms to deliver a near-optimal solution within reasonable time.

In the first approach, we enumerate all $2^{|V|}$ candidates in order to find the best one among all subsets of views. Although the optimal solution will be found by this search method if it exists and the method is easy to implement, it takes a long time to find unless the size of the search space is small. The time complexity for this method is $O(2^{|V|})$. However, in practice, we avoid this approach.

In the heuristic search method we try to find a near optimal solution by pruning the search space and spending a reasonable time rather than carrying out an exhaustive search in execution time (Zhang et al., 2001). If the true optimal solution cannot be obtained in practice we can trade the optimality for efficiency. That means we sacrifice the exact optimal solutions for obtaining near-optimal solution in reasonable time (Dorigo & Stützle, 2004).

According to the literature (Hanusse et al., 2009; Harinarayan et al., 1996; Liang et al., 2001; Lin & Kuo, 2004; Zhou, Wu, et al., 2008), there are many variation of the view selection problems and they have been well studied. All of them can be classified into main categories; namely, a single objective view selection category or a multi-objective view selection category.

### 2.14 Single Objective View Selection Problem

A single objective view selection problem is concerned with finding a proper subset of all possible views such that one objective function (or combination of multiple objectives) is minimized and constraints are satisfied as defined below:
Select a subset $M$ of views among the set of all views, $V$ such that:

**Single Objective Function** e.g. $Q(M)$ or $U(M)$ **Minimized**

**Constraint** e.g. $DS(M) \leq DS$ is satisfied

Table 2.2 is a list of the single objective view selection problem and its variations as well as works which addressed that particular variation. Note that the last two variants in Table 2.2 considers a linear combination (or weighted sum) of query response time and update time as a single objective. In this case the multi-objective view selection problem has been reduced to a single objective problem and single objective methods are applied to it.

<table>
<thead>
<tr>
<th>Objective (Minimize)</th>
<th>Constraint</th>
<th>Works</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk space Consumption</td>
<td>Total query response time</td>
<td>(Hanusse et al., 2009)</td>
</tr>
<tr>
<td>Total update time</td>
<td>Total query response time</td>
<td>(Zhou, Wu, et al., 2008)</td>
</tr>
<tr>
<td>Combination of total query response time and total update time</td>
<td>Free</td>
<td></td>
</tr>
</tbody>
</table>
2.14.1 Benefit Function

Let \( v \in V - M \), (see Figure 2.15) be a non-materialized view. The benefit gained after materializing view \( v \) (or adding \( v \) to set \( M \)) with respect to \( M \), the set of already materialized views, is denoted by \( B(v, M) \) and defined as below (Gupta & Mumick, 2005; Harinarayan et al., 1996):

\[
B(v, M) = Q(M) - Q(M \cup v)
\]

In the above equation, \( Q(M) \) is query response time for answering all queries in the presence of set \( M \) of materialized views. \( Q(M \cup v) \) is the query response time for all queries when view \( v \) is added to the current set of materialized views. Note that \( B(v, M) \geq 0 \) because by increasing the number of materialized views the query response time does not decrease that is \( Q(M) \geq Q(M \cup v) \).

Indeed, the benefit of a non-materialized view is the amount of reduction in the total query response time after materializing that view.

\[
\begin{align*}
\text{V-M} & \quad \text{Set of non-materialized views} \\
M & \quad \text{Set of materialized views}
\end{align*}
\]

Figure 2.15 Benefit Calculation

2.14.2 Related Works for Single Objective View Selection Problem

Even though this research concentrates only on the multi-objective form of the view selection problem but a number of prominent works in single objective area are also overviewed. Table 2.4 lists different works which were carried out for the single objective view selection problem as well as their performances.
One of the fundamental works in the area of materialized view selection has been done by (Harinarayan et al., 1996). View dependency lattice which plays an important role in the formulation of the view selection problem has been introduced for the first time in this work. Gupta proposed a greedy algorithm which selects the most beneficial view per unit of disk space at each stage and adds it to the set of already materialized views. The authors prove that if the largest views take $f$ percent of the total allocated disk space for view materialization, the benefit of the selected set of views is at least $(0.63 - f)$ times the benefit of the optimal set of views. The time complexity of the algorithm is $O(k \cdot n^2)$ where $k$ is the number of materialized views and $n$ is the number of all candidate views. Hereafter, throughout this chapter this algorithm is called BPUS.

The paper by (Gupta & Mumick, 1999) is one of the early papers that considers the view selection problem when the objective is minimizing the total query response time and the constraint is the time needed for updating the materialized views. As the view selection problem under update time constraint comes with non-monotonic benefit function, the problem becomes intractable and the greedy algorithm which has been adopted in (Harinarayan et al., 1996) is not applicable. In this case this makes the problem more difficult. In order to satisfy the monotonicity property (Bauer & Lehner, 2003; Gupta & Mumick, 1997), for the special case of the problem the authors have partitioned the lattice into sub-lattices called inverted tree set. Then a greedy algorithm is used to select the best inverted tree set among others at each stage.

The authors in (Shukla et al., 1998a) designed an algorithm called PBS (Pick By Size) which selects the views based on increasing orders of their size and return the solution with the same total query response time as in BPUS. The time complexity of the PBS is much less than the BPUS algorithm and is $O(n \cdot \log n)$ where $n$ is the number of possible views.
In (Shukla, Deshpande, & Naughton, 2000), as a novel study, the problem of selecting views to materialize through a single cube has been extended to multiple cubes in which several fact tables exist. As expected, the multi-cube view materialization seems to be significantly more complex than the conventional single cube version and single cube algorithms must be extended for the multi-cube case. To deal well with the multi-cube case, three different special algorithms called, SimpleLocal, SimpleGlobal and ComplexGlobal were devised. Their results show that applying the multi-cube algorithm to this kind of problem leads to a noticeable performance improvement rather than the traditional single cube view selection algorithms.

(Baralis et al., 1997) proposed two techniques for reducing the size of the search space by keeping only relevant views and removing the views which have the lesser effect on the optimal solution.

(Derakhshan, Dehne, Korn, & Stantic, 2006) introduced an application of simulated annealing approach in solving the view selection problem. Comparing to a heuristic and genetic method the proposed approach provides significant improvement in quality (sum of total query response time and total update time) of obtained solution.

(Lee & Hammer, 1999) investigated the view selection problem when the structure of views is the OR view graph (Gupta & Mumick, 2005) using genetic algorithm. They compared their proposed algorithm with optimal solutions. The optimal solution is calculated using an exhaustive search algorithm for an example of 20 views. The results indicate that the proposed algorithm yields a solution within 90% of the optimal solution quality while exhibiting a linear increase in execution time by increasing the number of views.

The authors in (Yu, Yao, Choi, & Gou, 2003) proposed a constrained evolutionary algorithm but unlike (Lee & Hammer, 1999) they have a novel stochastic ranking
procedures instead of a direct integration of penalty function for handling the constraint. Their algorithm was evaluated against two heuristics and another evolutionary algorithm. The result shows the proposed algorithm performs better than the compared algorithms in terms of minimizing the total query response time and the feasibility of solution.

In (Gou et al., 2006) an $A^*$ algorithm has been developed to solve the view selection problem under disk space constraint. It was claimed that the proposed algorithm improved the solution quality when the disk space limit is small and in that case the greedy $BPUS$ does not work as expected. They used two pruning technique called $H$-Pruning and $F$-Pruning in order to reduce the size of the search space and therefore accelerate the $A^*$ algorithm. Their theoretical and experimental results show the suggested algorithm is powerful, efficient and flexible to this problem.

(Zhang et al., 2001) combined the pure evolutionary algorithm and heuristic algorithm to form a hybrid algorithm. Their experimental result shows that the hybrid algorithm reduces the total query response time and total update time significantly. Furthermore, their study shows either of these algorithms were found to be impractical or unsatisfactory.

In (Nadeau & Teorey, 2002), Algorithm Polynomial Genetic Algorithm ($PGA$), was presented as an alternative in order to improve the time complexity and scalability of the Algorithm in $BPUS$. The proposed algorithm has polynomial time complexity rather than exponential time complexity of the $BPUS$ algorithm. In addition, with increasing number of dimensions $PGA$ performs better.

The authors of (Kalnis et al., 2002) explored the application of randomized search heuristic, namely, Iterative Improvement ($II$) and Simulated Annealing ($SA$) for solving the view selection problem. They modeled the search space as a graph of connected
state. Every node represents a feasible set of views subject to the update/query time constraint. They found that randomized algorithms are applicable to problems with bigger sizes, can be adopted for several variations of the problem and provide near-optimal solution in limited time.

The authors in (Kotidis & Roussopoulos, 1999) proposed a dynamic view selection system, called Dynamat. This system constantly monitors incoming queries and dynamically materializes views at multiple levels of granularity. The system unifies the selection of views and the updating of views in one problem and considers both disk space and update constraints. Another promising feature of Dynamat is it does not materialize the entire view but only a segment of the view that are relevant to queries. The experiment shows that Dynamat outperforms optimal static view selection algorithm.

(Ashadevi & Balasubramanian, 2008) developed a framework for materialized view selection in order to achieve the best combination of query response time and update time subject to disk space constraint. The views with a high query frequency are selected as initial materialized views. The proposed method, removes the views with low query frequency and high disk space requirements from the pool of already selected views. No comparisons to similar works were carried out.

In (Wang & Zhang, 2005), The proposed method works in two stages. In the first stage, the initial content of individuals is produced by using a greedy algorithm. The greedy algorithm selects the most beneficial views subject to a dedicated disk space. In the second stage, the solution is improved by using genetic algorithm. In order to deal with disk space constraint during the genetic algorithm process, a repair method is used. Their experimental result shows that the proposed algorithm is superior to heuristic and canonical genetic algorithm.
In the paper by (Kumar & Ghoshal, 2009), in order to decrease the high time complexity of the BPUS algorithm, an improved algorithm, called RLGA was suggested. RLGA selects views to materialize through a reduced dependency lattice instead of a complete lattice which is used in BPUS. The authors claimed that, the high time complexity of the BPUS algorithm is because of the high number of computation of the benefit functions. According to their experiments, in comparison to BPUS, RLGA selects good views with fewer re-computations and thus improves the execution time.

The authors in (Li et al., 2005) introduced the Integer Programming model in order to obtain an exact global optimal solution. The experimental result shows the practicality of the proposed approach in problem instances with realistic sizes.

The authors in (Talebi et al., 2009), have modeled the view selection problem as an Integer Programming (IP). An IP model was used to obtain the guaranteed optimal solutions. In addition, a heuristic method was proposed in order to find the competitive inexact solution in the cases that an exact method is not applicable. The authors experimentally compared the proposed approach against works in (Harinarayan et al., 1996) and (Shukla, Deshpande, & Naughton, 1998b) and delineate the applicability areas of the proposed and compared approaches.

In (Horng et al., 1999) the researchers applied Genetic Algorithm combining with Local Search (GLS). While the local search finds good solutions in a small region of the search space, the genetic algorithm finds good solutions for the whole search space. After the creation of the initial population and after applying the crossover and mutation operator the local search is used to improve the solution. Although no comparison to other works was done their result shows that GLS can steadily reach a good solution in a few seconds.
(Horng et al., 2003) proposed a genetic local search algorithm for solving the view selection problem. From the experimental results they found the proposed approach performed well in comparison to researchers’ previous work called YKL97.

The work in (Aouiche, Jouve, & Darmont, 2006) takes advantage of clustering, a data mining technique, to decide clusters of similar views. Also a greedy algorithm was proposed to select a set of views. Their experimental result shows the proposed strategy caused substantial gain in performance.

(Bauer & Lehner, 2003) and (Ye et al., 2005) focused on solving the view selection problem in a distributed data warehouse environment. Their study shows that the proposed approach yields significantly better results than greedy algorithm directly applied to each node.

In (Lin & Kuo, 2000), the authors adopted a simple genetic algorithm for the view selection problem. A reverse version of the BPUS algorithm was used as a repair method in order to deal with the problem constraint. Whenever the requirement exceeds the view buffer size, the reverse greedy algorithm removes the less beneficial view from the current materialized views set until the disk space constraint was satisfied. The experimental result shows that the genetic algorithm is superior to BPUS algorithm.

(Lin & Kuo, 2004) examined the application of genetic algorithm in solving the view selection problem. For dealing with infeasible solutions, a greedy repair method was incorporated. According to the experimental result the proposed genetic algorithm generates a better solution than the greedy algorithm.

The authors in (Boukra et al., 2007) tried to improve the work in (Yu et al., 2003). They proposed an evolutionary algorithm which replaces the crossover and mutation by an
ant colony algorithm. The experimental result shows that the performance of proposed algorithm is within 90% of the optimal solution while generates feasible solutions.

In (Zhang et al., 1999), explored using genetic algorithm for the selection of views based on a multiple global processing plan (Sellis, 1988; Shim, Sellis, & Nau, 1994; Zhang et al., 1999). They studied the performance of genetic algorithm and other heuristics. Their results reveal that in terms of performance and evaluation cost the combination of genetic algorithm and heuristic algorithm works better than using only one of them. However, they concluded that the genetic algorithm outperforms heuristics algorithm.

(Zhang & Yang, 1999b) addressed dynamic view selection issues. A set of algorithms for the dynamic view selection were proposed. In addition, a framework was developed for dynamic materialized views. The experimental work shows that the introduction of genetic algorithm to the problem may decrease the total cost (a combination of total query response time and total update time)

Several static and dynamic algorithms were proposed in (Fan, 1997). The proposed static algorithms share the greedy skeleton of the algorithm in (Harinarayan et al., 1996) and differs only in how the benefit function was defined. The dynamic algorithm consisted of admission and replacement algorithms. The admission algorithm works like a static algorithm. If the query response time of a specific query exceeds a threshold, the admission algorithm finds the best view to be materialized. As a replacement algorithm, the LRU (Least Recently Used) strategy is adopted. The simulation work was done by comparing the static and dynamic algorithms against several different data warehouses. The result shows that the performance of the proposed static algorithms are close to algorithm in (Harinarayan et al., 1996), and are much faster. However the dynamic algorithms did not work as expected probably because of two reasons: first locality and
second lack of overall performance due to using a combination of admission and replacement algorithms.

(Mami, Coletta, & Bellahsene, 2011) modeled the view selection problem as constraint satisfaction problem (CSP) (Russell & Norvig, 2009) and applied the constraint programming approach to solve the problem. The experiments show that the proposed approach provides better performance than the genetic algorithm subject to solution quality in limited time. The quality of the obtained solution was measured in proportion to the combination of total query response time and total update time. The authors also showed that their approach support scalability with increasing number of views.

In (Yin, Yu, & Lin, 2007), a dynamic method was addressed for solving the view selection problem. The proposed method uses the greedy algorithm, BPUS to select the primary set of materialized views. Decision for admission and replacing view is made based on the ratio of the query frequency over view size, i.e. \( \frac{f_q}{DS(v)} \), and the history of incoming views within a certain period of time, i.e. \( Q_{interval} \). However, no experimental study has been presented.

(Lawrence & Rau-Chaplin, 2006) investigated dynamic view selection. They studied BPUS and three randomized techniques (iterative improvement, simulated annealing and two-phase optimization). The experimental result shows that BPUS perform better than three randomized techniques. However, when the number of dimensions increases, the computational cost of the BPUS algorithm is too high and is thus impractical.

The authors in (Qiu & Ling, 2000) investigated the issue of pruning the search space prior to applying the view selection algorithm. They proposed two methods, called functional dependency filter and size filter in order to filter out large number of unhelpful views. Their test shows impressive result compared to other works.
The authors of (Agrawal, 2005) proposed two heuristic algorithms as well as a 0-1 integer programming for the materialized view selection issue. The heuristic and integer programming algorithms were aimed to solve different versions of the view selection problem. They also addressed the issue of view size estimation. Their findings show that the solutions which are returned by heuristic algorithms are very close to the optimal solution.

In (Shah et al., 2006), the authors proposed a hybrid approach for solving the view selection problem. The basic idea of their approach is to partition the view dependency lattice into two partitions, static partition and dynamic partition. The static views are selected from more detailed views and the dynamic views are selected from the more aggregated views. For selecting both static and dynamic set of materialized views they proposed a greedy algorithm. The proposed approach was compared to the Dynamat system (Kotidis & Roussopoulos, 1999). The result shows that average query and update time saving of the suggested method is higher than Dynamat.

(Hung, 2001) presented similar proofs as that in (Karloff & Mihail, 1999) to show that the optimality degree of BPUS (total response time of greedy solutions /total response time of optimal solution) can be as bad as $\frac{n+4}{8}$ where $n > 0$ is the number of all possible views and higher than $\frac{n}{12}$ as stated in (Karloff & Mihail, 1999).

Table 2.4 summarizes different works carried out with respect to the single objective view selection problem as well as their performances.
## Table 2.3 Different Works for Single Objective View Selection Problem

<table>
<thead>
<tr>
<th>Methods Proposed for Solving Single Objective View Selection Problem</th>
<th>Works</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy Algorithm</td>
<td>(Harinarayan et al., 1996)</td>
</tr>
<tr>
<td></td>
<td>(Gupta et al., 1997)</td>
</tr>
<tr>
<td></td>
<td>(Gupta &amp; Mumick, 1997)</td>
</tr>
<tr>
<td></td>
<td>(Nadeau &amp; Teorey, 2002)</td>
</tr>
<tr>
<td></td>
<td>(Wang &amp; Zhang, 2005)</td>
</tr>
<tr>
<td></td>
<td>(Uchiyama et al., 1999)</td>
</tr>
<tr>
<td></td>
<td>(Dhote &amp; ALi, 2007)</td>
</tr>
<tr>
<td></td>
<td>(Aouiche et al., 2006)</td>
</tr>
<tr>
<td></td>
<td>(Bauer &amp; Lehner, 2003)</td>
</tr>
<tr>
<td></td>
<td>(Ye et al., 2005)</td>
</tr>
<tr>
<td></td>
<td>(Yu et al., 2004)</td>
</tr>
<tr>
<td></td>
<td>(Kumar, Haider, &amp; Kumar, 2010)</td>
</tr>
<tr>
<td></td>
<td>(Agrawal et al., 2007)</td>
</tr>
<tr>
<td></td>
<td>(Lin &amp; Kuo, 2000)</td>
</tr>
<tr>
<td></td>
<td>(Zhou, Xu, Shi, &amp; Hao, 2008)</td>
</tr>
<tr>
<td></td>
<td>(Fan, 1997)</td>
</tr>
<tr>
<td></td>
<td>(Chan, Li, &amp; Feng, 2001)</td>
</tr>
<tr>
<td></td>
<td>(Yin et al., 2007)</td>
</tr>
<tr>
<td></td>
<td>(Serna-Encinas &amp; Hoyo-Montano, 2007)</td>
</tr>
<tr>
<td></td>
<td>(Yousri, Ahmed, &amp; El-Makky, 2005)</td>
</tr>
<tr>
<td></td>
<td>(Ligoudistianos, Theodoratos, &amp; Sellis, 1998)</td>
</tr>
<tr>
<td></td>
<td>(Shah et al., 2006)</td>
</tr>
<tr>
<td></td>
<td>(Chan, Li, &amp; Feng, 1999)</td>
</tr>
<tr>
<td></td>
<td>(Yang et al., 2002)</td>
</tr>
<tr>
<td></td>
<td>(Gupta &amp; Mumick, 1999)</td>
</tr>
<tr>
<td>A* Algorithm</td>
<td>(Gou et al., 2006)</td>
</tr>
<tr>
<td></td>
<td>(Gou et al., 2003)</td>
</tr>
<tr>
<td>Algorithm</td>
<td>References</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Simulated Annealing</td>
<td>(Derakhshan et al., 2006)</td>
</tr>
<tr>
<td></td>
<td>(Kalnis et al., 2002)</td>
</tr>
<tr>
<td></td>
<td>(Phu boon-ob &amp; Auepanwiriyakul, 2007a)</td>
</tr>
<tr>
<td></td>
<td>(Zhou, Xu, et al., 2008)</td>
</tr>
<tr>
<td></td>
<td>(Lawrence &amp; Rau-Chaplin, 2006)</td>
</tr>
<tr>
<td></td>
<td>(Zhou, Wu, et al., 2008)</td>
</tr>
<tr>
<td></td>
<td>(Lee &amp; Hammer, 1999)</td>
</tr>
<tr>
<td></td>
<td>(Yu et al., 2003)</td>
</tr>
<tr>
<td></td>
<td>(Nadeau &amp; Teorey, 2002)</td>
</tr>
<tr>
<td></td>
<td>(Wang &amp; Zhang, 2005)</td>
</tr>
<tr>
<td></td>
<td>(Horng et al., 1999)</td>
</tr>
<tr>
<td></td>
<td>(Lin &amp; Kuo, 2004)</td>
</tr>
<tr>
<td></td>
<td>(Boukra et al., 2007)</td>
</tr>
<tr>
<td></td>
<td>(Lin &amp; Kuo, 2000)</td>
</tr>
<tr>
<td></td>
<td>(Zhou, Xu, et al., 2008)</td>
</tr>
<tr>
<td></td>
<td>(Zhang et al., 1999)</td>
</tr>
<tr>
<td></td>
<td>(Zhou, Wu, et al., 2008)</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>(Agrawal et al., 2007)</td>
</tr>
<tr>
<td></td>
<td>(Agrawal, 2005)</td>
</tr>
<tr>
<td></td>
<td>(Talebi et al., 2009)</td>
</tr>
<tr>
<td></td>
<td>(Agrawal et al., 2007)</td>
</tr>
<tr>
<td>Particle Swarm Algorithm</td>
<td>(Sun &amp; Wang, 2009)</td>
</tr>
<tr>
<td>Integer Programming</td>
<td>(Aouiche et al., 2006)</td>
</tr>
<tr>
<td></td>
<td>(Bauer &amp; Lehner, 2003)</td>
</tr>
<tr>
<td></td>
<td>(Ye et al., 2005)</td>
</tr>
<tr>
<td></td>
<td>(Kumar et al., 2010)</td>
</tr>
<tr>
<td></td>
<td>(Agrawal et al., 2007)</td>
</tr>
<tr>
<td></td>
<td>(Lin &amp; Kuo, 2000)</td>
</tr>
<tr>
<td></td>
<td>(Zhou, Xu, et al., 2008)</td>
</tr>
<tr>
<td>Memetic Algorithm</td>
<td>(Zhang et al., 2009)</td>
</tr>
<tr>
<td></td>
<td>(Horng et al., 1999)</td>
</tr>
<tr>
<td></td>
<td>(Horng et al., 2003)</td>
</tr>
</tbody>
</table>

Table 2.4 Different Works for Solving Single Objective View Selection and Their Performance

<table>
<thead>
<tr>
<th>Paper</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Harinarayan et al., 1996)</td>
<td>The proposed greedy algorithm finds near optimal solution with $O(kn^2)$ time complexity where $k$ is number of views to be selected and $n$ is number of all possible views.</td>
</tr>
<tr>
<td>(Gupta et al., 1997)</td>
<td>The time complexity of proposed R-Greedy algorithm is $O(Kn^2)$ where $k$ is number of structures to be selected and $m$ is number of all possible views.</td>
</tr>
<tr>
<td>(Gupta &amp; Mumick, 1997)</td>
<td>A polynomial time heuristic presented.</td>
</tr>
<tr>
<td>(Nadeau &amp; Teorey, 2002)</td>
<td>The time complexity of the proposed algorithm is $O(dk^l)$ in which $d$ is number of dimension tables and $k$ is number of views and $l$ is number of views in dependency lattice. The space complexity is $O(dk^l)$.</td>
</tr>
<tr>
<td>(Wang &amp; Zhang, 2005)</td>
<td>The proposed algorithm delivers solution with less cost ($Q(M)+U(M)$) than (Gupta &amp; Mumick, 1997) and (Horng et al., 2003).</td>
</tr>
<tr>
<td>(Uchiyama et al., 1999)</td>
<td>The proposed algorithm, PVMA, provides significantly better results than BPUS in (Harinarayan et al., 1996) in large lattices.</td>
</tr>
<tr>
<td>(Aouiche et al., 2006)</td>
<td>The presented strategy guarantees a substantial gain in performance.</td>
</tr>
<tr>
<td>(Bauer &amp; Lehner, 2003)</td>
<td>The distributed greedy algorithm outperforms than greedy algorithm which directly applied to the each node.</td>
</tr>
<tr>
<td>(Ye et al., 2005)</td>
<td>In comparison to applying central methods on individual nodes, the proposed approach in distributed data warehouse is far better in terms of both query response time disk space usage.</td>
</tr>
<tr>
<td>(Kumar et al., 2010)</td>
<td>Proposed algorithm, PVGA, gives significant reduction in execution time.</td>
</tr>
<tr>
<td>(Agrawal et al., 2007)</td>
<td>Heuristic methods find near optimal solutions for some problem instances. The execution time of heuristic method are linear with problem size and less than the that of integer programming.</td>
</tr>
<tr>
<td>(Lin &amp; Kuo, 2000)</td>
<td>In comparison to BPUS, the proposed algorithm reach lower $Q(M)$ when the allocated disk space is less than 30% percent of $S(V)$. However, for disk space allocation more than 30% the $Q(M)$ is same.</td>
</tr>
<tr>
<td>(Zhou, Xu, et al., 2008)</td>
<td>Randomized algorithm outperform than traditional greedy Algorithm.</td>
</tr>
</tbody>
</table>

49
<table>
<thead>
<tr>
<th>References</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fan (1997)</td>
<td>Although the proposed static algorithm performs close to the BPUS algorithm and even executed faster but their dynamic did not act well as expected.</td>
</tr>
<tr>
<td>Ligoudistianos et al. (1998)</td>
<td>Comparing to exhaustive search, The heuristic algorithm explores a small fraction of the search space and gives the near optimal solution in most of the cases. The r-Greedy algorithm explores more states than heuristic algorithm to find the near optimal solution.</td>
</tr>
<tr>
<td>Shah et al. (2006)</td>
<td>Average query and update cost saving of the suggested method is higher than Dynamat (Kotidis &amp; Roussopoulos, 1999). Also, the method requires small number of replacements and eventually makes an optimal balance between query response time and update time.</td>
</tr>
<tr>
<td>Gupta &amp; Mumick (1999)</td>
<td>Experimental results exhibits optimal solution in most of the problem cases and for the other cases it delivers near optimal solutions.</td>
</tr>
<tr>
<td>Gou et al. (2006)</td>
<td>Their theoretical and experimental results show the suggested algorithm is powerful, efficient and flexible to this problem.</td>
</tr>
<tr>
<td>Derakhshan et al. (2006)</td>
<td>Comparing to a heuristic and genetic method the proposed approach provides significant improvement in quality (sum of total query response time and total update time) of obtained solution.</td>
</tr>
<tr>
<td>Kalnis et al. (2002)</td>
<td>Randomized algorithms are applicable to problem with bigger sizes, can adopted to several variations of the problem and provide near-optimal solution in limited time.</td>
</tr>
<tr>
<td>Phuboon-ob &amp; Anupaanwiriyakul (2007a)</td>
<td>Randomized algorithm outperform than traditional greedy algorithm in solving view selection problem. Even, the quality of the solution attained by the synthetic algorithm has more quality than simple genetic algorithm.</td>
</tr>
<tr>
<td>Zhou, Xu et al. (2008)</td>
<td>BPUS perform better than three randomized techniques. However, when the number of dimensions increases, the computational cost of the BPUS algorithm is too high and thus impractical.</td>
</tr>
<tr>
<td>Zhou, Wu, et al. (2008)</td>
<td>The proposed algorithm yields a solution within 90% of the optimal solution quality while exhibits a linear increase in execution time by increasing the number of views.</td>
</tr>
<tr>
<td>Lee &amp; Hammer (1999)</td>
<td>The proposed algorithm performs better than compared algorithms in terms of minimizing total query response time and feasibility of solution.</td>
</tr>
<tr>
<td>Yu et al. (2003)</td>
<td>The proposed algorithm has polynomial time complexity rather than exponential time complexity of BPUS algorithm. In addition, with increasing number of dimensions PGA performs better.</td>
</tr>
<tr>
<td>Nadeau &amp; Teorey (2002)</td>
<td>The proposed algorithm is superior to heuristic and canonical genetic algorithm.</td>
</tr>
<tr>
<td>Wang &amp; Zhang (2005)</td>
<td>Although no comparison to other works carried out but their result shows that GLS can steadily reach to a good solution in few seconds.</td>
</tr>
<tr>
<td>Horng et al. (1999)</td>
<td>The proposed genetic algorithm generates a better solution than the greedy algorithm.</td>
</tr>
<tr>
<td>Lin &amp; Kuo (2004)</td>
<td>The performance of proposed algorithm is within 90% of optimal solution while generates feasible solutions.</td>
</tr>
<tr>
<td>Boukra et al. (2007)</td>
<td>Randomized algorithm outperform than traditional greedy algorithm in solving view selection problem. Even, the quality of the solution attained by the synthetic algorithm has more quality than simple genetic algorithm. In terms of performance and evaluation cost the combination of genetic algorithm and heuristic algorithm works better than using only one of them. However, they concluded that the genetic algorithm outperforms heuristics algorithm.</td>
</tr>
<tr>
<td>Lin &amp; Kuo (2000)</td>
<td>The proposed algorithm is superior to BPUS algorithm.</td>
</tr>
<tr>
<td>Zhou, Xu, et al. (2008)</td>
<td>Randomized algorithm outperform than traditional greedy algorithm in solving view selection problem. Even, the quality of the solution attained by the synthetic algorithm has more quality than simple genetic algorithm.</td>
</tr>
<tr>
<td>Zhang et al. (1999)</td>
<td>The PSO based algorithm reach better performance than traditional algorithms (Heuristic Algorithm and Genetic Algorithm).</td>
</tr>
<tr>
<td>Sun &amp; Wang (2009)</td>
<td>The authors experimentally compared the proposed approach against works in (Harinarayan et al., 1996) and (Shukla et al., 1998b) and delineate the applicability areas of the proposed and compared approaches.</td>
</tr>
<tr>
<td>Talebi et al. (2009)</td>
<td>Their findings show that the solutions which are returned by heuristic</td>
</tr>
</tbody>
</table>
| Agrawal (2005) | }
algorithms are very close to the optimal solution

(Zhang et al., 2009) The proposed MA-Based algorithm works better than heuristic and genetic algorithm

(Horning et al., 2003) The proposed approach performs well in comparison to author’s previous work called YKL97

(Agrawal et al., 2007) heuristic methods find solution close to the optimal solution

### 2.15 Multi-Objective View Selection Problem

When two objective functions needs to be minimized simultaneously, we are dealing with the multi-objective view selection problem (Dhote & Ali, 2009). The single objective view selection problems received significant attention in the past and several heuristic methods proposed for solving this class of problems (see Table 2.3). While the multi-objective view selection problem introduces a broad area of research, it is rarely addressed in the literature. The multi-objective view selection problem is defined as stated in Section 1.3.5.

#### 2.15.1 Related Works for Multi-Objective View Selection Problem

The paper by (Lawrence, 2006) is one of the early papers that considers the multi-objective view selection problem in which both the query response time and the update time needs to be minimized simultaneously under the disk space constraint. All of the previous researches reviewed in this chapter involving both query response time and update time were carried out by converting the pure and original multi-objective problem to the reduced linear combination of two objectives as a single objective problem. In the work by Lawrence (2006), two non-elitist well-known multi-objective evolutionary algorithms, *Multiple Objective Genetic Algorithm (MOGA)* and *Niched-Pareto Genetic Algorithm (NPNGA)* were adopted to solve the view selection problem. In order to deal with constraints two methods have been chosen. The first constraint handling method integrates the constraint into the objective and defines the dominance notation in such way that an infeasible individual is always dominated by a feasible individual. The second one allows the infeasible offspring to be created and utilize a
repair function to convert an infeasible individual to a feasible one. In most of the problem instances these two Multi-Objective Evolutionary Algorithms (MOEA) work similarly but in some cases with high skew the NPGA performs better than other. The experiment shows the proposed algorithm delivers competitive solution in comparison with BPUS. However, the obtained result was not assessed using a performance metric. In addition, monotonicity is an important requirement for greedy heuristics to deliver reasonably good solutions (Bauer & Lehner, 2003; Gupta & Mumick, 2005). However the authors did not present any proofs that the combination of total query response time and total update time as a benefit function in BPUS satisfies the monotonicity property.

### 2.16 Summary

The analytical queries in data warehouse are complex queries which require the aggregation of large numbers of records. One of the common ways for accelerating the analytical queries is using views as a pre-calculated result of queries. Since materializing all possible views is not possible, in practice a subset of views is selected. In selecting views two goals are taken into account: minimizing the total query response time and minimizing the total view update time. The view selection problem can be defined in single objective or multi-objective form.

This chapter started with some background information about the area in which the view selection problem arises. Thereafter, some preliminary principles were presented. Two forms of view selection problems were formally defined, together with a number of related works for each of the forms.
Chapter 3. Evolutionary Multi-Objective Optimization

3.1 Introduction

Optimization is a procedure of finding and comparing different solutions from a set of possible values until no better solution is found. Measuring how good a solution is done by means of an objective function. These objectives for example can be the efficiency of a process, product reliability or the cost of production (Deb, 2001).

When the problem involves optimizing (either minimizing or maximizing) only one objective function, it is called single objective optimization problem. So far a significant amount of study has been devoted to techniques for optimization of single objective problems. These techniques may consist of deterministic search strategies or heuristic based approaches (Coley, 1998).

However, many real-world problems inherently include multiple objectives which must be optimized simultaneously. In some cases the objectives may even be conflicting, that is, trying to optimize one objective in our direction of interest cause the other objective value to change in contrast to the interest and vice versa. This type of optimization problem is called the multi-objective optimization problem. The multi objective optimization can be considered as a general form of single objective optimization problem.

In the presence of multiple objectives, the optimization process cannot concentrate on individual objectives and here we are seeking a set of equal solutions in order to balance
between the multiple objectives although some of them may be conflictive. A particular solution may be a very good solution subject to one objective but returns a poor value for another objective and vice versa. For example when you are planning to buy a laptop computer, two distinct goals are imaginable: computational power of laptop and price of the laptop. One desires to maximize the computation power while minimize the price as much as possible. The objectives are conflictive objectives since by choosing powerful laptops the price increases and by selecting the cheap laptop, the computation power drops. A set of solutions for the problem is shown in Figure 3.1. When only computation power matters solution 5 would be the optimal solution while if price is considered as the only objective, then, the optimal solution is solution 1. In fact, we are interested in solutions that make a good compromise between these two conflictive objectives. Solutions 1 and 5 define the two extreme points, while between these two there are some solutions that form trade-off solutions. All solutions in Figure 3.1 are equally good. Comparing any two of these solutions, when the first solution is better that second solution subject to first objective, it is worse subject to other objective. For example, solution 3 is computationally more powerful than solution 2 but cost more than solution 2.

Figure 3.1 Multi-Objective and Conflictive Optimization Problem
In contrast to single objective problems where a single best solution is desirable, in multiple objective problems there is no single optimal solution and instead we deal with a number of optimal solutions called the *Pareto Optimal Solutions*. The image of pareto optimal solutions in the objective space is called the *Pareto Front* (Talbi, 2009). Similar to pareto optimal solutions, no single solution in the pareto front is preferable to another (Deb, 2001).

Due to lack of suitable methods the earliest approach for solving multi-objective problems were artificially converting the problem into a single objective problem and then applying the methods originally designed for solving single objective problems. However, although, theoretically, all multi-objective problems may be transformed to some form of single objective problems, such reduction ignores the fundamental difference between these two classes of problem (Deb, 2001).

Evolutionary algorithms as nature-inspired methods are now becoming more popular especially in solving complex problems with a large search space. They are based on the Darwinian theory of Evolution (*survival of fittest*) in which the fittest individuals survive and produce the next generation. Evolutionary computation consists of several branches as shown in Figure 3.2 and forms a broad area of research. Many books (Bäck, Fogel, & Michalewicz, 1997; Deb, 2001; Goldberg, 1989; Holland, 1975; Michalewicz, 1996; Mitchell, 1998), conferences (Genetic and Evolutionary Computation Conference GECCO and IEEE Congress on Evolutionary Computation CEC) and journals ('Evolutionary Computation Journal' published by MIT Press, 'Transactions on Evolutionary Computation' published by IEEE and 'Genetic Programming , Evolvable Machines' published by Kluwer Academic Publishers and IEEE Transactions on Systems, Man, and Cybernetics published by IEEE) are dedicated to this topic.
An Evolutionary algorithm uses a population of potential solutions to the problems in each run. In classical or mathematical optimization methods, only one solution is returned after several point-to-point iterations and in each iteration the solution is supposed to be improved as compared to the previous one. In such methods a preference vector is assigned to objectives prior to solving the problem. Since in these methods there is no potential for dealing with several solutions at the same time they may not be appropriate for multi-objective problems. Furthermore some problems are too complex to be well solved by traditional techniques (Coello & Lamont, 2004; Coello, Lamont, & Veldhuizen, 2007; Deb, 2001; Sumathi, Hamsapriya, & Surekha, 2008).

In converse to the classical methods, the population-based feature of evolutionary algorithms allows multiple solutions for the problem to co-exist simultaneously within a single run. This feature is well suited for the nature of multi-objective problem in which a set of solutions are expected. However, in the case of the single objective problem, all members of the population converge to identical solutions to the problem.

Furthermore, since in each population of the evolutionary algorithm multiple solutions to the problem evolved at the same time some sense of parallelism is observed in these algorithms. This capability helps them to be computationally quick in searches (Branke, Deb, Miettinen, & Slowinski, 2008; Deb, 2001, 2010).
In addition, evolutionary algorithms do not require gradient information in their working process. The only knowledge they require is the objective function and therefore can be applied to a variety of applications (Coello & Lamont, 2004).

This chapter presents the principle and fundamentals of multi-objective optimization. Then, the evolutionary methods that are proposed for solving the multi-objective problems are described. Finally, the performance metric for assessment of evolutionary algorithms is explained.

### 3.2 Multi-Objective Optimization Principles

In order to understand multi-objective optimizations a series of useful definitions are required. These definitions provide a background for study and analysis of the multi-objective optimization problems. We start by discussing a formal definition of multi-objective optimization problem.

#### 3.2.1 Multi-Objective Optimization Problem Definition

Each multi-objective optimization problem comes with at-least two objective function, each of which either to be minimized or maximized. Most of the problems include some constraints which restrict the feasible area of solutions. The constraint function can be in an equation or inequality form. Formally the multi-objective optimization problem can be formulated as below:

<table>
<thead>
<tr>
<th>Find the vector</th>
<th>$X^* = (x_1^<em>, x_2^</em>, \ldots, x_n^*) \in \mathbb{R}^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Such that</td>
<td>Minimize/Maximize $f_i(X)$ $i = 1, 2, \ldots, k$</td>
</tr>
<tr>
<td>Subject to:</td>
<td>$m$ inequality constraints: $g_i(X) \leq 0$ $i = 1, 2, \ldots, m$</td>
</tr>
<tr>
<td>And:</td>
<td>$p$ equality constraints: $h_i(X) = 0$ $i = 1, 2, \ldots, p$</td>
</tr>
</tbody>
</table>

where $f_i(X)$ is the $i^{th}$ objective function and $k$ is the number of objective functions. Each objective is a function from $\mathbb{R}^n$ to $\mathbb{R}$, that is, $f_i: \mathbb{R}^n \rightarrow \mathbb{R}$. The set $\mathbb{R}^n$ forms the decision variable space while $\mathbb{R}$ forms the objective space. Each objective takes one
point from the decision space to the objective space (see Figure 3.3) (Coello et al., 2007; Deb, 2001; Engelbrecht, 2007; Talbi, 2009).

The goal is to find a vector $X^*$ that can optimize the objective functions while satisfying the problem constraints. The default inequality constraint is based on less than. However, the greater than form can be converted to less than by multiplying the function with $-1$. Likewise, the maximization objective function can be converted to a minimization objective function. The solution that satisfies all constraints is called the feasible solution.

### 3.2.2 Dominance Relation

Dominance is a fundamental concept in dealing with multi-objective optimization. This relation is used to compare two different solutions with respect to multiple objectives. The dominance concept is defined as follows:

**Definition 3.1.** The solution $X_1$ is said to dominate solution $X_2$ if the following conditions hold:
1. $X_1$ is no worse than $X_2$ subject to all objective functions values

2. $X_1$ is strictly better than $X_2$ subject to at least one objective function.

It is to be noted that worse and better notation depends on the form of the objective functions. In minimization sense the objective function, worse (better) means greater (less) than while in maximization of the objective worse (better) means less (greater) (Alba, Blum, Isasi, Leon, & Gomez, 2009; Deb, 2001; Koziel & Yang, 2011; Talbi, 2009; Tan, Khor, & Lee, 2005).

Given two different solutions $X_1$ and $X_2$ one of these situations happens:

1. $X_1$ dominates $X_2$ or

2. $X_2$ dominates $X_1$ or

3. $X_1$ does not dominate $X_2$ nor does $X_2$ dominates $X_1$

The situation number 3 implies that if $X_1$ does not dominate $X_2$, then $X_2$ does not necessarily dominate $X_1$. As an example, consider the objective space for a two objective problem as shown in Figure 3.4. Both objectives are assumed to be in minimization form. If $f_1$ was the only objective, the solution $X_2$ would be a single global optimal solution. Similarly, in the presence of only objective $f_2$ the solution $X_6$ would be a global optimal solution. Here, we can observe how dominance concept enables us to make a comparison between two different solutions in multi-objective space (Wiak & Juszczak, 2010; Xiaopeng, 2007).

Comparing solution $X_3$ and $X_4$, based on definition 3.1 we can conclude that $X_3$ dominates $X_4$ because:
1. \( f_1(X_3) \succ f_1(X_4) \) and \( f_2(X_3) \succ f_2(X_4) \)

2. \( f_2(X_3) < f_2(X_4) \)

As another instance consider solutions \( X_3 \) and \( X_6 \). \( X_3 \) does not dominate \( X_6 \) because:

- \( f_2(X_3) > f_2(X_6) \)

Thus, condition 1 does not hold. Also, \( X_6 \) does not dominate \( X_3 \) according to condition 1, since:

- \( f_1(X_6) > f_1(X_3) \)

Therefore, solutions \( X_3 \) and \( X_6 \) are considered incomparable or non-dominated by each other. As the dominance concept provides a way for comparing two different solutions subject to multiple objectives, it is used by most multi-objective optimization methods (Burke & Kendall, 2005; Coello et al., 2007; Deb, 2001).
3.2.3 Non-Dominated set of solutions

Let us continue with example of Figure 3.4. The solution $X_3$ does not dominate $X_6$ and $X_6$ also does not dominate $X_3$ thus solutions $X_3$ and $X_6$ are non-dominated with respect to each other because one cannot say which solution is better than the other. In other words, these two solutions are not comparable together. Also, $(X_2$ and $X_3)$, $(X_2$ and $X_6)$ have the same property as well. In order to find the set of all pairs of solutions that have such a property, we can compare all possible pairwise combinations of solutions. For the current example, this set is as indicated below:

$$\{(X_3, X_6), (X_2, X_3), (X_2, X_6)\}$$

Any two solutions in the set $\{X_2, X_3, X_6\}$ do not dominate each other. Any solution that is not included in this set may be dominated by at least one solution in this set. The solutions belonging to this set have preference to all other possible solutions and are called *non-dominated set of solutions*. (Burke & Kendall, 2005; Deb, 2001; Michalewicz & Fogel, 2004)

**Definition 3.2** If $S$ is a set of solutions, the set $N \subseteq S$ is called the non-dominated set of solutions if all solutions that belong to $N$ are not dominated by any solution in $S - N$. The Figure 3.5 shows the non-dominated ($N$) and dominated set ($S - N$) for the mentioned example in Figure 3.4.

$N$ is called the pareto-optimal set if $S$ is the entire search space (Deb, 2001). The image of pareto optimal solutions in the objective space is called *Pareto Front* (Talbi, 2009). The pareto front for two different search spaces has been illustrated in Figure 3.6. On the left part of Figure 3.6 both the objectives are in minimization form while on the right part both objectives are in maximization form.
3.2.4 Non-dominated Sorting (or Pareto ranking)

Although most of the topics on multi-objective evolutionary algorithm discussed in Section 3.3 only require the best non-dominated subset of solutions there exist some algorithms that need classifications of solutions space in several different levels of domination. The best non-dominated set of solutions falls into level 1. In order to find the other levels, first, the solutions of level 1 are removed from the population and then the non-dominated solutions of the remaining population are found by running the same algorithm. These non-dominated solutions form the level 2 solutions. In order to identify level 3 of the solutions, similar to the previous step, the level 2 solutions are removed from the population and then the non-dominated solution finding algorithm is
re-executed. The subsequent levels are also found in the same way until there are no solutions left in the population.

The Figure 3.7 shows the result of non-dominated sorting for the example of Figure 3.4.

3.3 Evolutionary Multi-objective Algorithms

Evolutionary algorithms are search methods inspired from nature to solve complex problem with large search space. Some interesting features of this technique make them popular and motivate researchers to study in diverse areas of applications as alternative to classical methods as described earlier.

This section aims to provide an overview of a number of well-known evolutionary algorithms designed for handling multi-objective optimization problems. Although this section mainly concerns multi-objective algorithms, but, since evolutionary multi-objective algorithms are designed based on the standard single objective genetic algorithm principle, for better understanding, an introduction to simple genetic algorithm is presented first.
3.3.1 Genetic Algorithm

Genetic algorithm (GA) is a computer program that mimics biological evolution in nature where the fittest living organisms will win the competition for available resources and produce the next generation. They are known as a robust search and optimization technique specially for finding approximate solution for complex problems with a large search space. The genetic algorithm concept was first introduced by John Holland in the University of Michigan (Holland, 1975). Today, the field of genetic algorithm and corresponding applications has received significant attention in literature (Gen & Cheng, 1999). Moreover, there are many books (Bäck, 1996; Burke & Kendall, 2005; Coello et al., 2007; Deb, 2001; Gen & Cheng, 1997, 1999; Goldberg, 1989; Haupt & Haupt, 1997; Michalewicz, 1996; Mitchell, 1998; Sivanandam & Deepa, 2009; Yu & Gen, 2010), journal (IEEE Transactions on Evolutionary Computation published by IEEE and Evolutionary Computation published by MIT Press) and conferences (Genetic and Evolutionary Computation Conference (GECCO) and IEEE Congress on Evolutionary Computation(CEC)) devoted to this topic. The flowchart shown in Chapter 1 (Figure 1.1) presents a general procedure for a simple genetic algorithm. A conventional genetic algorithm breaks into several cycles called generations. The initial generation is a population of potential random solutions to the problem. Each solution is called an individual. In each generation all individuals in the population are evaluated by means of a fitness function to measure how good they are. Then a selection mechanism is used to select the fittest individuals. Thereafter, crossover and mutation operators are applied to the selected individual in order to produce the offspring and these offspring forms the new generation. The same process is repeated in the next generation. The evolution terminates once the stopping criteria is met. The evolution cycle is illustrated in Figure 3.8.
Every genetic algorithm consists of the following:

- **Representation (or encoding)**: a way for expressing a real-world problem into a computer data structure
- **Initialization**: a number of initial guesses to the problem
- **Genetic Operators** (selection, mutation and crossover): methods for choosing good solutions, mixing parts of good solution and altering some part of a solution
- **Fitness function**: a way for calculating how good a solution is
- **Termination Condition**: when to stop the execution of the Genetic Algorithm

### 3.3.1.1 Vocabulary of Genetic Algorithm

Genetic algorithm borrows terminology from genetic science. Here, some related terms are defined as following:

- **Phenotype**: a potential real-world solution to the problem. It refers to the observable appearance of an individual. The mapping from genotype to phenotype in called decoding (see Figure 3.9).
- **Individual or chromosome**: representation of problem solution as a computer data structure.
- **Population**: A collection of individuals
- **Genotype**: refers to the genetic structure of individual. Mapping from phenotype to genotype is called encoding (see Figure 3.9)
- **Gene**: the smallest unit of a chromosome is called a gene. The gene encodes a particular feature of organism.
- **Allele**: A specific value for a gene is called the allele.

(Bagchi, 1999; Chakrabarti & Cox, 2008; Coello et al., 2007; Donoso & Fabregat, 2007; Engelbrecht, 2007; Gen & Cheng, 1997; Larose, 2006; Mitchell, 1998; Reeves & Rowe, 2002)

![Figure 3.9 Real World Space versus Genetic Space](image)

**Example 3.1** In order to give a better understanding of how genetic algorithm works we use the following simple single-objective optimization example in the next sections:

Minimize \( f(x) = x^2 \sin(x) \)

Subject to:
\[
\begin{align*}
g_1(x) &= x \leq 1 \\
g_2(x) &= -x \leq 1
\end{align*}
\]

**3.3.1.2 Representation**

In genetic algorithm, representation means encoding a real-world problem solution, called the phenotype, to the computer data structure, called the genotype or chromosome. Representation is regarded as a key issue and fundamental step in designing every genetic algorithm (Gen & Cheng, 1999). The common data structures
are of fixed length array of bits, strings or real values. Amongst these, the array of binary values defined by Holland (1975) is the simplest and most used encoding form (Sivanandam & Deepa, 2009; Zalzala & Fleming, 1997). Choosing a proper representation is an important decision in designing a good genetic algorithm as it affects the performance of the genetic algorithm (Zhang & Tsai, 2007). In fact, the type of representation depends on the problem (Huang, Wunsch, Levine, & Jo, 2008) and may vary from one problem to other problem according to the problem characteristic. The Figure 3.10 shows a structure of a binary encoding for the Example 3.1. An array of binary values is used for encoding a real number variable as a solution to the example problem. The size of the array depends on the expected level of precision. For this example, we allocate 10 binary cells for the array. According to the problem specification, the domain length for the problem is \((-1 \leq x \leq 1\). Thus, the distance between -1 and 1 is divided by \(2^{10} - 1\) equal size steps. The decoding procedure from the array of bits to a real-world solution is as follows:

- Convert the binary number to the decimal number

\[
(b_0, b_1, ..., b_{10})_2 = \sum_{i=0}^{10} b_i \cdot 2^i = y
\]

- Adjust the value \(y\)

\[
x = (-1) + \frac{y}{2^{10} - 1}
\]

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
\end{array}
\]

10 bits

Figure 3.10 A Sample Representation

3.3.1.3 Initialization

Usually the genetic algorithm starts with a population of random generated individuals as a first generation. Therefore individuals in the first population may have a low fitness
value as compared to the consequent generations. The quality of the initial population has a significant impact on the performance of the genetic algorithm. The initial population must be evenly distributed over the search space to increase the population diversity (Li, Jia, Sun, Fei, & Irwin, 2010). However, there are other intelligent alternatives to form a better initial population such as (Miettinen, Neittaanmäki, Mäkelä, & Périaux, 1999):

- A previously saved set of good solutions.
- A set of solutions suggested by a human expert.
- A set of solutions which are returned by a heuristic program.

A sample initial population with 10 individuals for Example 3.1 is shown in Figure 3.11. The population size is an important factor that must be decided during the design of a genetic algorithm (Michalewicz, 1996). It indicates the number of individuals which exists within a population. Too small a population size lacks diversity and may cause the genetic algorithm to get stuck in the local optima by converging too quickly and may prevent the genetic algorithm to reach the global optimal solution. On the other hand, too large a population size slows down the genetic algorithm since the computation time will then increase (Ahn, 2006; Raphael & Smith, 2003; Tzafestas, 1999). Thus, the population size must be carefully tuned to a tradeoff value between efficiency and effectiveness (Reeves & Rowe, 2002).

![Figure 3.11 A sample random initial population with 10 members](image)
3.3.1.4 Operators

The genetic algorithm includes three operators called selection, crossover and mutation. The operators are applied to individuals of current population in order to produce a new offspring for the new generation. Figure 3.12 shows how the genetic operators act on the current population. In this section we will describe the selection, crossover and mutation operators.

- Selection

Selection is a process of choosing two parents to breed the new offspring based on the Darwinian principle of natural selection (Gen & Cheng, 1999). Normally, the fitter individuals are given more chance to be selected since they are more likely to produce good children. The selection pressure is defined as the degree of tendency (or probability) to select the best individuals in the population. With more selection pressure more individuals with higher fitness values are favored by the selection operator. If the selection pressure is too small the convergence slows down and if it is too high the genetic algorithm may prematurely converge to a local optimal solution and fail to reach the global optimal solution. In fact, a proper balance between these two is required. The magnitude of the selection pressure has a significant effect on the convergence speed of the genetic algorithm (Diaz-Gomez, 2007; Haupt & Haupt, 1997; Kaylani, 2008; Sas Institute, 2003; Sivanandam & Deepa, 2009; Vonk, Jain, & Johnson, 1998). Often, a low pressure is selected in the early stages of evolution to cover wide parts of the search space and at the end of the evolution the selection pressure is decreased to narrow the search space (Gen & Cheng, 1999). Several forms of selection have been proposed but common methods of selection include the random selection, roulette–wheel selection, tournament selection and ranking selection (Alba & Dorronsoro, 2008; Blickle, 1997; Eiben & Smith, 2008; Engelbrecht, 2007; Freitas, 2002; Gendreau & Potvin, 2010; Gorunescu, 2011; Haupt & Werner, 2007; Lee & El-
Sharkawi, 2008; Reeves & Rowe, 2002; Shukla, Tiwari, & Kala, 2010; Sivanandam & Deepa, 2009; Talbi, 2009; Yu & Gen, 2010).

Crossover (Recombination)

Crossover is a genetic operator which takes two parent individuals as operand and exchange parts of them to produce two offspring which share some features with their parents (Negnevitsky, 2004). The crossover operator comes in several forms such as single point, multi-point and uniform. In the single point crossover in its traditional and simplest form, a single position of the chromosome, called the crossover point, splits the entire individual into two parts and the parts after the point are swapped to create two new offsprings. Figure 3.13 illustrates the single point crossover for two individuals which are designed for Example 3.1. In the multi-point crossover, multiple crossover points are used to divide the chromosome into more than two parts and corresponding
parts from the two individuals are exchanged. For instance, in a two point crossover as shown in Figure 3.14, two crossover points divide the chromosome into three parts and the middle part located between two points are exchanged between two parents (Achenie, Venkatasubramanian, & Gani, 2002; Laplante, 2003; Sivanandam & Deepa, 2009).

The crossover point is often randomly chosen between 1 and \( l \) where \( l \) is the length of the chromosome (Laplante, 2003; Sivanandam & Deepa, 2009).

Generally, the crossover operation is performed with the probability of \( P_c \) which is also called the crossover rate. Normally, the crossover rate is set to more than 0.5. If there are no crossovers the children are exact copies of their parents (Haupt & Haupt, 1997; Mitchell, 1998; Sivanandam & Deepa, 2009; Sumathi et al., 2008; Yu & Gen, 2010).

- **Mutation**

Mutation as unary operator is applied to each offspring after crossover. The operator alters a gene value in random position of a chromosome. Mutation rarely happens in nature, similarly, in GA, the mutation operator takes place with only a small probability,
\( P_m \), called the mutation rate, typically between 0.001 to 0.01 (Negnevitsky, 2004). Figure 3.15 shows how the mutation operator modifies the individual of Example 3.1. The mutation operator (Gong & Zhao, 2008) acts like a random walk in the search space (Pedrycz & Gomide, 1998) and plays a significant role in maintaining population diversity (Engelbrecht, 2007; Sivanandam & Deepa, 2009). The goal of the mutation operator is to extend the search space by introducing a new solution which has not been discovered before and thus prevents the genetic algorithm to trap into a local optima (Negnevitsky, 2004). If crossover is responsible for the exploitation of the characteristics of the parents in order to obtain better children, then, the mutation operator is responsible for the exploration of the search space for diversity (Sivanandam & Deepa, 2009). For a binary representation, the uniform mutation is performed by flipping gene values at randomly chosen positions as illustrated in Figure 3.15 (Engelbrecht, 2007).

![Mutation Operator](image)

**Figure 3.15 Mutation Operator**

### 3.3.1.5 Fitness Function (or evaluation function)

The fitness function is used to measure how good an individual is. The greater the fitness value, the smaller is the distance of the solution to the optimal solution.

In the case of a single objective optimization, the fitness function may have a direct relationship to the problem objective (with some adjustments). In order to calculate the fitness value for an individual its genotype is first converted to the phenotypic equivalent. Then the fitness function maps the phenotype to a real number. That real number is the fitness of the individual. In other words, the individuals are evaluated using their phenotype but not their genotype (Chakrabarti & Cox, 2008; Lee & El-Sharkawi, 2008; Rennard, 2006; Yanushkevich, 2004; Ziman, 2003).
For the Example 3.1 the fitness function can be defined as following:

\[
\text{Fitness}(x) = f(x) = x \cdot \sin(x)
\]

However, in contrast to the single objective problems, in multi-objective optimization problems, defining fitness function so that individuals in a population can be compared is not an easy task (Sivanandam & Deepa, 2009). A simple approach is to convert the multiple objectives into a single objective and then treat the problem as a single objective problem.

Since the fitness function must be calculated thousands of times (for every chromosome in the population at each generation), and is therefore regarded as a computational bottleneck of the genetic algorithm, it is recommended that the function be a fast computed one and thus should not be a computationally complex function. (Champandard, 2003; Goodman, 2009). Conventionally, the fitness function returns a positive value. However, in case the value is negative the fitness function can be adjusted by some fitness scaling methods such as: *Linear Scaling*, *Sigma Scaling* or *Power Law Scaling* (Lee & El-Sharkawi, 2008).

### 3.3.1.6 Termination Condition

Evolution in nature never stops but in computer we need to stop the genetic cycle sometime (Yu & Gen, 2010). The criteria for stopping genetic algorithm is called the *termination condition* (Cox, 2005). The genetic algorithm stops when at least one of the pre-defined termination conditions are satisfied. The condition may be the following items (Chen, 2002; Engelbrecht, 2007; Gen & Cheng, 1999; Reeves & Rowe, 2002; Sivanandam & Deepa, 2009; Yu & Gen, 2010):

- Reach a satisfactory result
- Maximum number of generations exceeded.
- Elapsed running time exceeded the predetermined value.
- Small amount of improvement observed in last generations.
• Fixed number of fitness evaluation reached
• Population convergence (when fitness values for all individuals within population are identical)

3.3.2 Elitism

Evolutionary multi-objective optimization algorithm can be classified into two main groups; non-elitist algorithms and elitist algorithms. The idea of elitism was first introduced by Jong (1975). Elitism is a mechanism to preserve good individuals of the current generation by saving them in a separate secondary population called an archive and forwarding them to the next generation (Bui & Alam, 2008; Drechsler & Drechsler, 2002). The archive stores a number of best solutions encountered since the start of the execution of the genetic algorithm (Talbi, 2009). Elitism ensures that good solutions that has been found will not be lost unless a better solution is discovered (Deb, 2001).

The addition of the elitism feature to the evolutionary multi-objective optimization provides a monotonically non-decreasing performance (Branke et al., 2008; Talbi, 2009). Using the elitism capability, it has been demonstrated that the genetic algorithm converges to the global optimal solution in some problems (Rudolph, 1996).

In the single objective optimization, the identification of an elite solution from the population is an easy task. The individual with the highest fitness value (for maximization problem) will be selected as the elite solution. However, discovering elite solutions in the presence of multiple objectives is not as simple as the single objective case. In such area, the concept of domination as a remedy enables us to sort the individuals in the population to different groups. The individuals who form the first group of the non-dominated set are considered as the elite solutions. Note that in contrast to the single objective optimization where there is only one elite solution, here we deal with a set of elite solutions which are equally important (Deb, 2001).
In the non-elitist algorithm no explicit form of keeping the best found individuals for the next generation is foreseen while the elitist-algorithms take advantage of such feature in order to reach a faster convergence toward the pareto front and more precise approximation of the pareto front shape (Talbi, 2009).

3.3.3 Non-Elitist Algorithms

The evolutionary multi-objective algorithms tend to be discussed hereafter are divided into two section: non-elitist algorithms and elitist algorithms. In the current section the algorithms, Weight Based Genetic Algorithm (WBGA), Vector Evaluated Genetic Algorithm (VEGA), Non-dominated Sorting Genetic Algorithm (NSGA), Niched Pareto Genetic Algorithm (NPGA) and Multi-Objective Genetic Algorithm (MOGA) is explained.

3.3.3.1 Weight Based Genetic Algorithm

The Weight Based Genetic Algorithm (WBGA) (also called HLGA) was proposed by Hajela and Lin in (Hajela & Lin, 1992). In this method, the objective functions, $f_i$ is multiplied to a weight coefficient $w_i$ to form a weighted sum of objectives. However, in contrast to the simple weighted sum approach where a predefined fixed weight vector is used, WBGA encodes a weight vector to each chromosome in addition to the normal decision variable and allows weight coefficients to be evolved as well. Therefore, instead of finding a single solution for a fixed weight vector a population of individuals with variable weight vectors is maintained in parallel to reach the diverse set of pareto optimal points in a single run. The diversity of solutions is preserved in WBGA in two ways: In the first way a niching method is applied to the weight vector part of the individual while in the second way, selected subpopulations are evaluated based on predetermined weight vectors similar to the VEGA approach. The advantage of WBGA is its low complexity and only a minor modification of the simple objective approach is required to convert the simple objective method to WBGA. The disadvantage of the
algorithm is when one objective is in minimization form and other objective is in maximization form the fitness function becomes unduly complex (Coello et al., 2007; Deb, 2001; Konaka, Coit, & Smith, 2006; Tan et al., 2005; Zitzler & Thiele, 1999).

3.3.3.2 Vector Evaluated Genetic Algorithm

Vector Evaluated Genetic Algorithm (VEGA) (Schaffer, 1985) was first introduced by Shaffer in the mid-1980s (1984, 1985). VEGA is considered as the early efforts of using genetic algorithm for solving multi-objective optimization problems (Nedjah & Mourelle, 2005). VEGA differs from a simple genetic algorithm only in the way the selection is performed (Nedjah & Mourelle, 2005; Sarker, Mohammadian, & Yao, 2002). The main idea of VEGA is to divide the population into several subpopulations equal to the number of the objective functions. That is, for a problem with $k$ objective functions and population size of $m$, the whole population is divided by $k$ sub-populations of size $\frac{m}{k}$ individuals. Then, evolution takes place for all sub-populations in parallel. That is, every sub-population is evaluated based on a single corresponding objective and the roulette wheel selection is also performed on the same objective.

The selected individuals of each subpopulation then form a mating sub-pool. All sub-pools merged together and the entire population is shuffled and then the crossover and mutation operators are applied on it. After applying the operators, the resultant offspring forms the new generation. A schematic of the way in which VEGA works is shown in Figure 3.16.
VEGA is straightforward and easy to implement (Zitzler, Deb, Thiele, Coello, & Corne, 2001), however, the solution returned by VEGA are locally non-dominated and not necessarily pareto front (Nedjah & Mourelle, 2005). Since VEGA emphasizes each objective their results have a tendency to be in the vicinity of the minimum for each individual objective and therefore it may be unable to deliver tradeoff solutions subject to all objective functions (H. Nakayama, Z. Yun, & M. Yoon, 2009b). Furthermore, VEGA behaves like aggregating methods and thus, shares the problems of such techniques (Zitzler, Deb, et al., 2001).

### 3.3.3.3 Non-dominated Sorting Genetic Algorithm

Non-dominated Sorting Genetic Algorithm (NSGA) (Srinivas & Deb, 1994) was introduced by Srinivas and Deb in 1995. The approach is based on the classification of the entire population into several layers according to the domination concept (Goldberg, 1989). The algorithm starts by initial randomly generated individuals. Among these individuals the non-dominated set is identified and forms the first non-dominated front and they are given a dummy fitness value equal to the population size; \textit{popsize}; in order to give same reproductive chance to all individuals that exist in the first front (Michalewicz, 1996). In order to maintain the diversity of the population, a fitness sharing method (Deb, 2001; Eiben & Smith, 2008; Engelbrecht, 2007; Gen & Cheng, 1999; Mumford & Jain, 2009; Nakayama et al., 2009b; Yu & Gen, 2010) is used to give different fitness value to individuals of the same rank. Thereafter, the first front individuals are removed temporarily from the population (or ignored) and the non-
dominated set of individuals within the rest of the population is determined. Again, the
second non-dominated front’s individual are assigned a new dummy fitness value less
than the dummy fitness value given to the first front. The smaller value is assigned to
reflect the superiority of the first front’s individuals to the second front’s individuals.
The procedure is repeated until the entire population is classified into several distinct
layers. After classification has been completed, the \textit{stochastic remainder selection} is
adopted to select individuals according to their shared fitness. Then, crossover and
mutation are performed. The advantage of \textit{NSGA} is its fitness assignment based non-
dominated levels. However \textit{NSGA} is considered as a computationally complex
algorithm because of the ranking and fitness sharing procedures (Coello et al., 2007;
Deb, 2001; Lee & El-Sharkawi, 2008).

\textbf{3.3.3.4 Niched Pareto Genetic Algorithm}

The \textit{Niched Pareto Genetic Algorithm (NPGA)} was proposed by Horn et al in (Horn,
Nafpliotis, & Goldberg, 1994). Different selection strategy has been used in this
algorithm. In contrast to \textit{VEGA}, \textit{NSGA} and \textit{MOGA} where the proportionate selection
method is used, the authors preferred to adopt a tournament as well as dominance
concept. In \textit{NPGA}, first, a subpopulation of entire population; \( S \); (typically 10 percent of
the main population) is picked from the main population. Consequently, two random
competitor individuals \( a \) and \( b \) are drawn from the main population. Then individuals \( a \)
and \( b \) are compared to each individuals in the subpopulation for domination. After
comparison there are four different results imaginable:

1. \( a \) dominated by at least one individual in the subset \( S \) but \( b \) is not dominated
2. \( b \) dominated by at least one individual in the subset \( S \) but \( a \) is not dominated
3. \( a \) and \( b \) both dominated by at least one individual in the subset \( S \)
4. \( a \) and \( b \) both are non-dominated subject to the subset \( S \)
In the case of situation 1 individual $b$ is chosen and in the case of situation 2 individual $a$ is chosen.

When situation 3 or 4 happens, the tie is broken by comparing both individuals $a$ and $b$ to the partially filled offspring population. Each individual is placed in the offspring population and the *niche count* is calculated for them. Finally, the individual with the smaller niche count value is the winner of the tournament. The selection mechanism of *NPGA* is shown in Figure 3.17. The advantage of *NPGA* is that it does not need any direct fitness assignment. The disadvantage of *NPGA* is the need for setting proper values for the parameters of $t_{dom}$ and $\sigma_{share}$ which may affect the performance of the algorithm.

(Coello et al., 2007; Deb, 2001; Nedjah & Mourelle, 2005).

![Figure 3.17 NPGA Selection Mechanism](image)

### 3.3.3.5 Multiple Objective Genetic Algorithm

The Multiple Objective Genetic Algorithm (*MOGA*) Algorithm was introduced by Fonseca and Fleming in (Fonseca & Fleming, 1993). In *MOGA*, a rank is assigned to each individual equal to the number of individuals in the population by which it is dominated. For example, if the individual $I$, is dominated by $t$ different individuals in
the population then $Rank(I) = t + 1$. Therefore, all non-dominated individuals receive the same rank value 1 since they are not dominated by any other individual. In this way, the individual dominated by more individuals receives a higher rank. The rank itself is not used as a fitness function. Instead, an efficiency function is defined based on the individual’s rank. The efficiency function can be calculated as below:

- Sorting the individuals based on their ranks
- Use a linear function; $f(rank)$ as fitness function in order to assign an efficiency value to each individual. The efficiency function is often in linear form but it is not necessarily. The $f(rank)$ must satisfy following condition:

$$\text{If \ } rank_a < rank_b \text{ \ then } f(rank_a) > f(rank_b)$$

The advantage of MOGA is its simple fitness assignment and it can be easily applied to different optimization problems because the niching is done in the objective space. Despite of the dominance concept which is used in this algorithm, there is a strong possibility that the algorithm may bias towards specific solutions in the search space. Moreover, the algorithm might be sensitive to the shape of the pareto front.

(Bagchi, 1999; Coello et al., 2007; Collette & Siarry, 2003; Erickson, Mayer, & Horn, 2002; Nedjah & Mourelle, 2005; Tan et al., 2005)

### 3.3.4 Elitist Algorithms

In this section, the evolutionary algorithms which use the elitism concept will be described. These algorithms are Non-dominated Sorting Genetic Algorithm II (NSGA-II), Strength Pareto Evolutionary Algorithm (SPEA) and Strength Pareto Evolutionary Algorithm (SPEA-II). Elitism ensures that the quality of the solution never degrades during the evolution process from the current generation to the next generation. Elitism accelerates the convergence of the population toward a pareto front (Bui & Alam, 2008; Engelbrecht, 2007; Talbi, 2009; Vonk et al., 1998).
3.3.4.1 Non-dominated Sorting Genetic Algorithm II

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) algorithm was proposed by Kalyanmoy Deb and others in (Deb, Agrawal, Pratap, & Meyarivan, 2000). The procedure for one iteration of the NSGA-II is illustrated in Figure 3.19. The algorithm starts by generating an initial population of random individuals with a predetermined size known as \( \text{popsize} \). Then the offspring population with identical size is produced by applying the usual operators (selection, crossover and mutation). Then, the offspring population is combined to the parent population to form a whole population with size \( 2 \times \text{popsize} \). Thereafter, the non-dominated sorting algorithm (Goldberg, 1989) is used to classify the entire population into several hierarchical fronts. The fronts are assigned a label \( 1, 2, \ldots, N \) such that the best non-dominated front receives label 1 and worst one receives \( N \). Once, the classification is completed, the new population is filled by inserting the front’s individuals starting from the best non-dominated fronts (with label 1) and with the increasing order of label values. Since, the size of the accumulated population is twice \( \text{popsize} \), it is not possible to accommodate all fronts to the new population. When there is no room, the remaining fronts are simply eliminated. However, for the last feasible front, there still may exist some individuals which cannot be accommodated. In that case the last allowed front is partially inserted to the new population. Instead of simply deleting extra individual from the last front it would be wise to select those individuals that help diversify the new population. Hence, the individuals of the last allowed front are sorted based on descending order of a metric which is called the crowding distance. The crowding distance for an individual is a measure to determine crowding by other individuals in the same front. It is an estimate for the density of solutions surrounded by them. As shown in Figure 3.18 the crowding distance for individual \( i \) is defined as a half perimeter of cuboid formed by the closest left and right neighboring individuals that encompass individual \( i \). The advantage of the
algorithms is that it does not need setting for niching parameters like $\sigma_{share}$. The disadvantage of this algorithm is that the crowded comparison as means of limiting the size of population weaken its convergence power (Benyoucef & Grabot, 2010; Deb, 2001; Drechsler & Drechsler, 2002; Nakayama et al., 2009b; Nedjah & Mourelle, 2006; Sarker et al., 2002; Talbi, 2009; Yu & Gen, 2010).

![Figure 3.18: Crowding Distance Calculation](image)

![Figure 3.19: Schematic of NSGA-II procedure](image)

### 3.3.4.2 Strength Pareto Evolutionary Algorithm

The Strength Pareto Evolutionary Algorithm (SPEA) was designed by Zitzler and Thiele (1999). SPEA is considered as a combination of several multi-objective optimization algorithms. The method stores non-dominated individuals discovered since the beginning of the algorithm and is continuously updated with a fixed size external
population. At each generation, the external population is updated by copying the newly found non-dominated individuals in the current generation. Once the size of the external population exceeds a pre-determined amount, it is pruned to reach a standard size by using a clustering technique called the *average linkage method* (Morse, 1980). After the termination of the algorithm individuals placed in the external population forms the output of the algorithm. A binary tournament selection is used to choose individuals with the smaller fitness from both main and external population. The strength value; $S(i)$; similar to rank used in MOGA is assigned to each individual within the external population. The strength value of an individual in the external population is a real value in $[0,1)$ and is proportional to the number of individuals in the main population it dominates. The strength value for the member $i$ of the main population is calculated as the following:

$$S(i) = \frac{n_i}{\text{popsize} + 1}$$  \hspace{1cm} (3.2)

where $n_i$ is the number of individuals in the main population dominated by individual $i$. On the other hand, the fitness value; $F(j)$; for an individual $j$ in the main population is computed as the sum of the strength values of individuals in the external population dominated by them plus one and calculated as the following:

$$F(j) = 1 + \sum_{i \in P' \land i \succeq j} S(i)$$  \hspace{1cm} (3.3)

where $P'$ represents the external population and $i \succeq j$ means individual $j$ (in main population) is dominated by individual $i$ (in external population). The addition of 1 ensures that the fitness of any member within the main population is greater than the fitness of any external population member and therefore the external individuals always have a higher fitness value. Figure 3.20 illustrates the SPEA fitness assignment for a number of sample solutions (Barba, 2009; Coello & Lamont, 2004; Coello et al., 2007; Deb, 2001; Mumford & Jain, 2009; Nedjah & Mourelle, 2005; Talbi, 2009).
The advantage of $SPEA$ algorithm is in its parameter-less clustering technique which provides a better spread among non-dominated solutions. Moreover, the individuals’ fitness in $SPEA$ can be easily calculated (Deb, 2001).

### 3.3.4.3 Strength Pareto Evolutionary Algorithm II

The Strength Pareto Evolutionary Algorithm II ($SPEA$-II) was suggested by Zitzler (Zitzler, Laumanns, & Thiele, 2001) as an improvement to the original $SPEA$. Three major enhancements have been made to its predecessor:

- Incorporate a fine-grained fitness assignment for each individual taking into consideration the number of individuals it dominates and the number of individuals dominated by them.
- A density estimation technique
- Enhanced archive population truncation method.

In $SPEA$, the individuals which are dominated by the same archive members have equal fitness values (for example solutions $X_4$ and $X_5$ in Figure 3.20). To avoid such situations in $SPEA$-$II$, in calculating the fitness for each individual both dominating and dominated solutions are taken into consideration. For each individual $i$ in the union of the main population and the archive population (represented by $P + \bar{P}$) the strength
value; \( S(i) \); is computed as the number of individuals in \( P + \bar{P} \) which are dominated by \( i \) as stated in the following equation:

\[
S(i) = |\{j | j \in P_t + \bar{P}_t \land i > j \}|
\]

\(| \) represents the cardinality or the number of the elements in the set and \( > \) indicates the dominance relation (i.e. \( i \) dominates \( j \)). A larger value of \( S(i) \) indicates that individual \( i \) is stronger. Thereafter, the raw fitness value for individual \( i \); \( R(i) \); in \( P + \bar{P} \) is calculated as a sum of the strength values of individuals in \( P + \bar{P} \) which dominate \( i \) and is expressed by the following equation:

\[
R(i) = \sum_{j \in P_t + \bar{P}_t, j \succ i} S(j)
\]

Note that \( R(i) = 0 \) implies that the individual \( i \) is a non-dominated solution. A higher value of \( R(i) \) means that individual \( i \) is dominated by more individuals and thus the fitness is to be minimized. Figure 3.21 shows the strength and raw fitness for a set of solution.

Figure 3.21 SPEA-II Strength And Raw Fitness for a Set of Solutions
In order to discriminate between the individuals that have the same fitness value, a density estimation is adopted. The density value (Zitzler, Laumanns, et al., 2001); \( D(i) \); is calculated as follows:

\[
D(i) = \frac{1}{d_i^k + 2}
\]

where \( k = \sqrt{\text{popsize} + \bar{\text{popsize}}} \) and \( d_i^k \) is the distance of individual \( i \) to the \( k^{th} \) nearest neighbor and \( \text{popsize} \) and \( \bar{\text{popsize}} \) are sizes of the main and archive populations respectively. Then the density used to convert the raw fitness to the new fitness value is the following:

\[
F(i) = D(i) + R(i)
\]

The archive population in SPEA-II has a constant size over time and whenever the size goes beyond the predetermined value a truncation method is used to decrease the size of the archive population. During the selection, all non-dominated solution in the main and archive population with fitness value less than 1 are moved to the archive population as expressed by the following equation:

\[
\bar{P}_{t+1} = \{i| i \in P_t + \bar{P}_t \land F(i) < 1\}
\]

If the non-dominated solutions fit the archive exactly, that is ( \(|\bar{P}_{t+1}| = \text{popsize} \) ) no more task is performed; otherwise if the archive population is too large, that is ( \(|\bar{P}_{t+1}| > \text{popsize} \) ) the archive truncation procedure is invoked.

(Drechsler & Drechsler, 2002; Gandibleux, Sevaux, Sörensen, & T’Kindt, 2004; H. Nakayama, Y. Yun, & M. Yoon, 2009a; Yu & Gen, 2010; Zitzler, Laumanns, et al., 2001)

The main advantage of SPEA-II is in its strong performance in diversity and convergence (Zheng, Ling, Shi, & Xie, 2005).
3.3.5 Constraint Handling

The existence of constraints in real-world problems motivates researchers to pay special attention in dealing with constraints. The constraints divide the entire search space by two regions: feasible region and infeasible region as illustrated in Figure 3.22. The optimal solutions are desirable only from the feasible region. Since not all individuals in a population may be a feasible solution we need to devise a method in order to deal with infeasible solutions.

In this section a technique for tackling constraints in evolutionary multi-objective optimization is introduced. (Coello et al., 2007; Deb, 2001; Engelbrecht, 2007; Talbi, 2009)

![Search Space, Feasible and Infeasible Regions](image)

**Constrained dominance**

Deb (2000) proposed a new technique for handling constraints which is well suited for evolutionary multi-objective optimization. The technique is a modification of the standard dominance concept and does not need any penalty function. They combined dominance concept and feasibility check to define a constrained dominance concept. Two solutions can be feasible or infeasible while at the same time they may or may not
dominate each other. The constrained dominance concept for two solutions $x$ and $y$ is defined as follows:

**Definition:** The solution $x$ is said to *constraint dominate* solution $y$ if any of the following conditions hold:

a) Solution $x$ is feasible and solution $y$ is infeasible
b) Both $x$ and $y$ are not feasible solutions but $x$ have less constraint violation.
c) Both $x$ and $y$ are feasible solutions but solution $x$ dominates solution $y$ according to the normal dominance concept presented in definition 3.1

In this method, two solutions are picked from the population and a better solution in terms of constrained dominance is selected as the winner.

All algorithms based on the prior definition of dominance, such as the *NSGA*, which has been described in Section 3.3 can still work with constraint dominance. The only required change is the replacement of new dominance definition.

(Branke et al., 2008; Deb, 2001; Freschi & Repetto, 2005; King & Rughooputh, 2003; Mezura-Montes, 2009; Yu & Gen, 2010)

**3.3.6 Applications of Evolutionary Multi-Objective Algorithms in Other Areas.**

Apart from view selection problem, evolutionary multi-objective algorithms have been applied to a variety of optimization problems in different areas. Table 3.1 shows the application of evolutionary multi-objective algorithm in a number of problems in different areas.
### Table 3.1 List of some other Applications of Evolutionary Algorithms

<table>
<thead>
<tr>
<th>Area</th>
<th>Algorithm(s) Used</th>
<th>Paper(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gas supply network</td>
<td>VEGA</td>
<td>(Surry, Radcliffe, &amp; Boyd, 1995), (Surry &amp; Radcliffe, 1997)</td>
</tr>
<tr>
<td>Allocation in radiological facilities</td>
<td>MOGA, NPGA and SPEA</td>
<td>(Lahanas, Milickovic, Baltas, &amp; Zamboglou, 2001)</td>
</tr>
<tr>
<td>Data Mining</td>
<td>SPEA-II</td>
<td>(Hetland &amp; Sætrom, 2005)</td>
</tr>
<tr>
<td>Design of electromagnetic devices</td>
<td>NPGA, NSGA</td>
<td>(Weile, Michielssen, &amp; Goldberg, 1996)</td>
</tr>
<tr>
<td>Design of an electromechanical system</td>
<td>NSGA-II</td>
<td>(R’egnier, Sareni, &amp; Roboam, 2005)</td>
</tr>
<tr>
<td>Design of combinational circuits</td>
<td>VEGA</td>
<td>(Coello, Aguirre, &amp; Buckles, 2000), (Luna, Coello, &amp; Aguirre, 2004), (Luna &amp; Coello, 2004)</td>
</tr>
<tr>
<td>Network Design</td>
<td>Modified NSGA-II</td>
<td>(Kleeman, Lamont, Hopkinson, &amp; Graham, 2007)</td>
</tr>
<tr>
<td>Multicast flows</td>
<td>SPEA</td>
<td>(Meisel, 2005)</td>
</tr>
<tr>
<td>Design of a thermal system for a building</td>
<td>MOGA</td>
<td>(Wright &amp; Loosemore, 2001) (Wright, Loosemore, &amp; Farmani, 2002)</td>
</tr>
<tr>
<td>Road systems</td>
<td>NPGA</td>
<td>(Haastrup &amp; Pereira, 1997)</td>
</tr>
<tr>
<td>Aerodynamic optimization</td>
<td>NSGA-II</td>
<td>(Nariman-Zadeh, Atashkari, Jamali, Pilechi, &amp; Yao, 2005)</td>
</tr>
<tr>
<td>Treatment planning</td>
<td>SPEA</td>
<td>(Petrovski &amp; McCall, 2001)</td>
</tr>
</tbody>
</table>

### 3.4 Performance Metrics (indicators)

Although in the early years of evolutionary multi-objective optimization visual comparison between the obtained solutions and the optimal set of solutions in objective space seemed to be sufficient for evaluating algorithms but with the emerging number of evolutionary algorithms in recent years, there is a greater need for an evaluation tool. Two distinct goals for multi objective optimization are the diversity of solutions and the convergence toward the true pareto front. The first goal refers to how a set of found solutions are well distributed along the pareto front and the second goals says how well the solutions have converged toward the true pareto front. Figure 3.23 illustrates these two goals for a hypothetical search space. In designing the performance metrics for evolutionary algorithms these two goals must be taken into account. These two goals are sometimes conflictive. An algorithm may return a well distributed set of solutions while the solutions do not converge well towards the pareto front. In the other algorithm, the opposite situation may happen. In fact, obtaining the algorithm which
optimizes both of these goals, can be regarded as another multi-objective optimization (Deb, 2001). In this section a number of performance metrics which have been proposed for the assessment of multi-objective optimization algorithms are explained. These metrics can be classified into two categories (Branke et al., 2008; Talbi, 2009):

1. The metrics designed to measure diversity of solutions
2. The metrics are meant to measure convergence
3. The metrics measure both the diversity and convergence goals

Please note that some metrics require the knowledge of the pareto optimal set while in others there is no need to access the pareto optimal set. However, in most real-world optimization problems the true pareto optimal set is unknown (Talbi, 2009) unless using a brute force algorithm to search the entire search space for a long time. Since the true pareto optimal set is not available in the view selection problem we exclude the metrics which require the set.

Figure 3.23 Convergence and Diversity
3.4.1 Set Coverage (C)

(Zitzler, 1999) proposed a binary metric for comparing the performance of two algorithms. Having two sets of solutions; A and B; the set coverage metric measures the percentage of solutions in B which are dominated by at least one solution in A. The metric is represented as follows:

\[ C(A, B) = \frac{|\{ b \in B | \exists a \in A : a \succ b \}|}{|B|} \]

Equation 3.9

\( C(A, B) = 0 \) means that no solution in B is dominated by solutions in A; likewise \( C(A, B) = 1 \) indicates that all the solutions in B are dominated by at least one solution in A. Since the domination relation is not a symmetric operator, in which \( C(A, B) \neq 1 - C(B, A) \) therefore both \( C(A, B) \) and \( C(B, A) \) should be calculated separately.

(Coello et al., 2007; Deb, 2001; Janssens & Pangilinan, 2010; Yu & Gen, 2010)

Figure 3.24 Ideal Value for Coverage Metric

Figure 3.24 shows the two sets of solutions, A and B. All solutions in set B are dominated by at least a solution in set A. In this condition the maximum value for \( C(A, B) \) is obtained as 1
3.4.2 Spacing (SP)

This metric was introduced by Schott (1995) and measures how solutions are uniformly distributed. *Spacing* calculates the standard deviation of distances between consecutive solutions. The metric is represented as follows:

\[
SP = \frac{1}{|Q|} \sum_{i=1}^{|Q|} (d_i - \bar{d})^2
\]

where:

\[
d_i = \min_{k \in Q \land k \neq i} \sum_{m=1}^{M} |f_m^i - f_m^k|
\]

\(d_i\) represents the minimum distance between solution \(i\) and any other solution in the obtained set. Figure 3.25 illustrates \(d_i\) for a set of obtained solutions.

\(\bar{d}\) is average of all \(d_i\) distances and is calculated as follows:

\[
\bar{d} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} d_i
\]

A small \(SP\) indicates more equally spaced solutions. \(SP = 0\), means that the solutions in \(Q\) are evenly distributed. That is, the distance between consecutive solutions in the obtained set is identical. Figure 3.26 shows a set of solutions with *Spacing* equal to zero.

---

Figure 3.25 Distances between Neighboring Solutions in Set of Obtained Solutions

Figure 3.26 Set of Solutions with *Spacing* Equal to Zero
(Abraham & Goldberg, 2005; Coello et al., 2007; Deb, 2001; Goh, Ong, & Tan, 2009; Janssens & Pangilinan, 2010; Tan et al., 2005)

3.4.3 Maximum Spread (MS)

(Zitzler, 1999) proposed a metric which calculates the length of the diagonal of the hyperbox formed by extreme solutions in the obtained set as follows:

\[
MS = \sqrt{\sum_{m=1}^{M} \left( \max_{i=1}^{t} f_m^i - \min_{i=1}^{t} f_m^i \right)^2}
\]

In the case of two objective optimization problems the metrics is equal to the Euclidian distance between the two extreme solutions in each objective. As an example, Figure 3.27 illustrates the maximum spread for a set of the discovered non-dominated solutions:
The larger the maximum spread, the better the values are, since it implies that the obtained solution set are spanned along a larger part of the pareto front. However, the Maximum Spread metric does not measure the uniformity of intermediate solutions.

(Alberto & Mateo, 2008; Deb, 2001; Tan et al., 2005)

The maximum possible extent for a set of solutions is illustrated in Figure 3.28

---

**Figure 3.27 Maximum Spread for A Set of Solutions**

**Figure 3.28 Ideal Value for Maximum Spread Metric**
3.4.4 Hypervolume (HV)

The *hypervolume* metric (Zitzler & Thiele, 1999) as a metric which evaluates both diversity and convergence calculates the volume covered by set of obtained solutions; \( Q \); in the objective space for minimization problems. In the example of Figure 3.29 the *Hypervolume* is the enclosed area within the dashed line. For each point \( i \in Q \) the hypercube \( v_i \) is constructed between the reference point \( w \) and solution \( i \) as the diagonal corner of the hypercube. The reference point can be identified by combining the worst values in each objective as a vector. Thereafter, the *hypervolume* is calculated as a union of all constructed hypercubes as follows:

(Alba et al., 2009; Chiong, 2009; Coello et al., 2007; Deb, 2001; Janssens & Pangilinan, 2010; Talbi, 2009; Tan et al., 2005; Yu & Gen, 2010)

\[
HV = \text{volume} \left( \bigcup_{i=1}^{\left| Q \right|} v_i \right)
\]

(3.14)

![Figure 3.29 Hypervolume for a Set of Non-Dominated Solutions](image)

For *hypervolume* metric the maximum value is the area which is shown in grey in Figure 3.30. However, this value may not be the outcome of a practical set of solutions.
3.5 Summary

Optimization is a procedure of finding and comparing different solutions from a set of possible values until no better solution is found. In many real-world optimization problems multiple objectives must be optimized at the same time. Evolutionary multi-objective algorithms are considered as good candidates for solving these problems. In this chapter some principles and fundamentals for evolutionary multi-objective optimization was presented. The evolutionary algorithms were divided into two different classes: the elitist algorithms and the non-elitist algorithms. In the elitist algorithm, a percentage of individual with highest quality are preserved while in the non-elitist such a capability is not foreseen. Of non-elitist algorithms, WBGA, VEGA, NPGA, MOGA and NSGA were discussed. Among the elitist algorithms, SPEA, SPEA-II and NSGA-II were described. Finally four different performance metrics Coverage, Hypervolume, Spacing and Maximum Spread for evaluating evolutionary multi-objective algorithms were presented.
Chapter 4. Methodology

4.1 Introduction

The current chapter is an introduction to the way that the mentioned algorithms have been applied to the view selection problem. The general structure of the current work can be classified into different domains as the following:

- The Problem domain where the characteristics of the problem at hand is defined.
- The Methodology consisting of the algorithm which acts on the problem.

Figure 4.1 illustrates this classification. The left panel belongs to the method domain which includes the different evolutionary multi-objective optimization algorithms such as NSGA, and SPEA while the right panel is the problem domain consisting of the problem to be solved by these algorithms.

The entire design and implementation of the solution system is based on this abstraction. Breaking the system into two manageable domains reduces the complexity and makes it easy to understand. In addition, the individual methods and problem
variants can be easily substituted by alternative ones and thus enhance the extendibility of the system.

The rest of this chapter is organized as follows:

Since both the method and problem domain are designed according to the object oriented concept the first section is devoted to the definition of objects and relationship between them. Each evolutionary algorithm requires some configuration to be well suited to a particular problem. The parameter setting for the applied algorithms will be given in the next section. Then metrics which is used for the evaluation of the algorithms is stated. The problem representation schema is explained next. The initialization, stopping criteria, constraint handling technique and objective normalization is discussed in subsequent sections. The view size estimation used is given in following section. Thereafter, the problem instances which are used as the inputs of the algorithms will be introduced. Finally, the last section presents the hardware and software platform in which the algorithms are implemented.

### 4.2 Object Oriented Architecture

This section explains the designed objects, their properties and methods as well as the relationship between them. The entire objects are classified in two domains: the objects which are defined within the problem domain and the objects that belong to the method domain. The *UML* class diagram for the architecture is shown in Figure 4.2.
4.2.1 Objects in Problem Domain

**View**

As a basic object in the problem domain, the view object encapsulates the characteristics of a view such as size, view update frequency and query frequency. The
property *hierarchy levels* in view class (see Table 4.1) is a list of hierarchy levels in which the view is constructed from them. As mentioned earlier in Section 2.6, in the presence of dimension hierarchies each view is built by choosing one level per dimension hierarchy. For example, Figure 4.3 illustrates 3 hierarchies for three different dimension tables named *Supplier, Customer* and *Part*. Figure 4.4 is a view dependency lattice which is constructed based on the mentioned hierarchies in Figure 4.3. Each view in the lattice of Figure 4.4 is built by choosing only one level from each dimension hierarchy. For instance, the view *SuppliedID-CustomerID-Size* is made up from level *SuppliedID, CustomerID* and *Size* from the dimension hierarchy *Supplier, Customer* and *Part* respectively. The corresponding *group by* query is as follows:

```sql
Select Supplier.SupplierID, Customer.CustomerID, Part.Size, SUM(Sales.Price) 
From Sales 
Group By Supplier.SupplierID, Customer.CustomerID, Part.Size 
```

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Unique number given to the view</td>
</tr>
<tr>
<td>Size</td>
<td>The number of records in the view</td>
</tr>
<tr>
<td>Update Frequency</td>
<td>Frequency by which the view is updated</td>
</tr>
<tr>
<td>Query Frequency</td>
<td>Frequency by which the view is queried</td>
</tr>
<tr>
<td>Hierarchy Levels</td>
<td>List of hierarchy levels in different dimensions which from this view</td>
</tr>
<tr>
<td></td>
<td>constructed from.</td>
</tr>
<tr>
<td>Maximum Size</td>
<td>Maximum possible size of the view</td>
</tr>
<tr>
<td>IsTopView</td>
<td>The view is fact table or not</td>
</tr>
</tbody>
</table>
Figure 4.3 Hierarchy Defined Within Each Dimension Table

Figure 4.4 View Dependency Lattice Calculated Based On Hierarchies in Figure 4.2

- **Lattice**

The lattice object is a data structure which consists of all possible views as well as the relationships between them. Table 4.2 shows the properties and methods for the lattice object. Also, Figure 4.5 visualizes a sample instance of a lattice object as well as the corresponding properties and methods.
### Table 4.2 Lattice Class

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections</td>
<td>List of edges between views in the lattice; an adjacency matrix</td>
</tr>
<tr>
<td>Items</td>
<td>List of views in the lattice</td>
</tr>
<tr>
<td>Count</td>
<td>Number of views in the lattice</td>
</tr>
<tr>
<td>TopNode</td>
<td>The Top node(or fact table) in the lattice</td>
</tr>
<tr>
<td>BottomNode</td>
<td>The bottom node in the lattice</td>
</tr>
<tr>
<td>Edge(i,j)</td>
<td>Whether there is an edge between view i and j or not</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentsOf(V)</td>
<td>List of views which are parent of view V</td>
</tr>
<tr>
<td>AncestorsOf(V)</td>
<td>List of views which are Ancestor of view V</td>
</tr>
<tr>
<td>Childs(V)</td>
<td>List of views which are child of view V</td>
</tr>
<tr>
<td>Draw</td>
<td>Draws the lattice</td>
</tr>
<tr>
<td>Clear</td>
<td>Deletes all views and edges in the lattice</td>
</tr>
</tbody>
</table>

Figures 4.5 The Lattice Object

- **VSP (View Selection Problem)**

The VSP is the main object in the problem domain representing the view selection problem instance. Table 4.3 shows the list of properties and methods for the VSP class. The $objective_1$ and $objective_2$ property holds the address of methods which act as objectives to the problem. These properties can be flexibly set to the $UpdateTime$, $QueryTime$, $Space$ or any other extendable methods. Note that such a way provides...
<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective 1</td>
<td>Refers to the first objective of the problem</td>
</tr>
<tr>
<td>Objective 2</td>
<td>Refers to the second objective of the problem</td>
</tr>
<tr>
<td>Constraint</td>
<td>Refers to the constraint of the problem</td>
</tr>
<tr>
<td>TheLattice</td>
<td>The lattice which is associated with the problem</td>
</tr>
<tr>
<td>CubeSize</td>
<td>The disk space amount needed to store all views</td>
</tr>
<tr>
<td>Qmin</td>
<td>Minimum query response time for answering all possible queries</td>
</tr>
<tr>
<td>Qmax</td>
<td>Maximum query response time for answering all possible queries</td>
</tr>
<tr>
<td>Umin</td>
<td>Minimum update time for updating all materialized views</td>
</tr>
<tr>
<td>Umax</td>
<td>Maximum update time for updating all materialized views</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateTime (M)</td>
<td>The time needed for updating set M of materialized views</td>
</tr>
<tr>
<td>QueryTime (M)</td>
<td>The time needed for answering all queries in presence of set M of views</td>
</tr>
<tr>
<td>Space (M)</td>
<td>The disk space required for storing set M of Materialized views</td>
</tr>
<tr>
<td>DiskSpaceViolation (M)</td>
<td>Amount of disk space violation caused by materializing set M of views</td>
</tr>
<tr>
<td>MostBeneficialView (M)</td>
<td>The view if added to current set of views; M; cause maximum reduction in query response time</td>
</tr>
<tr>
<td>RepairFunction (M)</td>
<td>The function to repair infeasible solution</td>
</tr>
<tr>
<td>LeastCostMaterializedAncestor(V)</td>
<td>The smallest materialized ancestor of V</td>
</tr>
<tr>
<td>SearchSpaceSize</td>
<td>The size of search space</td>
</tr>
</tbody>
</table>

freedom to change the objectives of the problem based on the view selection problem variation at any time in future. For example, when the view selection problem is only a single objective case the first objective may refer to the UpdateTime or QueryTime and the second objectives may be left as null. Moreover, when a combination of query response time and view update time as a single objective is taken into consideration Objective₁ refers to the extended method which adds UpdateTime and QueryTime (QueryTime+UpdateTime). Similarly, the constraint refers to a method which acts as a constraint to the problem. In this research, objective₁ refers to QueryTime, objective₂ refers to UpdateTime and constraint refers to Space. An instance of the lattice object as the property indicates the lattice associated with each view selection problem. The parameter $M$ in the VSP methods is defined as the VSPPhenotype class which will be discussed in the next section.
• VSP Phenotype

In biology the phenotype is the observable features of an organism such as color, shape and size which directly originates from the genotype (Gorunescu, 2011). In evolutionary algorithms the phenotypes are possible solutions to a given problem. Likewise, the VSP Phenotype object is defined as the potential solution to the view selection problem. Table 4.4 shows the different properties and methods for the VSP Phenotype class. Each view selection problem is associated with two values which are total query response time and total update time in this research respectively. These two values are represented by $F_1$ and $F_2$ properties.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>The value for the first objective function</td>
</tr>
<tr>
<td>$F_2$</td>
<td>The value for the second objective function</td>
</tr>
<tr>
<td>$A(i)$</td>
<td>whether the $i^{th}$ view is selected</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Number of all possible views</td>
</tr>
<tr>
<td>Clone</td>
<td>Creates a copy of the current phenotype</td>
</tr>
</tbody>
</table>

4.2.2 Objects in Method Domain

The method domain consists of objects that are pertinent to techniques used to solve the view selection problem. However, these objects are classified into two types, namely, core objects and shell objects. Figure 4.6 shows a scheme for such classification. The objects which are defined within the core section are considered as essential objects. Examples of such basic objects are the individual and the population. The core objects play a fundamental role in the working of each evolutionary algorithm. Every evolutionary algorithm may take advantage of these fundamental objects in its procedure. These objects do not direct any evolutionary process to themselves but instead, they provide ready structure for higher level objects. The shell objects consist of fully independent evolutionary routines such as VEGA, WBGA, NSGA or so on. The
advantage of such classification is the re-usability of the carefully designed core objects.

![Diagram of Core and Shell Objects]

**Figure 4.6 Core And Shell Objects**

### 4.2.2.1 Core Objects

- **GA Object**

All classes within the method domain share a common behavior. Each of these classes implements an evolutionary algorithm which follows a common logic. The *GA* class as an abstract base class provides a way for representing all such shared features among the shell classes in a single entity. The list of properties and methods for the *GA* class is shown in Table 4.5. In object oriented principle in contrast to normal classes (called concrete classes), abstract classes cannot be instantiated. The abstract class can merely be inherited by deriving the classes (Deitel, Deitel, & Nieto, 2001). The purpose of the abstract class is to provide an elegant logical organization for closely related objects. Examples of the shared properties in classes defined in the methods domain are *population size, crossover rate* and *mutation rate* which are customary parameters for all evolutionary algorithms. In addition to these general settings, most of the multi-objective optimization algorithms require common calculations. An example of such calculation is the measurement of the distance between two different individuals either in the decision variable space or objective space. Dominance check is another frequent calculation which determines whether one individual is better than another individual.
with respect to the multiple objectives. All of these tasks are placed as inheritable ready methods inside the GA class and they do not need to be re-defined in the derived classes. The class diagram in Figure 4.7 shows the inheritance between the GA class and the shell objects.

### Table 4.5 GA Class

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopulationSize</td>
<td>Number of individuals in population</td>
</tr>
<tr>
<td>MaximumGeneration</td>
<td>Maximum number of generations to be evolved</td>
</tr>
<tr>
<td>CrossoverRate</td>
<td>Probability by which crossover is applied</td>
</tr>
<tr>
<td>MutationRate</td>
<td>Probability by which mutation is applied</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dominate(a,b)</td>
<td>Whether individual a dominates individual b or not</td>
</tr>
<tr>
<td>ConstrainedDominate(a,b)</td>
<td>Whether individual a constrained dominates individual b or not</td>
</tr>
<tr>
<td>VariableDistance(a,b)</td>
<td>The distance between individual a and b in decision variable space</td>
</tr>
<tr>
<td>ObjectiveDistance(a,b)</td>
<td>The distance between individual a and b in objectives space</td>
</tr>
</tbody>
</table>

![Figure 4.7 Inheritance In Method Domain](image)

- **Individual**

The Individual class contains all characteristics, data structures and procedures which are required in a typical chromosome. Table 4.6 shows the list of properties, methods and operators for the individual class. Each chromosome consists of a series of smaller cells called gene. The GList property is an internal structure of individual class which
stores a list of such genes. Each item of $GList$ may take the value of either 0 or 1. Furthermore, the $Evaluate()$ function calculates the value of the objectives and constraints and assign these values to the corresponding properties in the individual class. The properties $Objective1Value$ and $Objective2Value$ and $ConstraintValue$ store the calculated values by the $evaluate$ method. In the case of this research this properties take values as shown in Table 4.7. In some evolutionary multi-objective algorithm such as $NSGA$ there is a need for a dummy fitness function. For these cases, a property named $Fitness$ is meant to store the dummy fitness value. For the individual class, two operators have been defined: equality and inequality. The equality operators return $true$ when the two individuals are exactly the same otherwise it returns $false$.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GList$</td>
<td>List of genes</td>
</tr>
<tr>
<td>$Objective1Value$</td>
<td>The value returned by first objective function</td>
</tr>
<tr>
<td>$Objective2Value$</td>
<td>The value returned by second objective function</td>
</tr>
<tr>
<td>$ConstraintValue$</td>
<td>The value returned by constraint function</td>
</tr>
<tr>
<td>$Fitness$</td>
<td>Dummy Fitness</td>
</tr>
<tr>
<td>$Rank$</td>
<td>Pareto rank assigned by the non-dominated sorting algorithm</td>
</tr>
<tr>
<td>$Crowding Distance$</td>
<td>The crowding distance of the individuals in a population</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Number of genes in chromosome</td>
</tr>
<tr>
<td>IsFeasible()</td>
<td>Whether the chromosome violates the constraint</td>
</tr>
<tr>
<td>$Evaluate(Objective1,Objective2,Constraint)$</td>
<td>Calculate Objectives and constraint functions and assign the values to the relevant properties</td>
</tr>
<tr>
<td>Clone()</td>
<td>Create the a full copy of the chromosome</td>
</tr>
<tr>
<td>Exchange(index1, index2)</td>
<td>Exchanges gene values in positions index1 and index2</td>
</tr>
<tr>
<td>Decode()</td>
<td>Maps a genotype to the corresponding phenotype($VSP_Phenotype$)</td>
</tr>
<tr>
<td>Flip(i)</td>
<td>Alters a gene value at position i from 0 to 1 or vice versa</td>
</tr>
<tr>
<td>Random()</td>
<td>Generate random value for each gene</td>
</tr>
</tbody>
</table>

Table 4.6 Individual Class
<table>
<thead>
<tr>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Chromosome a=chromosome b</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Chromosome a&lt;&gt;chromosome b</td>
</tr>
</tbody>
</table>

Two individuals are identical when they include equal values in corresponding positions. The inequality operator works in the inverse manner.

Each chromosome or genotype in the evolutionary algorithm is a code of a real-world problem encompassing the process of evolution. The original possible solution in the real-world is called the phenotype. To bridge the problem between the solution in the real world and the genetic world a two way link is required. This link is carried out through the mapping from the phenotype to the genotype and is a fundamental step of the evolutionary algorithm called representation. The inverse is the map of the genotype to the phenotype. Each chromosome must be designed to be invertible (Eiben & Smith, 2008). In the chromosome class a special method simply called Decode is responsible for mapping the chromosome in evolutionary algorithms space to a phenotypic solution (described earlier as VSP_Phenotype class) in real-world space.

- **Population**

The population class as a container stores a number of individuals in a single object. Table 4.8 shows the list of properties, methods and operators which are defined for the population class. The list which includes individuals is represented by IList property.
The function *Evaluate* is responsible for the evaluation of all individuals in the population. The non-dominated set of individuals in the population are calculated using the *Non-dominated* function.

One of the most important methods which play a key role in the working of every evolutionary algorithm is defined in the population class as *Random Generate* method. This method will be explained later in Section 4.6.

<table>
<thead>
<tr>
<th>Table 4.8 Population Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Properties</strong></td>
</tr>
<tr>
<td>ILList</td>
</tr>
<tr>
<td>Count</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>Rank</td>
</tr>
<tr>
<td>Member(i)</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
</tr>
<tr>
<td>Evaluate(Objective1, Objective2, Constraint)</td>
</tr>
<tr>
<td>Non-dominated()</td>
</tr>
<tr>
<td>Dominate(i,j)</td>
</tr>
<tr>
<td>Classify()</td>
</tr>
<tr>
<td>PartitionFeasibility()</td>
</tr>
<tr>
<td>Clone()</td>
</tr>
<tr>
<td>Add(Individual)</td>
</tr>
<tr>
<td>Sum(IndividualField)</td>
</tr>
<tr>
<td>Sort(IndividualField)</td>
</tr>
<tr>
<td>RemoveAt(i)</td>
</tr>
<tr>
<td>Remove(I)</td>
</tr>
<tr>
<td>Clear()</td>
</tr>
<tr>
<td>Contains(Individual)</td>
</tr>
<tr>
<td>RandomGenerate(PopulationSize, ChromosomeSize)</td>
</tr>
<tr>
<td>AssignCrowdingDistance()</td>
</tr>
<tr>
<td>FitnessSharing()</td>
</tr>
<tr>
<td>NicheCount(individual, alpha, SigmaShare)</td>
</tr>
<tr>
<td>Min(IndividualField)</td>
</tr>
<tr>
<td>Procedure</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>Max(IndividualField)</td>
</tr>
<tr>
<td>Find()</td>
</tr>
<tr>
<td>Sum(IndividualField)</td>
</tr>
<tr>
<td>Sort(IndividualField)</td>
</tr>
<tr>
<td>Contains(Individual)</td>
</tr>
<tr>
<td>FindMin(IndividualField)</td>
</tr>
<tr>
<td>FindMax(IndividualField)</td>
</tr>
<tr>
<td>Top()</td>
</tr>
<tr>
<td>Bottom()</td>
</tr>
<tr>
<td>SaveToFile(address)</td>
</tr>
<tr>
<td>DoClustering(Size)</td>
</tr>
<tr>
<td>Representative()</td>
</tr>
<tr>
<td>SuggestSigmaShare()</td>
</tr>
</tbody>
</table>

**Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Subtracts one population from another population</td>
</tr>
<tr>
<td>+</td>
<td>Joins two populations into a single population</td>
</tr>
<tr>
<td>=</td>
<td>Tests whether population $a$ is equal to population $b$</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Tests whether population $a$ is not equal to population $b$</td>
</tr>
</tbody>
</table>

**Population Set**

Some of the procedures like non-dominated sorting or clustering return a set of populations as output rather than a single population. In order to maintain the history of evolution one may need to maintain a number of populations in one place.

The *PopulationSet* class serves as a container for storing a series of relevant populations. However, any population set can be consolidated to an accumulated population by calling the *Consolidate* method. Table 4.9 shows the list of properties and methods for the *PopulationSet* class.
Table 4.9 PopulationSet Class

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Number of population in the population set</td>
</tr>
<tr>
<td>PList</td>
<td>List of including populations</td>
</tr>
<tr>
<td>Member(i)</td>
<td>Returns $i^{th}$ Population</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add(Population)</td>
<td>Adds a population to the population set</td>
</tr>
<tr>
<td>Remove(Population)</td>
<td>Removes a population from population set</td>
</tr>
<tr>
<td>Consolidate</td>
<td>Consolidates all population in population set and returns a single accumulated population</td>
</tr>
<tr>
<td>Merge(Population 1, Population 2)</td>
<td>Merges two member populations: Population 1 and Population 2 and thus the Count is deducted by one</td>
</tr>
<tr>
<td>ClusterDistance(Cluster1, Cluster 2)</td>
<td>Measures the distance between two populations (as Cluster1 and Cluster 2) in the population set</td>
</tr>
</tbody>
</table>

- **Crossover**

The crossover class comprises of all different techniques which are devised for the recombination of two different parent individuals. Table 4.10 shows the properties and methods for the crossover class. The class comes with shared members. In contrast to the normal classes where each instance have their own copy of members, in classes with shared members all instances share a single copy of a specific property or method (Deitel et al., 2001). Three different types of crossover are implemented; these are namely, the *Uniform*, *SinglePoint* and *TwoPoint* crossovers. The *Rate* property represents the probability by which the crossover operator is applied and is set to 0.9 by default for this research.

Table 4.10 Crossover Class

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate</td>
<td>The probability by which crossover is applied</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform (individual x, individual y)</td>
<td>Performs uniform crossover on two parent individuals: individual x and individual y</td>
</tr>
<tr>
<td>SinglePoint(individual x, individual y)</td>
<td>Performs Single crossover on two parent individuals: individual x and individual y</td>
</tr>
<tr>
<td>Two-Point(individual x, individual y)</td>
<td>Performs Two-point crossover on two parent individuals: individual x and individual y</td>
</tr>
</tbody>
</table>
• **Mutation**

The mutation operator is represented by the mutation class. Similar to the crossover class, the mutation class also includes shared property and methods. Table 4.11 shows the properties and methods for the mutation class. The *Rate* property refers to the probability by which the mutation is applied. By default the *Rate* is set to 0.01. Two types of implemented mutation are the *uniform* and *random* methods.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate</td>
<td>The probability by which mutation is applied</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform(individual)</td>
<td>Performs uniform mutation on the parent individual</td>
</tr>
<tr>
<td>Random(individual)</td>
<td>Performs random mutation on the parent individual</td>
</tr>
</tbody>
</table>

• **Selection**

The selection class includes all different ways for selecting one set of parent among a given population. Table 4.12 shows the methods in the selection class for the four different selection techniques namely, the *Roulette wheel(), Random(), StochasticRemainderSelection() and Tournament()*. All the methods take an instance of the population class as well as a field of individual as input and return a single individual as output. However, in the case of random selection no field is specified. The selection is done according to the field specified. For selecting two mates the selection function is required to be called twice. In the case of the tournament selection, the *Bios={greater, less}* parameter specifies whether the selection is done based on smaller values or bigger values.
### Table 4.12 Selection Class

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roulette wheel(Population, IndividualField)</td>
<td>Implements the Roulette wheel selection technique on a population based on the individualfield</td>
</tr>
<tr>
<td>Random(Population)</td>
<td>Implements the random selection technique on a population</td>
</tr>
<tr>
<td>StochasticRemainderSelection(Population, , IndividualField)</td>
<td>Implements the stochastic remainder selection technique on a population based on the individualfield</td>
</tr>
<tr>
<td>Tournament(Population, IndividualField, Bios)</td>
<td>Implements the tournament selection technique on a population based on the individual field and Bios</td>
</tr>
</tbody>
</table>

#### 4.2.2.2 Shell Objects

Shell classes implement evolutionary algorithms based on the fundamental object defined in the core area. These algorithms present a complete evolutionary algorithm and can be executed independently. The shell classes follow a shared interface as shown in Figure 4.8. The interface comes with only one method named `Run` (see Table 4.13) which serves as a starter of the algorithm.

### Table 4.13 Evolutionary Multi-Objective Algorithm Interface

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run(Objective1, Objective2, Constraint)</td>
<td>Takes the three parameters: the first objective, second objective and constraint of the problem and starts the evolutionary algorithm</td>
</tr>
</tbody>
</table>

The parameters for `Run` are `objective_1` as first objective to the problem, `objective_2` as second objective to the problem and `constraint` as the problem constraint, as presented in Table 4.13. These parameters hold the address of an already defined function.
4.2.3 Performance Evaluation

In order to evaluate the performance of the applied algorithms, a special class is designed. All the used metrics are defined as methods in this class. These methods receive one or two approximations and calculate the metrics. Each approximation is a set of obtained non-dominated solutions by a specific algorithm in the final population.

As an example, the metric *Coverage* takes two approximations called *approximation*₁ and *approximation*₂ and returns the corresponding value for the two set coverage. The list of methods in the performance evaluation class is shown in Table 4.14.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Spread (Approximation)</td>
<td>Calculates the <em>maximum spread</em> metric for an approximation</td>
</tr>
<tr>
<td><em>HyperVolume</em> (Approximation, ReferencePoint)</td>
<td>Calculates the <em>hypervolume</em> metric</td>
</tr>
<tr>
<td>Coverage(Approximation1,Approximation2)</td>
<td>Calculates the <em>coverage metric</em> for an two approximations: Approximation1 and Approximation2</td>
</tr>
<tr>
<td>Spacing(Approximation)</td>
<td>Calculates the <em>spacing</em> metric for an approximation</td>
</tr>
</tbody>
</table>

4.3 Parameter Setting

Each evolutionary algorithm consists of a set of parameters which need to be well-determined before the execution of the algorithm, as the performance of the algorithm is affected by such parameters. For example, if the population size is too small, the
An evolutionary algorithm may be trapped in the local optima and may fail to discover the global optima. On the other hand too large a population size slows down the algorithm and wastes the computational resource. Choosing a proper value for such parameter is not an easy task and in practice is usually done by trial-and-error (Haupt & Haupt, 1997; Michalewicz, 1996). Although some studies (Back, 1993; Davis, 1989; Grefenstette, 1986; Jong, 1975; Srinivas & Patnaik, 1994) were performed to find the optimal parameter settings for particular test cases, in general, a theoretical prescription is not available (Bagchi, 1999) and there is no conclusion on what setting is best (Mitchell, 1998). In fact, the control parameters are problem-specific (Bagchi, 1999). In this research, the crossover, population size and generation number have been experimentally tuned. The mutation rate was set to \( \frac{1}{\text{Chromosome Length}} \) as recommended by (Back, 1993). The distance between two individuals is calculated as the Euclidian distance in objective space. For example, the Euclidean distance between two points shown in Figure 4.9 is calculated as the following:

\[
\begin{align*}
    d &= \sqrt{((f_1(a) - f_1(b))^2 + (f_2(a) - f_2(b))^2)} \\
\end{align*}
\]

Table 4.15 shows the list of chosen parameters for the research.

![Figure 4.9 Calculation of distance in 2D objective space](image)
Table 4.15 GA Parameter Setting

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Population Size</td>
<td>100</td>
</tr>
<tr>
<td>Secondary Population Size</td>
<td>20</td>
</tr>
<tr>
<td>( t_{dom} )</td>
<td>10%</td>
</tr>
<tr>
<td>( \sigma_{share} )</td>
<td>According to the procedure proposed by (Fonseca &amp; Fleming, 1993)</td>
</tr>
<tr>
<td>Maximum Generation Number</td>
<td>100</td>
</tr>
<tr>
<td>Crossover Type</td>
<td>Single-point</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>0.8</td>
</tr>
<tr>
<td>Mutation Type</td>
<td>Bit-wise</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>1/number of views</td>
</tr>
<tr>
<td>Selection Method</td>
<td>Binary Tournament Selection unless specified by algorithm</td>
</tr>
<tr>
<td>Number of Runs</td>
<td>30</td>
</tr>
</tbody>
</table>

Concerning the hypervolume metric the reference point defined by the value 100 in each objective as shown in Figure 4.10.

In algorithms NSGA, MOGA and NPGA where a sharing strategy is required the niche radius, \( \sigma_{share} \), was calculated using the Fonseca and Fleming update rule (Fonseca & Fleming, 1993).

Since combining the fitness sharing and tournament selection may cause chaotic behavior as reported by (Oei, Goldberg, & Chang, 1991) wherever a combination of
them is required a slightly modified version of sharing called *continuously updating sharing* (Oei et al., 1991) is used. Although in the original paper of *NPGA* (Horn et al., 1994) no procedure was proposed for setting the $t_{dom}$ but the authors recommend 10% of the main population. For *NPGA*, $t_{dom} = 10$ was selected. For *SPEA* and *SPEA-II* the external population size of 20 was selected while 100 was selected as size of the primary population.

The non-dominated solutions from the last generation of each run were identified and they are considered as outcome of the optimization run.

### 4.4 Performance Metrics Used

It is to be noted that as mentioned in Section 3.4, some performance metrics for the assessment of evolutionary multi-objective optimization algorithms require knowledge of the true pareto optimal set. These metric are useful when the set of optimal solution for a specific problem is available. Example of such metrics is the *Generational Distance (GD)* or *Error Ratio (ER)* (Deb, 2001). Since in the case of the view selection problem the pareto optimal set is unknown therefore the metrics could not be applied.

As stated in Section 3.4 the metrics for examining the performance of the evolutionary multi-objective algorithms are convergent based or diversity based. However some hybrid metrics measure both of these aspects. Four complementary metrics used for performance assessment of the algorithms. *Two Set Coverage* as a convergence based metrics and *Maximum Spread* and *Spacing* as two diversity based metrics were used. In addition the *hypervolume* metric as a hybrid metric which measures both convergence and spread of solutions are used. All the metric used except the *two set coverage* are considered as unary metric since they take the result obtained by one algorithm and return one real value as output. The *Two Set Coverage* as a binary metric takes the
obtained result from two different algorithms and returns an output value which implies the comparison of two algorithms.

For each metric the ideal values is listed in Table 4.16. The ideal value represents the best imaginable value for a particular metric. The Maximum Spread and Hypervolume is calculated by substituting the value of 100 to the variable $F_{1\text{max}}$ and $F_{2\text{max}}$ in Figure 3.28 and Figure 3.30

<table>
<thead>
<tr>
<th>Metric</th>
<th>Ideal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HyperVolume</td>
<td>10000</td>
</tr>
<tr>
<td>Spacing</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Spread</td>
<td>$\sqrt{100^2 + 100^2}$</td>
</tr>
<tr>
<td>Two Sets Coverage</td>
<td>1</td>
</tr>
</tbody>
</table>

### 4.5 Problem Representation

Representation is considered as a fundamental step and key element in designing an evolutionary algorithm. As mentioned in Chapter 3, representation refers to encoding a real-world problem characteristic to an appropriate computer data structure. An array of binary values is the most common way of encoding (Sivanandam & Deepa, 2009). In the case of the view selection problem, a potential solution to the problem is encoded to an array of binary values. The size of the array is identical to the number of possible views. A one (1) in the $i^{th}$ position of the array means that the $i^{th}$ view is selected for materialization while a zero (0) in the $i^{th}$ position indicates the $i^{th}$ view is not selected. For example, Figure 4.11 shows a dependency lattice for a view selection problem with 8 possible views. In addition, a corresponding array with 8 cells is shown. The views which are labeled with numbers 2, 3, and 6 (shown in grey) are the views that have been selected for materialization and the rest of the views that have not been selected. As can be seen, the array cells with number 2, 3 and 6 are set to one while the other cells take
the value of zero. Based on this form of encoding, it is clear that with $|V|$ possible views the total number of points in the search space (as search space size) is the number of all the combination of arrays with different values and is equal to $2^{|V|}$. For the example shown in Figure 4.11, the search space size is $2^8$ (see Figure 4.12).

All the eight (8) algorithms were implemented using the same binary encoding scheme presented here with 64 bits for $VSP_1$ and 48 bits for $VSP_2$ to represent the decision variable.

![Diagram](image)

Figure 4.11 View Selection Problem Encoding

![Diagram](image)

Figure 4.12 Calculating the Size of Search Space

### 4.6 Initialization

An evolutionary algorithm starts by an initial population. The common way for creating an initial population is to generate a population by assigning random values. An ideal random population is supposed to be well distributed in the entire search space (Engelbrecht, 2007). For the view selection problem, a random population of
individuals is generated by the \textit{RandomGenerate} method in the population class. The parameter \textit{Populationsize} determines the size of the population to be generated while the \textit{ChromosomeSize} refers to the number of genes which an individual include. Before calling \textit{RandomGenerate} the population is supposed to be empty, otherwise all existing individuals are simply deleted first.

In all studies, in order to avoid the impact of random effect, 30 independent runs with different random seeds (to create different random initial population) were made per algorithm/problem instance which leads to 30 sets of solutions in the final generation.

\textbf{4.7 Stopping Criteria}

An evolutionary algorithm stops when a specific stopping criteria holds. The different possible criteria were discussed in Termination Condition in Section 3.3.1.6. However, the algorithms implemented in this research are terminated when they reach to the maximum number of generation.

\textbf{4.8 Constraint Handling}

As mentioned in Section 1.3 the variation of the view selection problem pertinent to this research involves the disk space constraint. That is a potential solution to the problem that must be fulfilled is the total disk space requirement for storing all views. Otherwise, the solution is regarded as an infeasible solution. The constrained dominance technique (Deb, 2001) is a parameterless constraint handling approach which uses the original dominance concept and the binary tournament. The advantage for such technique is that all methods designed based on the normal dominance definition can still work with only minor modifications. In addition, the approach results in a better pareto spread and convergence as stated in (Deb, 2001). For \textit{VEGA} and \textit{WBGA} which are not based on dominance concept a modified binary tournament selection operator is used as
described in (Deb, 2001). The modified binary tournament selection operator picks two random individual; $x$ and $y$ from the population and of them one individual is chosen based on two criteria: feasibility and objective value. Taking constraints into consideration three different situations may happen:

a) Both individuals are feasible

b) One individual is feasible and the other is infeasible

c) Both individuals are infeasible

Thereafter one individual is chosen following a simple rule:

Case a) An individual with better objective value is chosen

Case b) The feasible individual is chosen

Case c) The individual with less constraint violation is chosen.

### 4.9 Objective Normalization

The objective of a multi-objective optimization problem may take values of different order of magnitude. In order to make each objective to be in the same order of magnitude and equally important the objectives need to be scaled properly. The procedure is called objective normalization and requires the knowledge of maximum and minimum values for each objective (Deb, 2001). In the case of view selection problem, both objectives, i.e. total query response time and total view update time, were normalized to give value between 0 and 100 and calculated using following equations:

$$Q_n(M) = \frac{Q(M) - Q_{\text{min}}}{Q_{\text{max}} - Q_{\text{min}}} \times 100$$  \hspace{1cm} 4.2

$$U_n(M) = \frac{U(M) - U_{\text{min}}}{U_{\text{max}} - U_{\text{min}}} \times 100$$  \hspace{1cm} 4.3
The $Q_n(M)$ and $U_n(M)$ corresponds to the normalized values for total query response time and total view update time respectively. $Q_{min}$, $Q_{max}$, $U_{min}$ and $U_{max}$ were described in Sections 2.10 and 2.11.

### 4.10 View Size Estimation

As mentioned in Chapter 2, the view selection algorithms require the knowledge of the size of a view without actually computing it since the computation of large number of views is considered as an expensive task and is considered to be impractical. Without the actual computation of a view, determining the exact size of a view may not be possible. In practice, in order to determine the size of a view, view size estimation is used instead. Of various suggested techniques for estimation the Cardenas’ formula (Cardenas, 1975) is utilized due to its simplicity and low computational complexity.

### 4.11 Problem Instances

TPC-H benchmark ("The TPC Benchmark™H," 2011) is a database generator which is recommended by the Transaction Processing Performance Council (TPC) (http://www.tpc.org) and is widely used as a standard in decision support applications. All view selection problem instances are derived from the database. The populated database has 1 GB size and is uniformly distributed. The star schema for this database is shown in Figure 4.15. The schema consists of three dimensions, that is: Supplier, Part and Customer; as well as a central fact table, Sales. The parts are obtained from a supplier and are sold to a customer for a specific price. All aggregations are applied to the price attribute as a measure of interest. In order to define the VSP instances a special tool is designed. The tool takes the meta-data which is driven from the synthetic database and calculates an instance of the VSP class. Figure 4.13 and Figure 4.14 show
the different steps for creating the problem instance and the tool interface for defining the problem instance respectively.
The dimension hierarchy differs from one problem instance to another. Since we are interested to investigate the behavior (convergence, diversity and computational time) of the algorithms with different size of the problem we derived two problem instances with 64 and 48 views. The smaller problem was created by logically ignoring the Region level from Customer hierarchy.

The reason behind selecting only two problem instances is that the performance of the view selection algorithms was expected to be more dependent on the metadata rather than the actual content of data. These metadata are view sizes, query frequency and view update frequency and logical structure of views. Since the calculation of actual view sizes is impractical as stated in Section 2.12 the view sizes are estimated using an analytical method (Cardenas, 1975). The query and view update frequency are also determined by a probability model and the structure of views are derived from the dimension hierarchies. Furthermore, the dimension hierarchies are logically assigned to each dimension table. As a result our main concern was to identify how the algorithm works with different sizes of the search space. Apart from this, a number of outstanding research works in the field of view selection problem such as the works by (Aouiche et al., 2006; Baralis et al., 1997; Baril & Bellahsene, 2003; Harinarayan et al., 1996; Hung et al., 2007; Lin & Kuo, 2004; Phuboon-ob & Auepanwiriyakul, 2007b; Song & Gao, 2010; Wang & Zhang, 2005) also use one or two problem instances in their experiments.

For the query and update frequency, a uniform frequency is assumed which indicates identical probability for query and update. For the disk space constraint, the disk space quota was set to 10% of the total size of all views.
4.11.1 VSP₁

The first problem instance called VSP₁ introduces the largest search space among the problem instances. All the dimension tables have four levels (or nodes) of summarization as shown in Figure 4.16. As mentioned earlier in Chapter 2, the hypothetical attribute All, implies aggregation of all records in a dimension. The total number of possible views are calculated as the product of the number of hierarchy nodes in the different dimension hierarchies (i.e. \(4 \times 4 \times 4\)); and in the case of VSP₁, is equal to 64. The list of all derived views is detailed in Table 4.17. Furthermore, the size of the search space is the power set of all the views which is \(2^{64}\). The dependency lattice with 64 views for VSP₁ is shown in Figure 4.17.

![Figure 4.16 Dimension Hierarchies for VSP₁](image-url)
<table>
<thead>
<tr>
<th>Node</th>
<th>Attributes</th>
<th>Node</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Supplier ID, Customer ID, Part ID,</td>
<td>32</td>
<td>Nation, All, Part ID,</td>
</tr>
<tr>
<td>1</td>
<td>Nation, Customer ID, Part ID,</td>
<td>33</td>
<td>Nation, Region, Type,</td>
</tr>
<tr>
<td>2</td>
<td>Supplier ID, Nation, Part ID,</td>
<td>34</td>
<td>Nation, Region, Size,</td>
</tr>
<tr>
<td>3</td>
<td>Supplier ID, Customer ID, Type,</td>
<td>35</td>
<td>Nation, Nation, All,</td>
</tr>
<tr>
<td>4</td>
<td>Supplier ID, Customer ID, Size,</td>
<td>36</td>
<td>Supplier ID, All, Type,</td>
</tr>
<tr>
<td>5</td>
<td>Region, Customer ID, Part ID,</td>
<td>37</td>
<td>Supplier ID, All, Size,</td>
</tr>
<tr>
<td>6</td>
<td>Nation, Nation, Part ID,</td>
<td>38</td>
<td>Supplier ID, Region, All,</td>
</tr>
<tr>
<td>7</td>
<td>Nation, Customer ID, Type,</td>
<td>39</td>
<td>All, Region, Part ID,</td>
</tr>
<tr>
<td>8</td>
<td>Nation, Customer ID, Size,</td>
<td>40</td>
<td>All, Nation, Type,</td>
</tr>
<tr>
<td>9</td>
<td>Supplier ID, Region, Part ID,</td>
<td>41</td>
<td>All, Nation, Size,</td>
</tr>
<tr>
<td>10</td>
<td>Supplier ID, Nation, Type,</td>
<td>42</td>
<td>All, Customer ID, All,</td>
</tr>
<tr>
<td>11</td>
<td>Supplier ID, Nation, Size,</td>
<td>43</td>
<td>Region, All, Part ID,</td>
</tr>
<tr>
<td>12</td>
<td>Supplier ID, Customer ID, All,</td>
<td>44</td>
<td>Region, Region, Type,</td>
</tr>
<tr>
<td>13</td>
<td>All, Customer ID, Part ID,</td>
<td>45</td>
<td>Region, Region, Size,</td>
</tr>
<tr>
<td>14</td>
<td>Region, Nation, Part ID,</td>
<td>46</td>
<td>Region, Nation, All,</td>
</tr>
<tr>
<td>15</td>
<td>Region, Customer ID, Type,</td>
<td>47</td>
<td>Nation, All, Type,</td>
</tr>
<tr>
<td>16</td>
<td>Region, Customer ID, Size,</td>
<td>48</td>
<td>Nation, All, Size,</td>
</tr>
<tr>
<td>17</td>
<td>Nation, Region, Part ID,</td>
<td>49</td>
<td>Nation, Region, All,</td>
</tr>
<tr>
<td>18</td>
<td>Nation, Nation, Type,</td>
<td>50</td>
<td>Supplier ID, All, All,</td>
</tr>
<tr>
<td>19</td>
<td>Nation, Nation, Size,</td>
<td>51</td>
<td>All, All, Part ID,</td>
</tr>
<tr>
<td>20</td>
<td>Nation, Customer ID, All,</td>
<td>52</td>
<td>All, Region, Type,</td>
</tr>
<tr>
<td>21</td>
<td>Supplier ID, All, Part ID,</td>
<td>53</td>
<td>All, Region, Size,</td>
</tr>
<tr>
<td>22</td>
<td>Supplier ID, Region, Type,</td>
<td>54</td>
<td>All, Nation, All,</td>
</tr>
<tr>
<td>23</td>
<td>Supplier ID, Region, Size,</td>
<td>55</td>
<td>Region, All, Type,</td>
</tr>
<tr>
<td>24</td>
<td>Supplier ID, Nation, All,</td>
<td>56</td>
<td>Region, All, Size,</td>
</tr>
<tr>
<td>25</td>
<td>All, Nation, Part ID,</td>
<td>57</td>
<td>Region, Region, All,</td>
</tr>
<tr>
<td>26</td>
<td>All, Customer ID, Type,</td>
<td>58</td>
<td>Nation, All, All,</td>
</tr>
<tr>
<td>27</td>
<td>All, Customer ID, Size,</td>
<td>59</td>
<td>All, All, Type,</td>
</tr>
<tr>
<td>28</td>
<td>Region, Region, Part ID,</td>
<td>60</td>
<td>All, All, Size,</td>
</tr>
<tr>
<td>29</td>
<td>Region, Nation, Type,</td>
<td>61</td>
<td>All, Region, All,</td>
</tr>
<tr>
<td>30</td>
<td>Region, Nation, Size,</td>
<td>62</td>
<td>Region, All, All,</td>
</tr>
<tr>
<td>31</td>
<td>Region, Customer ID, All,</td>
<td>63</td>
<td>All, All, All,</td>
</tr>
</tbody>
</table>
4.11.2 VSP₂

The second problem instance is called VSP₂. The Supplier dimension table includes four levels of aggregation as Supplier ID, Nation, Region and All. The Customer dimension table consists of three levels as CustomerID, Nation and All. The third dimension table, the Part dimension, consists of four hierarchy nodes as PartID, Type, Size and All. The dimension hierarchies are shown in Figure 4.18. The total number of possible views is 48 and hence the size of the search space is $2^{48}$. The list of derived views is presented in Table 4.18 while Figure 4.19 shows the dependency lattice for VSP₂.
Table 4.18 List Of Views for VSP$_2$

<table>
<thead>
<tr>
<th>Node</th>
<th>Attributes</th>
<th>Node</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Supplier ID, Customer ID, Part ID,</td>
<td>24</td>
<td>All, Nation, Part ID,</td>
</tr>
<tr>
<td>1</td>
<td>Nation, Customer ID, Part ID,</td>
<td>25</td>
<td>All, Customer ID, Type,</td>
</tr>
<tr>
<td>2</td>
<td>Supplier ID, Nation, Part ID,</td>
<td>26</td>
<td>All, Customer ID, Size,</td>
</tr>
<tr>
<td>3</td>
<td>Supplier ID, Customer ID, Type,</td>
<td>27</td>
<td>Region, All, Part ID,</td>
</tr>
<tr>
<td>4</td>
<td>Supplier ID, Customer ID, Size,</td>
<td>28</td>
<td>Region, Nation, Type,</td>
</tr>
<tr>
<td>5</td>
<td>Region, Customer ID, Part ID,</td>
<td>29</td>
<td>Region, Nation, Size,</td>
</tr>
<tr>
<td>6</td>
<td>Nation, Nation, Part ID,</td>
<td>30</td>
<td>Region, Customer ID, All,</td>
</tr>
<tr>
<td>7</td>
<td>Nation, Customer ID, Type,</td>
<td>31</td>
<td>Nation, All, Type,</td>
</tr>
<tr>
<td>8</td>
<td>Nation, Customer ID, Size,</td>
<td>32</td>
<td>Nation, All, Size,</td>
</tr>
<tr>
<td>9</td>
<td>Supplier ID, All, Part ID,</td>
<td>33</td>
<td>Nation, Nation, All,</td>
</tr>
<tr>
<td>10</td>
<td>Supplier ID, Nation, Type,</td>
<td>34</td>
<td>Supplier ID, All, All,</td>
</tr>
<tr>
<td>11</td>
<td>Supplier ID, Nation, Size,</td>
<td>35</td>
<td>All, All, Part ID,</td>
</tr>
<tr>
<td>12</td>
<td>Supplier ID, Customer ID, All,</td>
<td>36</td>
<td>All, Nation, Type,</td>
</tr>
<tr>
<td>13</td>
<td>All, Customer ID, Part ID,</td>
<td>37</td>
<td>All, Nation, Size,</td>
</tr>
<tr>
<td>14</td>
<td>Region, Nation, Part ID,</td>
<td>38</td>
<td>All, Customer ID, All,</td>
</tr>
<tr>
<td>15</td>
<td>Region, Customer ID, Type,</td>
<td>39</td>
<td>Region, All, Type,</td>
</tr>
<tr>
<td>16</td>
<td>Region, Customer ID, Size,</td>
<td>40</td>
<td>Region, All, Type,</td>
</tr>
<tr>
<td>17</td>
<td>Nation, All, Part ID,</td>
<td>41</td>
<td>Region, Nation, All,</td>
</tr>
<tr>
<td>18</td>
<td>Nation, Nation, Type,</td>
<td>42</td>
<td>Nation, All, All,</td>
</tr>
<tr>
<td>19</td>
<td>Nation, Nation, Size,</td>
<td>43</td>
<td>All, All, Type,</td>
</tr>
<tr>
<td>20</td>
<td>Nation, Customer ID, All,</td>
<td>44</td>
<td>All, All, Size,</td>
</tr>
<tr>
<td>21</td>
<td>Supplier ID, All, Type,</td>
<td>45</td>
<td>All, Nation, All,</td>
</tr>
<tr>
<td>22</td>
<td>Supplier ID, All, Size,</td>
<td>46</td>
<td>Region, All, All,</td>
</tr>
<tr>
<td>23</td>
<td>Supplier ID, Nation, All,</td>
<td>47</td>
<td>All, All, All,</td>
</tr>
</tbody>
</table>
4.12 Hardware and Software Specification

All experiments were performed on a computer with Intel Core 2 duo 1.8 GHz processor, 3 GB of memory and 160 GB of hard disk running Microsoft windows 7 Professional. The system was free from other computation or being interrupted by other programs. The implementation of the algorithms was carried out using Microsoft Visual Basic 2008. The Visual Basic programming code is presented in Appendix A.

4.13 Summary

The general structure of the current work is classified into different domains as the following: The problem domain where the characteristics of the problem at hand are defined and the methodology consisting of the algorithm which acts on the problem. In this chapter different objects defined for each domain were discussed. Each
evolutionary algorithm works with a set of parameters; the values chosen for the parameters in this research were stated. The metrics which has been used for evaluating the performance of the evolutionary multi-objective algorithm listed. Thereafter, problem representation, stopping criteria, constraint handling technique and object normalization used was discussed.

Two problem instances called $VSP_1$ and $VSP_2$ were used in this research. The description for each problem instance was given and finally the hardware and software specification for experimental work mentioned.
Chapter 5. Results and Discussion

This chapter presents the results for the comparison of eight well-known evolutionary multi-objective algorithms based on four different measures and computational time. The algorithms included in the experiments were WBGA (Hajela & Lin, 1992), NSGA (Srinivas & Deb, 1994), NSGA-II (Deb, Pratap, Agarwal, & Meyarivan, 2002), SPEA (Zitzler & Thiele, 1999), SPEA-II (Zitzler, Laumanns, et al., 2001), VEGA (Schaffer, 1985), MOGA (Fonseca & Fleming, 1993) and NPGA (Horn et al., 1994) which was described in Chapter 3.

VEGA, NPGA, MOGA and NSGA are considered as the most important and most popular algorithms for MOEA as stated in (Coello, 1999). The eight (8) chosen algorithms come with different perspectives and approaches and are frequently used in different real-world applications as stated in Table 3.1. Some of these algorithms use the dominance concept (NSGA-II, NSGA, SPEA, SPEA-II, MOGA, NPGA) while there are algorithms (VEGA, WBGA) which are not based on the concept of dominance. Some of the algorithms use the elitism (SPEA, SPEA-II, NSGA-II) feature while other algorithms (WBGA, NPGA, MOGA, VEGA, NSGA) do not use elitism feature. Furthermore, different selection mechanisms and different fitness assignment techniques of these algorithms make them a diverse set of algorithms for experimentation.

Table 5.2 to Table 5.23, Figure 5.2 and Figure 5.3 summarize the experimental results for each problem instance with respect to the performance metrics Two Set Coverage, Hypervolume, Spacing, Maximum Spread and computational time. It is to be noted that
the comparison between different algorithms were made based on the mean values of each metric and computational time in 30 runs.

The distribution of values for each metric and problem instance in 30 simulation runs is shown in a set of box plots (also known as Box and Whisker (Chambers, Cleveland, Tukey, & Kleiner, 1983)) in Appendix B which visualize the distribution of data set across a range at glance. Each plot includes a central box with 50% of the data as well as two tails which called whiskers. The plot (see Figure 5.1) consists of five numbers (called five number summary): lower extreme, lower quartile, median, upper quartile and upper extreme which divide the whole data into four parts. Each of four parts contains 25 percent of data. The box extends from lower quartile to upper quartile. The horizontal line inside the middle of the box corresponds to the median of data. The upper and lower edge of the box shows the upper quartile and lower quartile which are 75th and 25th percentile of data respectively. The upper and lower horizontal line represent the maximum and minimum observed value. All other observed values beyond the whiskers are called outliers and marked by *. (Dekking, Kraaikamp, Lopuhaä, & Meester, 2007; Ouellette, 2009; Ross, 1987; Wackerly, Mendenhall, & Scheaffer, 2001; Zhang, 2006)

Multiple comparisons between different algorithms subject to a performance metric are shown in a number of tables. Each cell of the multiple comparisons gives the difference between the mean value of one algorithm (in row) with respect to another algorithm (in column). In addition, difference between algorithms which do not show any statistical significance is denoted by an asterisk (*).
For each metric and problem instance, the algorithms are ranked based on the metric value from the algorithm with the best value to the algorithm with the worst value. Moreover, when there is no statistically significance difference between two or more algorithm they are placed in identical ranks.

### 5.1 Coverage Metric Results

Table 5.2 and Table 5.3 show the mean values for the two set coverage metric subject to VSP\(_1\) and VSP\(_2\). Each cell in Table 5.2 and Table 5.3 represents the two set coverage metric value with respect to the algorithm in the corresponding row and column. For example, the value 0.04 in row 3 and column 2 in Table 5.2 represents \(C(SPEA, \text{NSGA-II})\). As mentioned in Section 3.4, the metric \(C(A,B)\) calculates the percentage of solutions in set \(B\) which are dominated by solutions in set \(A\). \(C(A,B)=1\) indicates that all the solutions in set \(B\) are dominated by solutions in \(A\) while \(C(A,B)=0\) indicates that there is no solution in \(B\) which is dominated by a solution in \(A\). Since the two set
coverage is not a symmetric relation then \(C(A,B) \neq 1 - C(B,A)\). \(C(A,A)\) always takes the value of zero since equal populations do not dominate each other.

Two multiple comparisons of the coverage metric for \(VSP_1\) and \(VSP_2\) are derived from the mean of two set coverage metric (Table 5.2 and Table 5.3) and the values are represented in Table 5.4 and Table 5.5 for \(VSP_1\) and \(VSP_2\) respectively. Each cell in Table 5.4 and Table 5.5 represents the values of \(C(A,B) - C(B,A)\) with algorithm \(A\) in the row and algorithm \(B\) in the column of the table.

Next, the ranking of the algorithm is calculated based on the following:

A particular algorithm is placed in rank \(i\) and called \(A_i\) if for each \(j > i\) (higher ranks):

\[
Cell(A_i,A_j) > 0
\]

where \(Cell(A_i,A_j)\) is a particular cell in Table 5.4 or Table 5.5 with algorithm \(A_i\) in row and algorithm \(A_j\) in column. For example, \(SPEA-II\) is placed in rank 1 in Table 5.6 because the above condition holds as listed in Table 5.1:

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Cell(SPEA-II,NSGA-II)&gt;0)</td>
<td>Yes</td>
</tr>
<tr>
<td>(Cell(SPEA-II,SPEA)&gt;0)</td>
<td>Yes</td>
</tr>
<tr>
<td>(Cell(SPEA-II,NSGA)&gt;0)</td>
<td>Yes</td>
</tr>
<tr>
<td>(Cell(SPEA-II,VEGA)&gt;0)</td>
<td>Yes</td>
</tr>
<tr>
<td>(Cell(SPEA-II,WBGA)&gt;0)</td>
<td>Yes</td>
</tr>
<tr>
<td>(Cell(SPEA-II,NPGA)&gt;0)</td>
<td>Yes</td>
</tr>
<tr>
<td>(Cell(SPEA-II,MOGA)&gt;0)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Table 5.2 Mean Values of Two Set Coverage Metric for VSP1

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>0.77</td>
<td>0.95</td>
<td>0.95</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>0.42</td>
<td>0.97</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SPEA</td>
<td>0.15</td>
<td>0.04</td>
<td>0.72</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NSGA</td>
<td>0</td>
<td>0</td>
<td>0.22</td>
<td>0.97</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>VEGA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.78</td>
<td>0.85</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.17</td>
<td>0.62</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.09</td>
<td>0.47</td>
<td>0.95</td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.24</td>
<td>0.11</td>
<td>0.11</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3 Mean Values of Two Set Coverage Metric for VSP2

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>0.58</td>
<td>0.71</td>
<td>0.96</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>0.56</td>
<td>0.90</td>
<td>0.93</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SPEA</td>
<td>0.07</td>
<td>0.180</td>
<td>0.91</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NSGA</td>
<td>0</td>
<td>0</td>
<td>0.17</td>
<td>0.80</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>VEGA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.715</td>
<td>0.911</td>
<td>0.83</td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.26</td>
<td>0.500</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.10</td>
<td>0.60</td>
<td>0.90</td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.17</td>
<td>0.08</td>
<td>0.19</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4 Multiple Comparison of Coverage Metric for VSP1

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>0.35</td>
<td>0.79</td>
<td>0.95</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>0.94</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEA</td>
<td>0.05</td>
<td>0.97</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSGA</td>
<td>0</td>
<td>0.61</td>
<td>0.77</td>
<td>0.70</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEGA</td>
<td>0</td>
<td>0.15*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td>0.10</td>
<td>0.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td></td>
<td>0.85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.5 Multiple Comparison of Coverage Metric for VSP2

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>0.02*</td>
<td>0.64</td>
<td>0.96</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>0.72</td>
<td>0.93</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SPEA</td>
<td>0.80</td>
<td>0.80</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NSGA</td>
<td>0.45</td>
<td>0.82</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEGA</td>
<td>0.45</td>
<td>0.82</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td>-0.10*</td>
<td>0.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td></td>
<td>0.72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

However two algorithms, A and B are placed in equal rank if there is no statistical significance between \( C(A,B) \) and \( C(B,A) \) (or \( cell(A,B) \) is represented by *). For example
in Table 5.6, WBGA and NPGA are in rank 6 since cell (WBGA, NPGA) represented by * in Table 5.4.

Table 5.6 represents a ranking table based on the values of the multiple comparison of VSP₁ and Table 5.7 shows the ranking table for VSP₂. From Table 5.6 and Table 5.7 it can be observed that all elitist multi-objective algorithms (SPEA-II, NSGA-II, SPEA) perform better than the non-elitist algorithms (NSGA, MOGA, NPGA, VEGA, WBGA). This implies that elitism plays an important role in directing the population towards the pareto optimal set. Among the elitist algorithms, the SPEA-II is slightly better than the NSGA-II. However, the difference is not significant for VSP₂. That indicates two rival algorithms may have almost equal performances in solving VSP₂. This may be due to the smaller size of the VSP₂ problem and therefore both algorithms encounter fewer difficulties to converge to the pareto optimal set. Amongst the non-elitist algorithms NSGA seems to be superior. MOGA is particularly weak in converging to the true pareto optimal set as compared to the other algorithms. In solving both VSP₁ and VSP₂, no significance difference is seen between WBGA and NPGA. In both problem instances VEGA exhibit a fair performance. The set of solutions returned by NSGA-II and SPEA-II mostly cover that of NSGA and SPEA. This is reasonable since they are improved versions of their predecessor.

Table 5.6 Ranking of the Algorithms Based on Two Set Coverage Metric and for VSP₁

<table>
<thead>
<tr>
<th>Elitism</th>
<th>Rank</th>
<th>Algorithm VSP₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elitist</td>
<td>1</td>
<td>SPEA-II</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>NSGA-II</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>SPEA</td>
</tr>
<tr>
<td>Non-Elitist</td>
<td>4</td>
<td>NSGA</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>VEGA</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>WBGA</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>MOGA</td>
</tr>
</tbody>
</table>

136
Table 5.7 Ranking of the Algorithms Based on *Two Set Coverage* Metric With Respect to *VSP₂*

<table>
<thead>
<tr>
<th>Elitism</th>
<th>Rank</th>
<th>Algorithm <em>VSP₂</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Elitist</td>
<td>1</td>
<td>SPEA-II</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NSGA-II</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>SPEA</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>NSGA</td>
</tr>
<tr>
<td>Non-Elitist</td>
<td>4</td>
<td>VEGA</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>NPGA</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>WBGA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MOGA</td>
</tr>
</tbody>
</table>

5.2 Hypervolume Metric Results

Table 5.10 and Table 5.11 show the mean values of the *Hypervolume* metric as well as the variance of the values for *VSP₁* and *VSP₂*. Larger values of the *Hypervolume* are better since they represent a larger area in the objective space which is covered by a set of solutions. The rows in Table 5.10 and Table 5.11 are ordered based on the descending values of the mean column so that each row in the first column represents the rank of the algorithm. In addition, Table 5.8 and Table 5.9 show multiple comparisons of *Hypervolume* metric for *VSP₁* and *VSP₂* respectively.

Table 5.8 Multiple Comparison of *Hypervolume* for *VSP₁*

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>126.98</td>
<td>345.81</td>
<td>563.78</td>
<td>928.92</td>
<td>895.56</td>
<td>1061.21</td>
<td>1368.64</td>
<td></td>
</tr>
<tr>
<td>NSGA-II</td>
<td>218.83</td>
<td>436.80</td>
<td>801.94</td>
<td>768.58</td>
<td>934.23</td>
<td>1241.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEA</td>
<td>217.97</td>
<td>583.10</td>
<td>549.75</td>
<td>715.40</td>
<td>987.22</td>
<td>1022.83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSGA</td>
<td>365.13</td>
<td>331.78</td>
<td>497.42</td>
<td>804.86</td>
<td>132.29</td>
<td>439.73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEGA</td>
<td>33.35*</td>
<td>596.04</td>
<td>987.22</td>
<td>761.96</td>
<td>1312.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td>165.64</td>
<td>473.08</td>
<td>716.36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td>307.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.9 Multiple Comparison of *Hypervolume* for *VSP₂*

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>161.46*</td>
<td>256.28</td>
<td>376.82</td>
<td>852.32</td>
<td>1243.50</td>
<td>1018.24</td>
<td>1568.68</td>
<td></td>
</tr>
<tr>
<td>NSGA-II</td>
<td>94.82</td>
<td>215.36</td>
<td>690.86</td>
<td>1082.05</td>
<td>856.78</td>
<td>1407.23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEA</td>
<td>120.54*</td>
<td>596.04</td>
<td>987.22</td>
<td>761.96</td>
<td>1312.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSGA</td>
<td>475.50</td>
<td>866.68</td>
<td>641.42</td>
<td>1191.86</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEGA</td>
<td>391.18</td>
<td>165.92</td>
<td>716.36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td>-225.27</td>
<td>325.18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td></td>
<td>550.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
According to Table 5.10 SPEA-II outperforms NSGA-II for VSP₁. However, from Table 5.11 it is observed that both SPEA-II and NSGA-II give the highest values for VSP₂ since there is no significant difference between them. In addition, based on the hypervolume metric, the performance of the most non-elitist algorithms is inferior to the elitist algorithms.

Among the non-elitist algorithms NSGA is the most promising one while the VEGA and WBGA exhibit fair performance. The results also show that the MOGA algorithm is the weakest algorithm in terms of the hypervolume metric. The performance gap which is seen between the SPEA and NSGA in VSP₁ could be because of the lack of the elitism mechanism in NSGA. However, in VSP₂ which is the smaller problem instance, both algorithms exhibit similar performance.

The results for the hypervolume metric are almost supported by results of the two set coverage because both of the metrics evaluate the same aspects which is the closeness of solutions to the pareto optimal set.

Table 5.10 Mean and variance values of the Hypervolume metric for VSP₁

<table>
<thead>
<tr>
<th>Rank</th>
<th>Algorithm</th>
<th>Mean</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SPEA-II</td>
<td>6032.43</td>
<td>50842.88</td>
</tr>
<tr>
<td>2</td>
<td>NSGA-II</td>
<td>5905.45</td>
<td>12025.93</td>
</tr>
<tr>
<td>3</td>
<td>SPEA</td>
<td>5686.62</td>
<td>132297.39</td>
</tr>
<tr>
<td>4</td>
<td>NSGA</td>
<td>5468.65</td>
<td>95957.57</td>
</tr>
<tr>
<td>5</td>
<td>WBGA</td>
<td>5136.87</td>
<td>36853.64</td>
</tr>
<tr>
<td>6</td>
<td>VEGA</td>
<td>5103.51</td>
<td>10601.85</td>
</tr>
<tr>
<td>7</td>
<td>NPGA</td>
<td>4971.22</td>
<td>26155.29</td>
</tr>
</tbody>
</table>

Table 5.11 Mean and Variance Values of the Hypervolume Metric for VSP₂

<table>
<thead>
<tr>
<th>Rank</th>
<th>Algorithm</th>
<th>Mean</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SPEA-II</td>
<td>6060.84</td>
<td>213843.74</td>
</tr>
<tr>
<td>2</td>
<td>NSGA-II</td>
<td>5899.39</td>
<td>58051.47</td>
</tr>
<tr>
<td>3</td>
<td>SPEA</td>
<td>5804.02</td>
<td>93335.57</td>
</tr>
<tr>
<td>4</td>
<td>NSGA</td>
<td>5684.02</td>
<td>53668.22</td>
</tr>
<tr>
<td>5</td>
<td>VEGA</td>
<td>5208.52</td>
<td>77758.32</td>
</tr>
<tr>
<td>6</td>
<td>NPGA</td>
<td>5042.61</td>
<td>9143.25</td>
</tr>
<tr>
<td>7</td>
<td>WBGA</td>
<td>4817.34</td>
<td>129496.55</td>
</tr>
<tr>
<td>8</td>
<td>MOGA</td>
<td>4492.16</td>
<td>86648.48</td>
</tr>
</tbody>
</table>
5.3 Result for Spacing metric

Table 5.14 and Table 5.15 shows the mean and variance of values for the *Spacing* metric. The Table’s rows are ordered based on the ascending order of the mean and are ranked accordingly. The lower *Spacing* values are regarded as better values since they indicate less variation between distances and therefore the solutions are near uniformly spaced (Deb, 2001). In addition, Table 5.12 and Table 5.13 show multiple comparisons of *Spacing* metric for *VSP*₁ and *VSP*₂ respectively. It can be observed from the Table 5.14 and Table 5.15 that *SPEA-II* performs well with respect to the population diversity which reveals its ability to preserve a well distributed set of solutions. The results also show that spread of solutions returned by *SPEA* is similar to *NSGA-II* for

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>-0.25</td>
<td>-0.20</td>
<td>-0.65</td>
<td>-0.90</td>
<td>-0.96</td>
<td>-1.20</td>
<td>-1.41</td>
<td></td>
</tr>
<tr>
<td>NSGA-II</td>
<td></td>
<td>0.04*</td>
<td>-0.41</td>
<td>-0.65</td>
<td>-0.72</td>
<td>-0.95</td>
<td>-1.17</td>
<td></td>
</tr>
<tr>
<td>SPEA</td>
<td></td>
<td></td>
<td>-0.45</td>
<td>-0.69</td>
<td>-0.76</td>
<td>-0.99</td>
<td>-1.21</td>
<td></td>
</tr>
<tr>
<td>NSGA</td>
<td></td>
<td></td>
<td></td>
<td>-0.25</td>
<td>-0.31</td>
<td>-0.54</td>
<td>-0.76</td>
<td></td>
</tr>
<tr>
<td>VEGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.07*</td>
<td>-0.30</td>
<td>-0.52</td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.23</td>
<td>-0.45</td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.22</td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>-0.48</td>
<td>-0.10</td>
<td>-0.52</td>
<td>-0.85</td>
<td>-0.94</td>
<td>-1.30</td>
<td>-1.17</td>
<td></td>
</tr>
<tr>
<td>NSGA-II</td>
<td></td>
<td>0.39</td>
<td>-0.04</td>
<td>-0.36</td>
<td>-0.46</td>
<td>-0.82</td>
<td>-0.68</td>
<td></td>
</tr>
<tr>
<td>SPEA</td>
<td></td>
<td></td>
<td>-0.43</td>
<td>-0.75</td>
<td>-0.84</td>
<td>-1.21</td>
<td>-1.07</td>
<td></td>
</tr>
<tr>
<td>NSGA</td>
<td></td>
<td></td>
<td></td>
<td>-0.32</td>
<td>-0.42</td>
<td>-0.78</td>
<td>-0.65</td>
<td></td>
</tr>
<tr>
<td>VEGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.09*</td>
<td>-0.46</td>
<td>-0.32</td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.36</td>
<td>-0.23</td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 5.14 Mean and Variance Values for Spacing Metric for VSP₁

<table>
<thead>
<tr>
<th>Rank</th>
<th>VSP 1</th>
<th>Mean</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SPEA-II</td>
<td>0.20</td>
<td>0.004</td>
</tr>
<tr>
<td>2</td>
<td>SPEA</td>
<td>0.40</td>
<td>0.021</td>
</tr>
<tr>
<td>3</td>
<td>NSGA-II</td>
<td>0.45</td>
<td>0.018</td>
</tr>
<tr>
<td>4</td>
<td>NSGA</td>
<td>0.85</td>
<td>0.123</td>
</tr>
<tr>
<td>5</td>
<td>VEGA</td>
<td>1.10</td>
<td>0.170</td>
</tr>
<tr>
<td>6</td>
<td>WBGA</td>
<td>1.16</td>
<td>0.086</td>
</tr>
<tr>
<td>7</td>
<td>NPGA</td>
<td>1.40</td>
<td>0.144</td>
</tr>
<tr>
<td>8</td>
<td>MOGA</td>
<td>1.61</td>
<td>0.140</td>
</tr>
</tbody>
</table>

Table 5.15 Mean and Variance Values for Spacing Metric for VSP₂

<table>
<thead>
<tr>
<th>Rank</th>
<th>VSP 2</th>
<th>Mean</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SPEA-II</td>
<td>0.30</td>
<td>0.004</td>
</tr>
<tr>
<td>2</td>
<td>SPEA</td>
<td>0.39</td>
<td>0.012</td>
</tr>
<tr>
<td>3</td>
<td>NSGA-II</td>
<td>0.78</td>
<td>0.026</td>
</tr>
<tr>
<td>4</td>
<td>NSGA</td>
<td>0.82</td>
<td>0.061</td>
</tr>
<tr>
<td>5</td>
<td>VEGA</td>
<td>1.14</td>
<td>0.228</td>
</tr>
<tr>
<td>6</td>
<td>WBGA</td>
<td>1.24</td>
<td>0.098</td>
</tr>
<tr>
<td>7</td>
<td>MOGA</td>
<td>1.46</td>
<td>0.232</td>
</tr>
<tr>
<td>8</td>
<td>NPGA</td>
<td>1.60</td>
<td>0.113</td>
</tr>
</tbody>
</table>

VSP₁ while SPEA outperforms in VSP₂. This is also supported by the work of (Deb, Mohan, & Mishra, 2003). However, the NPGA and MOGA are amongst the poorest algorithms in terms of the Spacing algorithm. VEGA and WBGA also show similar performance in both problem instances.

5.4 Maximum Spread Metric Results

Table 5.18 and Table 5.19 present the mean and variance of values for the Maximum Spread metric. Larger values are better since they indicate the solutions are spanned over larger region of the objective space. The Table’s rows are ordered based on the descending values of the mean column and each row in the first column represents the rank for a particular algorithm. In addition, Table 5.16 and Table 5.17 show multiple comparisons of Maximum Spread metric for VSP₁ and VSP₂ respectively.

From Table 5.18 it can be seen that the SPEA-II is best in VSP₁. However according to the results in Table 5.19 the difference between two algorithm’s means is not
statistically significant in VSP₂. The results also show that WBGA performs worst among all the algorithms while algorithms NSGA exhibit a fair performance.

Table 5.16 Multiple Comparison for Maximum Spread Metric for VSP₁

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>1.65</td>
<td>10.25</td>
<td>8.60</td>
<td>13.72</td>
<td>5.13</td>
<td>8.47</td>
<td>12.11</td>
<td>-7.93</td>
</tr>
<tr>
<td>NSGA-II</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.17 Multiple Comparison for Maximum Spread Metric for VSP₂

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>-0.03*</td>
<td>8.22</td>
<td>14.69</td>
<td>32.13</td>
<td>35.48</td>
<td>27.02</td>
<td>24.26</td>
<td></td>
</tr>
<tr>
<td>NSGA-II</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.18 Mean and Variance values of the Maximum Spread Metric for VSP₁

<table>
<thead>
<tr>
<th>Rank</th>
<th>VSP 1</th>
<th>Mean</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SPEA-II</td>
<td>55.00</td>
<td>1.253</td>
</tr>
<tr>
<td>2</td>
<td>NSGA-II</td>
<td>53.35</td>
<td>0.163</td>
</tr>
<tr>
<td>3</td>
<td>SPEA</td>
<td>44.76</td>
<td>7.405</td>
</tr>
<tr>
<td>4</td>
<td>NSGA</td>
<td>39.63</td>
<td>18.327</td>
</tr>
<tr>
<td>5</td>
<td>VEGA</td>
<td>31.17</td>
<td>44.102</td>
</tr>
<tr>
<td>6</td>
<td>MOGA</td>
<td>29.86</td>
<td>84.414</td>
</tr>
<tr>
<td>7</td>
<td>NPGA</td>
<td>26.99</td>
<td>42.329</td>
</tr>
<tr>
<td>8</td>
<td>WBGA</td>
<td>19.05</td>
<td>31.753</td>
</tr>
</tbody>
</table>

Table 5.19 Mean and Variance Values of the Maximum Spread Metric for VSP₂

<table>
<thead>
<tr>
<th>Rank</th>
<th>VSP 2</th>
<th>Mean</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NSGA-II</td>
<td>54.04</td>
<td>0.294</td>
</tr>
<tr>
<td>2</td>
<td>SPEA-II</td>
<td>54.00</td>
<td>2.097</td>
</tr>
<tr>
<td>3</td>
<td>SPEA</td>
<td>45.78</td>
<td>10.547</td>
</tr>
<tr>
<td>4</td>
<td>NSGA</td>
<td>39.32</td>
<td>14.608</td>
</tr>
<tr>
<td>5</td>
<td>MOGA</td>
<td>29.74</td>
<td>56.439</td>
</tr>
<tr>
<td>6</td>
<td>NPGA</td>
<td>26.98</td>
<td>50.921</td>
</tr>
<tr>
<td>7</td>
<td>VEGA</td>
<td>21.87</td>
<td>21.216</td>
</tr>
<tr>
<td>8</td>
<td>WBGA</td>
<td>18.52</td>
<td>29.203</td>
</tr>
</tbody>
</table>
5.5 Visual Comparison for 30 runs

The final populations from all 30 independent runs for each algorithm were combined to form an accumulated population and thereafter the non-dominated solutions were identified. The non-dominated solutions are visualized in Figure 5.2 and Figure 5.3 corresponding to the VSP$_1$ and VSP$_2$ respectively.

From the Figure 5.2 and Figure 5.3 it can observed that the SPEA-II and NSGA-II perform best among all the algorithms with respect to convergence because the curve of...
Figure 5.3 Non-Dominated Front Obtained By Each Evolutionary Algorithm Solving VSP$_2$

solutions for these two algorithms are closer to the point (0,0). In addition the extent of the solutions of these two algorithms is much larger than the other algorithms. However, MOGA is the poorest algorithm with respect to convergence and extent of solutions. Furthermore, it can be seen that solutions by VEGA is denser in the center region. This is the region where both objectives are individually minimized. That implies that VEGA has a tendency to deliver good values subject to each objective rather than a distributed set of trade-off solutions. This issue is also mentioned in the work by (Deb, 2001; Nakayama et al., 2009a)

5.6 Computational Time Results

In order to compare the computational time of the algorithms the mean computational time for each algorithm in 30 runs is shown in Table 5.22 and Table 5.23. The Table’s rows are ordered based on the ascending order of the mean and each row in the first column represents the rank for a particular algorithm. In addition, Table 5.20 and
Table 5.21 show multiple comparisons of *computational time* metric for VSP$_1$ and VSP$_2$ respectively.

According to Table 5.22 and Table 5.23 most elitist algorithms except NSGA-II require more time to execute as compared to the non-elitist algorithms. VEGA appeared to be the fastest algorithm. However, as mentioned before its performance is less than the elitist algorithms in terms of diversity and convergence. Among the elitist algorithms the NSGA-II is considerably fast. This is likely due to the fast non-dominated sorting of this algorithm. In both problem instances NSGA is the slowest algorithm possibly due to its ranking and fitness sharing procedures. The result reveals that the elitism feature adds computational overhead on the evolutionary algorithm. As a result most elitist algorithms are slower than the algorithms which do not support elitism. For example the SPEA is slower than the MOGA. For each algorithm the computational time for VSP$_1$ is

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>2.59</td>
<td>0.31</td>
<td>-2.15</td>
<td>7.16</td>
<td>2.33</td>
<td>0.83</td>
<td>1.39</td>
<td></td>
</tr>
<tr>
<td>NSGA-II</td>
<td>-2.28</td>
<td>-4.73</td>
<td>4.58</td>
<td>-0.26</td>
<td>-1.75</td>
<td>-1.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEA</td>
<td>-2.45</td>
<td>6.86</td>
<td>2.02</td>
<td>0.53*</td>
<td>1.08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSGA</td>
<td>-2.48</td>
<td>9.31</td>
<td>4.48</td>
<td>2.98</td>
<td>3.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEGA</td>
<td>-4.83</td>
<td>-6.33</td>
<td>-5.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td>-1.50</td>
<td>-0.94</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td></td>
<td>0.56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.21 Multiple Comparisons for Computational Time Metric for VSP$_2$

<table>
<thead>
<tr>
<th></th>
<th>SPEA-II</th>
<th>NSGA-II</th>
<th>SPEA</th>
<th>NSGA</th>
<th>VEGA</th>
<th>WBGA</th>
<th>NPGA</th>
<th>MOGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA-II</td>
<td>2.49</td>
<td>0.64</td>
<td>-2.72</td>
<td>3.93</td>
<td>2.65</td>
<td>1.88</td>
<td>1.46</td>
<td></td>
</tr>
<tr>
<td>NSGA-II</td>
<td>-1.86</td>
<td>-5.21</td>
<td>1.43</td>
<td>0.16*</td>
<td>-0.61</td>
<td>-1.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEA</td>
<td>-3.35</td>
<td>3.29</td>
<td>2.02</td>
<td>1.24</td>
<td>0.83</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSGA</td>
<td>6.64</td>
<td>5.37</td>
<td>4.60</td>
<td>4.18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEGA</td>
<td>-1.27</td>
<td>-2.05</td>
<td>-2.46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WBGA</td>
<td>-0.77</td>
<td>-1.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPGA</td>
<td></td>
<td>-0.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 5.22 Mean of Computational Time (in Second) for VSP₁

<table>
<thead>
<tr>
<th>Rank</th>
<th>Algorithm</th>
<th>Mean Computational Time</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VEGA</td>
<td>18.027</td>
<td>0.51</td>
</tr>
<tr>
<td>2</td>
<td>NSGA-II</td>
<td>22.604</td>
<td>0.89</td>
</tr>
<tr>
<td>3</td>
<td>WBGA</td>
<td>22.859</td>
<td>0.77</td>
</tr>
<tr>
<td>4</td>
<td>MOGA</td>
<td>23.799</td>
<td>1.05</td>
</tr>
<tr>
<td>5</td>
<td>NPGA</td>
<td>24.356</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>SPEA</td>
<td>24.883</td>
<td>2.65</td>
</tr>
<tr>
<td>6</td>
<td>SPEA-II</td>
<td>25.191</td>
<td>0.59</td>
</tr>
<tr>
<td>7</td>
<td>NSGA</td>
<td>27.337</td>
<td>6.75</td>
</tr>
</tbody>
</table>

Table 5.23 Mean of Computational Time (in Second) for VSP₂

<table>
<thead>
<tr>
<th>Rank</th>
<th>Algorithm</th>
<th>Mean Computational Time</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VEGA</td>
<td>14.769</td>
<td>0.39</td>
</tr>
<tr>
<td>2</td>
<td>NSGA-II</td>
<td>16.202</td>
<td>1.03</td>
</tr>
<tr>
<td></td>
<td>WBGA</td>
<td>16.042</td>
<td>0.65</td>
</tr>
<tr>
<td>3</td>
<td>MOGA</td>
<td>17.231</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>NPGA</td>
<td>16.816</td>
<td>0.96</td>
</tr>
<tr>
<td>5</td>
<td>SPEA</td>
<td>18.058</td>
<td>1.47</td>
</tr>
<tr>
<td>6</td>
<td>SPEA-II</td>
<td>18.694</td>
<td>0.45</td>
</tr>
<tr>
<td>7</td>
<td>NSGA</td>
<td>21.412</td>
<td>2.34</td>
</tr>
</tbody>
</table>

noticeably higher than that of VSP₂. This is explained by a larger search space of VSP₁ than VSP₂. NSGA-II as an enhanced version is significantly faster than NSGA. However, in the case of SPEA-II the older version i.e. SPEA is faster in solving VSP₂. Since SPEA-II uses a fine-grained fitness assignment strategy it has more computational time than its predecessor; SPEA.

Generally, it can be said that none of the eight (8) algorithms can be considered as the best with respect to the four performance metrics and computational time. However, in most of the metrics NSGA-II and SPEA-II perform better than the other algorithms. The result shows also that features such as elitism and sharing strategy which are implemented in SPEA-II, SPEA, NSGA-II are important factors in order to reach better convergence and diversity of solutions while at the same time it increases the computational overhead.
Chapter 6. Conclusion

This chapter presents the conclusion of this thesis. The chapter is organized as follows: Section 6.1 gives a summary of the research undertaken. Section 6.2 presents the research results and the contributions and finally the future work will be stated in Section 6.3

6.1 Summary of Research

The materialized view selection problem is considered as an important challenge in data warehouse optimization. The problem of selecting the right subset of views such that a goal is minimized is an NP-Hard problem (Gupta & Mumick, 1999). The problem received significant attention in the past. Several approaches such as greedy, Genetic Algorithm, A*, simulated annealing and etc. has been suggested (see Table 2.3). However, most of the proposed works merely consider the problem in a single objective form where either the total query response time, total update time or a combination of these are taken into consideration. The multi-objective view selection is an innovative approach to the problem and refers to selecting a subset of views such that both goals, that is the total query response time and the total view update time is minimized simultaneously. On the other hand, evolutionary algorithms are regarded as a promising candidate to solve the general multi-objective problems (Deb, 2001). The application of these algorithms were investigated in several optimization problems with multiple objectives in different areas (Coello, 2007; Deb, 2001; Goldberg, 1989; Yu & Gen, 2010). However, in the field of the view selection problem in the multi-objective variation no published comprehensive and comparative study has been carried out. This research is about the application of evolutionary multi-objective optimization
algorithms in the multi-objective view selection problem. As a comparative study a number of well-known evolutionary algorithms were applied to the multi-objective view selection problem.

The entire architecture for the proposed object oriented model is classified into two different domains:

- Problem domain
- Methods domain

The problem domain includes all relevant classes to the problem such as Lattice, view and VSP problem instance. The methods domain includes all classes that are relevant to the methods. The classes in the methods domain are divided in two different groups: The shell classes and core classes. The core classes are fundamental classes which are used as a basic part in the shell classes. Examples of the core classes are the individual, population, selection operator, mutation operator and crossover operator. The shell classes implement a fully standalone evolutionary algorithm and can be executed independently. These algorithms rely on ready-made classed in the core area. The advantage of such a classification is that any time in future, the problems and methods can be replaced to other problems and methods.

In order to deal with the disk space constraint of the view selection problem in this research, constrain dominance is used for constraint handling, since it was shown as a promising technique in (Deb, 2001). The technique slightly modifies the definition of the original dominance concept so as to make the right decisions about the infeasible solutions encountered.
Most view selection algorithms require the knowledge of the size of the views. However, the exact size of views would be obtainable only by creating and storing the view. For view size estimation the Cardenas’ formula (Cardenas, 1975) was used.

6.2 Contribution and results

Two different goals for the evolutionary multi-objective algorithms are (Mumford & Jain, 2009):

1. Finding a set of solutions which are close to the true pareto optimal set.
2. Finding a set of solutions that are well distributed.

In designing the performance metric for the evolutionary multi-objective algorithms these two goals are taken into consideration. Several performance metrics (Deb, 2001) are suggested for the assessment of the evolutionary multi-objective algorithms. Generally, the metrics are classified in three groups: the convergence based metric which measure a set of obtained solutions based on the first goal, diversity based metrics which evaluates the set of solutions based on the second goal and hybrid metrics which are meant to measure the performance based on both of the two above goals.

Two problem instances, called VSP₁ and VSP₂, is derived from a synthetic database populated according to the TPC-H proposal ("The TPC Benchmark™H," 2011). The size of the search space for VSP₁ and VSP₂ is $2^{64}$ and $2^{48}$ respectively.

For evaluating the performance of the algorithms studied in this research, Two Sets Coverage as convergence based, Hypervolume as a hybrid metric, spacing and maximum spread as a diversity based was used. It is to be noted that some metrics require the knowledge of the true pareto optimal set. Examples of such measures are the Error Ratio, Generational Distance and Spread (Deb, 2001). However, in the case of the view selection problem these metrics were not applicable since the set of true pareto
optimal set is unknown. The outcomes of all the eight (8) algorithms in 30 different runs with different initial population were compared based on these three metrics. In addition, the computational times for these algorithms were also compared.

It is to be noted that the contribution of this research is limited to the multi-objective view selection problem area. The general contributions of this research (with respect to the multi-objective view selection problem) are as follows:

- Identification of the algorithm which performed well (as compared to various others) in solving the multi-objective view selection problem; and these algorithms are namely, *SPEA-II* and *NSGA-II*. These two algorithms is recommendation of this research for solving multi-objective view selection problem.
- Our findings show that the elitist algorithms (*SPEA-II, SPEA, and NSGA-II*) perform better than the non-elitist algorithms (*MOGA, NPGA, WBGA* and *VEGA*) in solving the multi-objective view selection problem.
- The strategies such as fitness sharing and crowding help in the diversity of the solutions to the multi-objective view selection problem.
- In solving the multi-objective view selection problem, although using a secondary population for preserving the best ever found solutions helps to 1) give a more distributed solution; and 2) obtain a set of solutions which are closer to the optimal solution, however at the same time managing the secondary population increases the computational complexity of the algorithm.

### 6.3 Future Work

Future perspective on the view selection problem can be the investigation of the following items:
• Study the application of other possible meta-heuristic such as the ant colony optimization, Particle swarm optimization, Bee algorithms on the multi-objective view selection problem

• A new evolutionary multi-objective algorithm developed by combining good features of different evolutionary multi-objective algorithms.

• Investigate the application of parallel genetic algorithms on the multi-objective view selection problem.
References


Appendix A. Visual Basic Code

A.1 Classes in Method Domain

A.1.1 Core Classes

A.1.1.1 Individuals Class

Imports GA_for_View_Selection.General
Namespace GeneticAlgorithm
<Serializable()> Public Class Individual
    Implements System.IEquatable(Of Individual)
    Private GList() As Integer

    Public Objective1Value As Double
    Public Objective2Value As Double
    Public ConstraintValue As Double
    Public Fitness As Double
    Public Value1 As Double
    Public Value2 As Double
    Public Value3 As Double
    Public Value4 As Double
    Public Rank As Short
    Public CrowdingDistance As Double

    Sub New(ByVal Size As Integer)
        ReDim GList(Size - 1)
    End Sub

    Default Public Property Gene(ByVal i As Short) As Integer
        Get
            Return GList(i)
        End Get
        Set(ByVal value As Integer)
            GList(i) = value
        End Set
    End Property
Public Function(EqualTo(ByVal I As Individual) As Boolean Implements IEquatable(Of Individual).Equals
    If GList.SequenceEqual(I.GList) = True Then
        Return True
    Else
        Return False
    End If
End Function

Public ReadOnly Property Count() As Integer
    Get
        Return GList.Count
    End Get
End Property

Public Overrides Function ToString() As String
    Return GList.ToString()
End Function

Public Function IsFeasible() As Boolean
    Return ConstraintValue <= 0
End Function

Public Shared Operator =(ByVal x As Individual, ByVal y As Individual) As Boolean
    If x.GList.SequenceEqual(y.GList) Then
        Return True
    Else
        Return False
    End If
End Operator

Public Shared Operator <>(ByVal x As Individual, ByVal y As Individual) As Boolean
    If x.GList.SequenceEqual(y.GList) = False Then
        Return True
    Else
        Return False
    End If
End Operator

Public Sub Evaluate(ByVal Objective1 As Func(Of Object, Double), ByVal Objective2 As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
    Dim a As VSPPhenotype

    a = DecodeToPhenotype()

    Objective1Value = Objective1(a)
    Objective2Value = Objective2(a)
    ConstraintValue = Constraint(a)
End Sub

Public Function Clone() As Individual
    Dim I As Individual
    I = MemberwiseClone()
    I.GList = GList.Clone()
    Return I
End Function

Public Sub Random()
    For i = 0 To Count - 1
        GList(i) = Random01()
    Next
End Sub

Public Function Dominate(ByVal P As Population) As Population
    'individuals in P which this individual dominates
    Dim DominatedIndividuals As New Population
    For Each I As Individual In P
        If GA.Dominate(Me, I) = True Then
            DominatedIndividuals.Add(I)
        End If
    Next
    Return DominatedIndividuals
End Function

Public Function DominateAny(ByVal P As Population) As Boolean
    'Checkes whether this individual dominates population P or not
    Dim DominatedIndividuals As New Population
    For Each I As Individual In P
        If GA.Dominate(Me, I) = True Then
            Return True
        End If
    Next
    Return False
End Function

Public Function Exchange(ByVal index1 As Short, ByVal index2 As Short) As Boolean
    If GList(index1) <> GList(index2) Then
        Dim temp As Short
        temp = GList(index1)
        GList(index1) = GList(index2)
        GList(index2) = temp
        Exchange = True
    Else
        Return False
    End If
Public Function DecodeToPhenotype() As VSPPhenotype
    Dim S As New VSPPhenotype(GList.Count)
    S.List = GList
    Return S
End Function

Public Function ToArray() As Array
    Return GList.ToArray()
End Function

Public Sub Flip(ByVal i As Integer)
    GList(i) = 1 - GList(i)
End Sub

Private Function Random01() As Short
    Dim R As Double
    Randomize()
    R = Rnd()
    If R <= 0.5 Then
        Return 0
    Else
        Return 1
    End If
End Function

Private Class IndividualCounter
    Implements IEnumerable
    Private Ilist As List(Of Individual)
    Private Position As Integer = -1
    Public Sub New(ByVal L As List(Of Individual))
        Ilist = L
    End Sub
    Public Function MoveNext() As Boolean Implements System.Collections.IEnumerable.MoveNext
        If Position < Ilist.Count - 1 Then
            Position = Position + 1
            Return True
        Else
            Return False
        End If
    End Function
    Public Sub Reset() Implements System.Collections.IEnumerable.Reset
        Position = -1
    End Sub
End Class
A.1.1.2 Population Class

Imports GA_for_View_Selection.General
Imports System.Math
Imports GA_for_View_Selection.GeneticAlgorithm.GA
Imports System.IO

Namespace GeneticAlgorithm

<Serializable()> Public Class Population

    Implements ICloneable
    Implements IEnumerable

    Private IList As List(Of Individual)
    Public ID As Integer
    Private DominatedBy() As List(Of Integer)
    Private Dominates() As List(Of Integer)

    Public Event AddEvent(ByVal I As Individual)
    Public Event RemoveEvent(ByVal I As Individual)

    Public Sub New()
        IList = New List(Of Individual)
    End Sub

    Public ReadOnly Property Count() As Integer
        Get
            Return IList.Count
        End Get
    End Property

    Public WriteOnly Property Rank() As Integer
        Set(ByVal value As Integer)
            For Each Individual In IList
                Individual.Rank = value
            Next
        End Set
    End Property

    Default Public Property Member(ByVal i As Integer) As Individual
        Set(ByVal value As Individual)
            IList(i) = value
        End Set
        Get
            If i < Count And i >= 0 Then
                Return IList(i)
            Else
                Return Nothing
            End If
        End Get
    End Property

End Class

End Class
End Namespace
Return IList(i)
Else
  Return Nothing
End If
End Get
End Property

Private Function IndexOf(ByVal I As Individual) As Integer
  Return IList.IndexOf(I)
End Function

Public Sub Evaluate(ByVal Objective1 As Func(Of Object, Double), ByVal Objective2 As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
For Each I As Individual In IList
  I.Evaluate(Objective1, Objective2, Constraint)
Next
End Sub

Public Function NonDominated() As Population 'Method two for finding non-dominated individuals
  Dim PartialSet As New Population
  Dim flag As Boolean
  If Count = 0 Then
    Return Nothing
  End If
  PartialSet.Add(Member(0))
  For Each a As Individual In Me
    flag = False
    For Each b As Individual In PartialSet
      If GA.Dominate(a, b) = True Then
        PartialSet.Remove(b)
      ElseIf GA.Dominate(b, a) = True Then
        flag = True
        Exit For
      End If
    Next
    If flag = False Then
      PartialSet.Add(a)
    End If
  Next
  Return PartialSet
End Function
Public Function Top() As Population 'returns top division of population
    Dim T As New Population
    T.IList = IList.GetRange(0, Count \\ 2)
    Return T
End Function

Public Function Bottom() As Population 'returns bottom division of population
    Dim B As New Population
    B.IList = IList.GetRange((Count \\ 2), (Count - Count \\ 2))
    Return B
End Function

Public Sub DominationCheck() 'Updates Dominates and DominatedBy Lists
    ReDim Dominates(Count)
    ReDim DominatedBy(Count)
    For i = 0 To Count - 1
        Dominates(i) = New List(Of Integer)
        DominatedBy(i) = New List(Of Integer)
    Next
    For i = 0 To Count - 1
        For j = i + 1 To Count - 1
            If Dominate(i, j) = True Then
                Dominates(i).Add(j)
                DominatedBy(j).Add(i)
            End If
            If Dominate(j, i) = True Then
                Dominates(j).Add(i)
                DominatedBy(i).Add(j)
            End If
        Next
    Next
End Sub

Public Sub SaveToFile(ByVal Address As String)
    Dim W As StreamWriter = New StreamWriter(Address)
    W.WriteLine("A=[ ")
    For i = 0 To Count - 1
        W.Write(Member(i).Objective1Value) W.Write(" ")
        W.Write(Member(i).Objective2Value) W.WriteLine()
    Next
    W.Write("]")
    W.Close()
End Sub

Public Function Dominate(ByVal I As Individual) As Boolean ' checks whether any individual from the population dominates individual I or not
    For Each Individual In Me
If GA.Dominate(Individual, I) = True Then
    Return True
End If

Next

Return False
End Function

Public Function Dominate(ByVal i As Integer, ByVal j As Integer) As Boolean
    Return GA.Dominate(Member(i), Member(j))
End Function

Public Function Classify() As PopulationSet
    Dim Level As Population
    Dim PSet As New PopulationSet
    Dim Rank As Integer = 1
    Dim CheckList(Count) As Boolean

    DominationCheck()
    Level = PickNonDominated(CheckList)
    UpdateDomination(Level)
    While (Level.Count > 0)
        Level.Rank = Rank
        Rank = Rank + 1
        PSet.Add(Level)
        Level = PickNonDominated(CheckList)
        UpdateDomination(Level)
    End While

    Return PSet
End Function

Private Function PickNonDominated(ByVal CheckList() As Boolean) As Population
    Dim NonDominatedPopulation As New Population
    For i = 0 To Count - 1
        If (DominatedBy(i).All(Function(B As Boolean) B = False)) And CheckList(i) = False Then
            CheckList(i) = True
            NonDominatedPopulation.Add(Member(i))
        End If
    Next

    Return NonDominatedPopulation
End Function

Private Sub UpdateDomination(ByVal Level As Population)
    Dim Ind As Short
    For i = 0 To Level.Count - 1
        Ind = IndexOf(Level(i))
        For j = 0 To Dominates(Ind).Count - 1
            DominatedBy(Dominates(Ind)(j)).Remove(Ind)
        Next
        Dominates(Ind).Clear()
    Next
End Sub

Public Function Partition_Feasibility() As PopulationSet
    Dim PS As New PopulationSet
Dim NonFeasible As New Population
Dim Feasible As New Population

For Each Individual In IList
    If Individual.IsFeasible() = True Then
        Feasible.Add(Individual)
    Else
        NonFeasible.Add(Individual)
    End If
Next

PS.Add(Feasible)
PS.Add(NonFeasible)

Return PS
End Function

Public Function ShallowClone() As Object Implements ICloneable.Clone
    Dim P As New Population
    P = DirectCast(Me.MemberwiseClone, Population)
P.IList = IList.ToList()

    Return P
End Function

Public Function DeepClone() As Object
    Dim s As New MemoryStream
    Dim B As New BinaryFormatter()
    B.Serialize(s, Me)
s.Seek(0, SeekOrigin.Begin)
    Return B.Deserialize(s)
End Function

Public Function Add(ByVal I As Individual) As Boolean
    If IList.Contains(I) = False Then
        IList.Add(I)
        RaiseEvent AddEvent(I)
        Return True
    Else
        Return False
    End If
End Function

Public Function Sum(ByVal F As Func(Of Individual, Double)) As Double
    Dim Result As Double
    Result = IList.Sum(F)
    Return Result
End Function

Public Sub Sort(ByVal F As Func(Of Individual, Double))
    Dim IC As New IndividualComparer
    IC.Element = F
    IList.Sort(IC)
End Sub
Public Function Compare(ByVal x As Individual, ByVal y As Individual, ByVal F As Func(Of Individual, Double)) As Integer

    If F(x) < F(y) Then
        Return 1
    End If

    If F(x) > F(y) Then
        Return -1
    End If

    If F(x) = F(y) Then
        Return 0
    End If

End Function

Public Shared Operator -(ByVal x As Population, ByVal y As Population) As Population

    Dim Z As Population

    Z = x.ShallowClone()
    For Each Individual In y
        If Z.Contains(Individual) Then
            Z.Remove(Individual)
        End If
    Next

    Return Z
End Operator

Public Shared Operator +(ByVal x As Population, ByVal y As Population) As Population

    Dim z As New Population
    z.ID = x.ID
    z.IList = x.IList.Union(y.IList).ToList()
    Return z
End Operator

Public Shared Operator ==(ByVal x As Population, ByVal y As Population) As Boolean

    If x.IList.SequenceEqual(y.IList) Then
        Return True
    Else
        Return False
    End If
End Operator

Public Shared Operator <>=(ByVal x As Population, ByVal y As Population) As Boolean

    If x.IList.SequenceEqual(y.IList) = False Then
        Return False
    Else
        Return True
    End If
Public Function RemoveAt(ByVal i As Integer) As Boolean
    If i < IList.Count Then
        IList.RemoveAt(i)
        RaiseEvent RemoveEvent(IList(i))
        Return True
    Else
        Return False
    End If
End Function

Public Function Remove(ByVal I As Individual) As Boolean
    IList.Remove(I)
    RaiseEvent RemoveEvent(I)
End Function

Public Sub Clear()
    IList.Clear()
    ID = 0
End Sub

Public Function Contains(ByVal x As Individual) As Boolean
    Return IList.Contains(x)
End Function

Public Sub RandomGenerate(ByVal PSize As Integer, ByVal ISize As Integer)
    Dim x As Individual
    Clear()
    While Count < PSize
        x = New Individual(ISize)
        x.Random()
        Add(x)
    End While
End Sub

Public Sub AssignCrowdingDistance()
    Dim List1 As List(Of Individual) = IList.ToList()
    Dim List2 As List(Of Individual) = IList.ToList()
    Dim Max1, Max2, Min1, Min2 As Double
    List1.Sort(Function(a As Individual, b As Individual)
        a.Objective1Value < b.Objective1Value)
    List2.Sort(Function(a As Individual, b As Individual)
        a.Objective2Value < b.Objective2Value)
    Min1 = List1(0).Objective1Value
    Max1 = List1(Count - 1).Objective1Value
Min2 = List2(0).Objective1Value
Max2 = List2(Count - 1).Objective2Value

List1(0).CrowdingDistance = Double.MaxValue
List2(0).CrowdingDistance = Double.MaxValue
List1(Count - 1).CrowdingDistance = Double.MaxValue
List2(Count - 1).CrowdingDistance = Double.MaxValue

For i = 1 To Count - 2
List1(i).CrowdingDistance = Calc(List1(i + 1).Objective1Value, List1(i - 1).Objective1Value, Max1, Min1)
Next

For i = 1 To Count - 2
List2(i).CrowdingDistance = List2(i).CrowdingDistance + Calc(List2(i + 1).Objective2Value, List2(i - 1).Objective2Value, Max2, Min2)
Next

End Sub
Private Function Calc(ByVal Right As Double, ByVal Left As Double, ByVal MaxF As Double, ByVal MinF As Double) As Double
Dim a, b, result As Double
a = Right - Left
b = MaxF - MinF
result = a / b
Return result
End Function
Public Sub FitnessSharing(ByVal alpha As Short, ByVal SigmaShare As Double)
Dim NC As Double
For i = 0 To Count - 1
NC = NicheCount(Member(i), alpha, SigmaShare)
Member(i).Fitness = Member(i).Fitness / NC
Next
End Sub

Public Function NicheCount(ByVal a As Individual, ByVal alpha As Short, ByVal SigmaShare As Double) As Double
Dim Sum As Double = 0
Dim d As Double
For i = 0 To Count - 1
    d = Distance(a, IList(i), DistanceCalculationType.Objectives)
    Sum += SharingFunction(d, alpha, SigmaShare)
Next
Return Sum
End Function

Private Function SharingFunction(ByVal d As Double, ByVal alpha As Short, ByVal SigmaShare As Double) As Double
Dim s As Double
If \( d \leq \Sigma \text{Share} \) Then
\[ s = 1 - \text{Pow}((d / \Sigma \text{Share}), \alpha) \]
Return \( s \)
Else
Return \( 0 \)
End If
End Function

Public Function Min(ByVal F As Func(Of Individual, Double)) As Double
Return IList.Min(F)
End Function

Public Function Max(ByVal F As Func(Of Individual, Double)) As Double
Return IList.Max(F)
End Function

Public Function Find(ByVal Pre As Predicate(Of Individual)) As Individual
Return IList.Find(Pre)
End Function

Public Function FindMax(ByVal F As Func(Of Individual, Double)) As Individual
Dim Max As Double = Double.MinValue
Dim MaxIndividual As Individual
For Each Individual In IList
If F(Individual) > Max Then
Max = F(Individual)
MaxIndividual = Individual
End If
Next
Return MaxIndividual
End Function

Public Function FindMin(ByVal F As Func(Of Individual, Double)) As Individual
Dim Min As Double = Double.MaxValue
Dim MinIndividual As Individual
For Each Individual In IList
If F(Individual) < Min Then
Min = F(Individual)
MinIndividual = Individual
End If
Next
Return MinIndividual
End Function

Public Sub ForEach(ByVal Action As Action(Of Individual))
IList.ForEach(Action)
End Sub

Private Function GetEnumerator() As System.Collections.IEnumerator
Implements System.Collections.IEnumerable.GetEnumerator
Return New IndividualCounter(IList)
End Function

Public Function DoClustering(ByVal Size As Short) As PopulationSet
Dim ClusterList As New PopulationSet
Dim Cluster As Population
Dim TobeMerged_first, ToBeMergedSecond As Short

For i = 0 To Count - 1
    Cluster = New Population
    Cluster.Add(Member(i))
    ClusterList.Add(Cluster)
Next

While ClusterList.Count > Size
    ClusterList.DistanceList(TobeMerged_first, ToBeMergedSecond)
    ClusterList.Merge(TobeMerged_first, ToBeMergedSecond)
End While

Return ClusterList
End Function

Private Function SimpleDistance(ByVal x1 As Double, ByVal y1 As Double, ByVal x2 As Double, ByVal y2 As Double) As Double
    Return ((x1 - y1) ^ 2 + (x2 - y2) ^ 2) ^ 0.5
End Function

Public Function Representative() As Individual
    Dim sum1, sum2, x, y, D As Double
    Dim MinDistance As Double = Double.MaxValue
    Dim MinDistanceIndividual As Individual
    If Count = 1 Then
        Return Member(0)
    End If
    For Each I As Individual In IList
        sum1 = sum1 + I.Objective1Value
        sum2 = sum2 + I.Objective2Value
    Next
    x = sum1 / Count
    y = sum2 / Count
    For Each I As Individual In IList
        D = SimpleDistance(I.Objective1Value, I.Objective2Value, x, y)
        If D < MinDistance Then
            MinDistance = D
            MinDistanceIndividual = I
        End If
    Next
    Return MinDistanceIndividual
End Function

Public Function Representative2() As Individual
    Dim MinDistance As Double = Double.MaxValue
    Dim MinIndividualIndex As Short
    Dim Sum As Double = 0
    Dim i As Short
    For i = 0 To Count - 1
        Sum = 0
        For j = i + 1 To Count - 1
            Sum = Sum + Distance(Member(i), Member(j), DistanceCalculationType.Objectives)
        Next
        Sum = Sum / Count
        If Sum < MinDistance Then
            MinDistance = Sum
        End If
    Next
MinIndividualIndex = i
End If
Next
Return Member(MinIndividualIndex)
End Function

Public Function SpecialAdd(ByVal I As Individual) As Boolean
Dim TobeRemoved As New Population
For Each J As Individual In IList
  If J.Dominate(I) Then
    Return False
  End If
  If I.Dominate(J) Then
    TobeRemoved.Add(J)
  End If
Next
For Each J As Individual In TobeRemoved
  Remove(J)
Next
Add(I)
Return True
End Function

Public Function SuggestSigmaShare() As Double
Dim Y1min As Double = Double.MaxValue
Dim Y1max As Double = Double.MinValue
Dim Y2min As Double = Double.MaxValue
Dim Y2max As Double = Double.MinValue
Dim SigmaShare As Double
If Count = 1 Then
  Return 1
End If
For i = 0 To IList.Count - 1
  If IList(i).Objective1Value < Y1min Then
    Y1min = IList(i).Objective1Value
  End If
  If IList(i).Objective1Value > Y1max Then
    Y1max = IList(i).Objective1Value
  End If
  If IList(i).Objective2Value < Y2min Then
    Y2min = IList(i).Objective2Value
  End If
  If IList(i).Objective2Value > Y2max Then
    Y2max = IList(i).Objective2Value
  End If
Next

SigmaShare = ((Y1max - Y1min) + (Y2max - Y2min)) / (Count - 1)
Return SigmaShare
End Function

Private Class IndividualComparer
  Implements IComparer(Of Individual)
  Public Element As Func(Of Individual, Double)
  Public Sub New()
Public Sub New(ByVal F As Func(Of Individual, Double))
    Element = F
End Sub

Public Function Compare(ByVal x As Individual, ByVal y As Individual) As Integer Implements IComparer(Of GA_for_View_Selection.GeneticAlgorithm.Individual).Compare
    Return Element(x) > Element(y)
End Function

End Class

End Class

End Namespace

A.1.1.3 GA

Imports GA_for_View_Selection.General
Imports System.Math
Imports GA_for_View_Selection.ViewSelection

Namespace GeneticAlgorithm
    Public MustInherit Class GA

        Public ChromosomeSize As Short
        Public PopulationSize As Integer = 100
        Public MaximumGeneration As Integer = 100
        Protected GenerationNumber As Integer = 0

        Public Property CrossoverRate() As Double
            Get
                Return Crossover.Rate
            End Get
            Set(ByVal value As Double)
                Crossover.Rate = value
            End Set
        End Property

        Public Property MutationRate() As Double
            Get
                Return Mutation.Rate
            End Get
            Set(ByVal value As Double)
                Mutation.Rate = value
            End Set
        End Property

        Public Shared Function Dominate(ByVal a As Individual, ByVal b As Individual) As Boolean
Dim flag As Boolean = False

If a.Objective1Value > b.Objective1Value Then
    Return False
End If
If a.Objective1Value < b.Objective1Value Then
    flag = True
End If

If a.Objective2Value > b.Objective2Value Then
    Return False
End If
If a.Objective2Value < b.Objective2Value Then
    flag = True
End If

Return flag
End Function

Public Shared Function ConstrainedDominate(ByVal a As Individual, ByVal b As Individual) As Boolean
If a.IsFeasible = False And b.IsFeasible = False Then
    Return a.ConstraintValue < b.ConstraintValue
End If
If a.IsFeasible = False And b.IsFeasible = True Then
    Return False
End If
If a.IsFeasible = True And b.IsFeasible = False Then
    Return True
End If
If a.IsFeasible = True And b.IsFeasible = True Then
    Return Dominate(a, b)
End If
End Function

Public Shared Function Distance(ByVal a As Individual, ByVal b As Individual, ByVal type As DistanceCalculationType) As Double
Select Case type
    Case DistanceCalculationType.Variables
        Return VariableDistance(a, b)
    Case DistanceCalculationType.Objectives
        Return ObjectiveDistance(a, b)
    Case DistanceCalculationType.Genotypic
End Select
End Function

Public Shared Function VariableDistance(ByVal a As Individual, ByVal b As Individual) As Double
    Dim Sum As Double = 0
    For i = 0 To a.Count() - 1
        Sum += Pow((a(i) - b(i)), 2)
    End For
    Return Sum
End Function
Public Shared Function ObjectiveDistance(ByVal a As Individual, ByVal b As Individual) As Double
    Dim PartA, PartB As Double
    Dim Sum As Double = 0
    Dim Distance As Double = 0
    PartA = Pow((a.Objective1Value - b.Objective1Value), 2)
    PartB = Pow((a.Objective2Value - b.Objective2Value), 2)
    Sum = PartA + PartB
    Distance = Pow(Sum, 0.5)
    Return Distance
End Function

Public Enum DistanceCalculationType
    Variables = 0
    Objectives = 1
    Genotypic = 2
End Enum
End Class

End Namespace

A.1.1.4 Crossover

Namespace GeneticAlgorithm
    Public Class Crossover
        Public Shared Rate As Double

        Public Shared Sub Uniform(ByRef x As Individual, ByRef y As Individual)
            Dim i, R As Double
            For i = 0 To x.Count - 1
                If (x(i) <> y(i)) Then
                    Randomize()
                    R = Rnd()
                    If R < Rate Then
                        Exchange(x(i), y(i))
                    End If
                End If
            Next
        End Sub
    End Class
End Namespace
Public Shared Sub SinglePoint(ByVal x As Individual, ByVal y As Individual, ByRef Offspring1 As Individual, ByRef Offspring2 As Individual)
    Dim Site As Integer
    Dim R As Double
    Offspring1 = New Individual(x.Count)
    Offspring2 = New Individual(y.Count)
    Randomize()
    R = Rnd()
    Site = Int(Rnd() * x.Count)
    For i = 0 To Site
        Offspring1(i) = x(i)
        Offspring2(i) = y(i)
    Next
    If R > Rate Then
        For i = Site + 1 To x.Count - 1
            Offspring1(i) = y(i)
            Offspring2(i) = x(i)
        Next
    Else
        For i = Site + 1 To x.Count - 1
            Offspring1(i) = x(i)
            Offspring2(i) = y(i)
        Next
    End If
End Sub

Public Shared Sub Exchange(ByRef a As Integer, ByRef b As Integer)
    Dim temp As Integer
    temp = a
    a = b
    b = temp
End Sub

End Class
End Namespace

A.1.1.5 Mutation

Namespace GeneticAlgorithm
Public Class Mutation
    Public Shared Rate = 0.01
    Public Shared Sub Uniform(ByRef x As Individual)
        Dim R As Double
        For i = 0 To x.Count - 1
            Randomize()
            R = Rnd()
            If R < Rate Then
                x.Flip(i)
            End If
        Next
    End Sub
End Class
End Namespace
Public Shared Sub Random(ByRef x As Individual)
    Dim index As Integer
    Randomize()
    index = Int(Rnd() * x.Count)
    x.Flip(index)
End Sub

Public Shared Sub Swap(ByRef x As Individual)
    Dim S, S1, S2 As Double
    Dim Original As Individual
    Original = x.Clone
    Randomize()
    S = Rnd()
    If S < Rate Then
        S1 = Int(Rnd() * x.Count)
        S2 = Int(Rnd() * x.Count)
        x.Exchange(S1, S2)
    End If
End Sub
End Class
End Namespace

A.1.1.6 Selection

Imports System.Math
Namespace GeneticAlgorithm
    Public Class Selection

        Public Shared Function RouletteWheel(ByVal P As Population, ByVal F As Func(Of Individual, Double)) As Individual
            Dim i, sum, PartialSum As Double
            Dim R As Double
            sum = P.Sum(F)
            Randomize()
            R = sum * Rnd()
            For i = 0 To P.Count - 1
                PartialSum = PartialSum + F(P(i))
                If PartialSum > R Then
                    Return P(i)
                End If
            Next
        End Function

        Public Shared Function Random(ByVal P As Population) As Individual
            Dim R As Integer
            Randomize()
Public Shared Function SUS(ByVal P As Population, ByVal F As Func(Of Individual, Double), ByVal N As Short) As Population
    Dim partsize As Double = 0
    Dim partialsum As Double = 0
    Dim Parents As New Population
    Dim pickednumber As Short = 0
    Dim i As Short = 0
    Dim r As Double = 0
    partsize = P.Sum(F) / N
    Randomize()
    r = Rnd() * partsize
    While (pickednumber < N)
        partialsum = partialsum + F(P(i))
        While partialsum > r + pickednumber * partsize
            Parents.Add(P(i))
            pickednumber = pickednumber + 1
        End While
        i = i + 1
    End While
    Return Parents
End Function

Public Shared Function StochasticReminderSelection(ByVal P As Population, ByVal F As Func(Of Individual, Double), ByVal n As Short) As Population
    Dim P2 As Population = P.ShallowClone
    Dim Result As New Population
    P2 = ScalePopulation(P)
    P2 = CreateParentPool(P2)
    Return P2
End Function

Public Shared Function Tournament(ByVal P As Population, ByVal Size As Integer, ByVal F As Func(Of Individual, Double), Optional ByVal FBios As FitnessBios = FitnessBios.BiggerFitness) As Individual
    Dim Pool As New Population
    Dim Ind As Individual
    Dim Winner As Individual
    Randomize()
    For i = 0 To Size - 1
        Ind = Random(P)
        Pool.Add(Ind)
    Next
    If FBios = FitnessBios.BiggerFitness Then
        Winner = Pool.FindMax(F)
    End If
Public Shared Function CrowdTournament(ByVal P As Population) As Individual
Dim a, b As Individual
Randomize()
a = Random(P)
b = Random(P)
If a.Rank < b.Rank Then
    Return a
End If
If b.Rank < a.Rank Then
    Return b
End If
If a.CrowdingDistance > b.CrowdingDistance Then
    Return a
Else
    Return b
End If

End Function

Private Shared Function ScalePopulation(ByVal P As Population) As Population
Dim Sum As Double = P.Sum(Function(individual) individual.Fitness)
Dim D As Individual
For i = 0 To P.Count - 1
    D = P(i)
    D.Fitness = ((P(i).Fitness * P.Count) / Sum)
    P(i) = D
Next
Return P
End Function

Private Shared Function CreateParentPool(ByVal P As Population) As Population
Dim NumberOfCopies(P.Count - 1) As Short
Dim UpperMid As New Population
Dim LowerMid As New Population
Dim Ind As Individual
Dim Result As Population
Dim int As Integer
Dim D As Individual
UpperMid.ID = P.ID
For i = 0 To P.Count - 1
    D = P(i)
    int = Floor(P(i).Fitness)
    D.Fitness -= int
    P(i) = D
    For j = 0 To int - 1
        UpperMid.Add(P(i))
    Next
    If int = 0 Then
LowerMid.Add(P(i))
End If
Next
Result = UpperMid
For i = UpperMid.Count To P.Count / 2 - 1
    Ind = Tournament(LowerMid, 2, Function(individual)

    individual.Fitness)
    Result.Add(Ind)
Next
Return UpperMid
End Function

Enum FitnessBios
    BiggerFitness
    SmallerFitness
End Enum

End Class
End Namespace

A.1.2 Shell classes

A.1.2.1 WBGA

Imports GA_for_View_Selection.GeneticAlgorithm
Imports System.IO
Public Class WBGA
    Inherits GA

    Public Sub Run(ByVal Objective1 As Func(Of Object, Double), ByVal Objective2
As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
        Dim CurrentGeneration As New Population
        Dim NextGeneration As New Population
        Dim a, b As Individual

        CurrentGeneration.RandomGenerate(PopulationSize, ChromosomeSize + 7)
        For i = 0 To MaximumGeneration - 1
            For j = 0 To (PopulationSize / 2) - 1
                Evaluate(CurrentGeneration, 7, Objective1, Objective2,
                Constraint)

                a = Selection.Tournament(CurrentGeneration, 2,
                Function(individual) individual.Fitness, Selection.FitnessBios.SmallerFitness)
                b = Selection.Tournament(CurrentGeneration, 2,
                Function(individual) individual.Fitness, Selection.FitnessBios.SmallerFitness)

                Crossover.SinglePoint(a, b, a, b)
                Mutation.Random(a)
                Mutation.Random(b)

                NextGeneration.Add(a)
                NextGeneration.Add(b)
            Next
        CurrentGeneration = NextGeneration.ShallowClone()
        NextGeneration.Clear()
Next

Evaluate(CurrentGeneration, 7, Objective1, Objective2, Constraint)
CurrentGeneration.SaveToFile("C:\WBGA.txt")

End Sub

Private Sub Evaluate(ByVal P As Population, ByVal size As Short, ByVal Objective1 As Func(Of Object, Double), ByVal Objective2 As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
Dim a, b As Short
Dim w1, w2 As Double
Dim ind As Individual

For i = 0 To P.Count - 1
    a = GetWeightsIndex(P(i), 7)
    b = 127 - a
    w1 = (a / 127)
    w2 = 1 - w1

    ind = New Individual(P(0).Count - size)
    For j = size To P(0).Count - 1
        ind(j - size) = P(i)(j)
    Next
    ind.Evaluate(Objective1, Objective2, Constraint)
    P(i).Objective1Value = ind.Objective1Value
    P(i).Objective2Value = ind.Objective2Value
    P(i).Fitness = w1 * ind.Objective1Value + w2 * ind.Objective2Value
    P(i).Fitness = P(i).Fitness / NicheCount(P(i), P)

Next

End Sub

Private Function GetObjectiveValues(ByVal Ind As Individual, ByVal size As Short, ByRef Obj1 As Double, ByRef obj2 As Double, ByVal Objective1 As Func(Of Object, Double), ByVal Objective2 As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
Dim NewInd As New Individual(Ind.Count - size)

For i = size To Ind.Count - 1
    NewInd(i) = Ind(i)
Next

NewInd.Evaluate(Objective1, Objective2, Constraint)
Obj1 = NewInd.Objective1Value
obj2 = NewInd.Objective2Value

End Function

Private Function GetWeightsIndex(ByVal Ind1 As Individual, ByVal size As Short) As Short
Dim a As Short

For j = 0 To size - 1
    a += Ind1(size - 1 - j) * Math.Pow(2, j)
Next

Return a

End Function
Private Function NicheCount(ByVal Ind As Individual, ByVal P As Population) As Double
    Dim NC As Double = 0
    Dim SourceIndex, DestinationIndex As Short
    SourceIndex = GetWeightsIndex(Ind, 7)
    For i = 0 To P.Count - 1
        DestinationIndex = GetWeightsIndex(P(i), 7)
        NC += SharingFunction(Math.Abs(SourceIndex - DestinationIndex), 20)
    Next
    Return NC
End Function

Private Function SharingFunction(ByVal distance As Short, ByVal SigmaShare As Double) As Double
    If distance <= SigmaShare Then
        Return 1 - (distance / SigmaShare)
    Else
        Return 0
    End If
End Function

End Class

A.1.2.2 VEGA

Imports GA_for_View_Selection.GeneticAlgorithm
Public Class VEGA
    Inherits GA
    Public Sub Run(ByVal Objective1 As Func(Of Object, Double), ByVal Objective2 As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
        Dim CurrentGeneration As New Population
        Dim NextGeneration As New Population
        Dim MatingPool As New Population
        Dim P1, P2 As New Population
        Dim a = New Individual(ChromosomeSize)
        Dim b = New Individual(ChromosomeSize)
        CurrentGeneration.RandomGenerate(PopulationSize, ChromosomeSize)
        For i = 0 To MaximumGeneration - 1
            CurrentGeneration.Evaluate(Objective1, Objective2, Constraint)
            P1 = CurrentGeneration.Top()
            P2 = CurrentGeneration.Buttom()
            MatingPool.Clear()
            For j = 0 To (PopulationSize / 4)
                a = Selection.Tournament(P1, 2, Function(individual) individual.Objective1Value, Selection.FitnessBios.SmallerFitness)
                MatingPool.Add(a)
            Next
            For j = 0 To (PopulationSize / 4)
b = Selection.Tournament(P2, 2, Function(individual)
   individual.Objective2Value, Selection.FitnessBios.SmallerFitness)
MatingPool.Add(b)
Next

NextGeneration.Clear()
For j = 0 To PopulationSize / 2 - 1
   a = Selection.Random(MatingPool)
   b = Selection.Random(MatingPool)
   Crossover.SinglePoint(a, b, a, b)
   Mutation.Random(a)
   Mutation.Random(b)
   NextGeneration.Add(a)
   NextGeneration.Add(b)
Next

CurrentGeneration = NextGeneration.ShallowClone()

End Sub

End Class

A.1.2.3 NPGA

Imports GA_for_View_Selection.GeneticAlgorithm
Public Class NPGA
   Inherits GA
   Public Tdom As Short = 10
   Public Sub Run(ByVal Objective1 As Func(Of Object, Double), ByVal Objective2
      As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
      Dim CurrentGeneration As New Population
      Dim NextGeneration As New Population
      Dim Parent1, Parent2 As Individual

      CurrentGeneration.RandomGenerate(PopulationSize, ChromosomeSize)
      For i = 0 To MaximumGeneration - 1
         CurrentGeneration.Evaluate(Objective1, Objective2, Constraint)
      Next

      For j = 0 To PopulationSize / 2 - 1
         Parent1 = NPGA_Selection(CurrentGeneration, NextGeneration, Tdom)
         Parent2 = NPGA_Selection(CurrentGeneration, NextGeneration, Tdom)
         Crossover.SinglePoint(Parent1, Parent2, Parent1, Parent2)
         Mutation.Random(Parent1)
         Mutation.Random(Parent2)
      Next
   End Sub

End Class
Parent1.Evaluate(Objective1, Objective2, Constraint)
Parent2.Evaluate(Objective1, Objective2, Constraint)

NextGeneration.Add(Parent1)
NextGeneration.Add(Parent2)

Next

CurrentGeneration = NextGeneration.ShallowClone()
NextGeneration.Clear()

Next

CurrentGeneration.NonDominated.SaveToFile("C:\NPGA.txt")

End Sub

Private Function NPGA_Selection(ByVal P As Population, ByVal Q As Population, ByVal Tdom As Short) As Individual
    Dim a, b As Individual
    Dim nca, ncb As Short
    Dim Subpopulation As New Population
    Dim Temp As Population
    Dim SigmaShare As Double
    For i = 0 To ((Tdom * P.Count) / 100) - 1
        a = Selection.Random(P)
        Subpopulation.Add(a)
    Next
    a = Selection.Random(P)
    b = Selection.Random(P)
    If a.DominateAny(Subpopulation) = True Then
        If b.DominateAny(Subpopulation) = False Then
            Return a
        End If
    End If
    If a.DominateAny(Subpopulation) = False Then
        If b.DominateAny(Subpopulation) = True Then
            Return b
        End If
    End If
    If Q.Count < 2 Then
        Dim R As Double
        Randomize()
        R = Rnd()
        If R <= 0.5 Then
            Return a
        Else
            Return b
        End If
    End If
    Temp = Q.ShallowClone()
Temp.Add(a)
Temp.Add(b)

SigmaShare = Temp.SuggestSigmaShare()
nca = Temp.NicheCount(a, 1, SigmaShare)
ncb = Temp.NicheCount(b, 1, SigmaShare)

If nca < ncb Then
  Return a
Else
  Return b
End If

End Function

End Class

A.1.2.4 MOGA

Imports GA_For_View_Selection.GeneticAlgorithm
Public Class MOGA
  Inherits GA
  Public Sub Run(ByVal Objective1 As Func(Of Object, Double), ByVal Objective2 As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
    Dim CurrentGeneration As New Population
    Dim NextGeneration As New Population
    Dim a As New Individual(ChromosomeSize)
    Dim b As New Individual(ChromosomeSize)
    CurrentGeneration.RandomGenerate(PopulationSize, ChromosomeSize)
    For i = 0 To MaximumGeneration - 1
      CurrentGeneration.Evaluate(Objective1, Objective2, Constraint)
      MOGA_Fitness_Assignment(CurrentGeneration)
      For j = 0 To (PopulationSize / 2) - 1
        a = Selection.RouletteWheel(CurrentGeneration, Function(individual) individual.Fitness)
        b = Selection.RouletteWheel(CurrentGeneration, Function(individual) individual.Fitness)
        Crossover.SinglePoint(a, b, a, b)
        Mutation.Random(a)
        Mutation.Random(b)
        NextGeneration.Add(a)
        NextGeneration.Add(b)
        Next
      NextGeneration = NextGeneration.ShallowClone()
      NextGeneration.Clear()
      CurrentGeneration.Evaluate(Objective1, Objective2, Constraint)
      CurrentGeneration.NonDominated.SaveToFile("C:\MOGA.txt")
    Next
  End Sub
Private Sub MOGA_Fitness_Assignment(ByVal P As Population)
    Dim i As Short
    Dim Rank As PopulationSet
    Dim Sum1, Sum2 As Double
    Dim SigmaShare As Double
    Dim a As Individual
    Rank = P.Classify()
    For i = 0 To Rank.Count - 1
        Sum1 = 0
        For j = 0 To i - 1
            Sum1 = Sum1 + Rank(j).Count
        Next
        Sum2 = 0
        For j = 0 To Rank(i).Count - 1
            Rank(i)(j).Fitness = P.Count - Sum1 - 0.5 * (Rank(i).Count - 1)
            Rank(i)(j).Value1 = Rank(i)(j).Fitness / Rank(i).NicheCount(a, 1, SigmaShare)
            Sum2 = Sum2 + Rank(i)(j).Fitness
        Next
        For j = 0 To Rank(i).Count - 1
            Rank(i)(j).Fitness *= Rank(i)(j).Value1 * (Rank(i).Count / Sum2)
        Next
    Next
End Sub

End Class

A.1.2.5 SPEA

Imports GA_for_View_Selection.GeneticAlgorithm

Public Class SPEA
    Inherits GA
    Public ExternalSize As Short = 0.2 * PopulationSize

    Public Sub Run(ByVal Objective1 As Func(Of Object, Double), ByVal Objective2 As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
        Dim CurrentGeneration As New Population
        Dim NextGeneration As New Population
Dim CombinedGeneration As Population
Dim ExternalPopulation As New Population
Dim Clusterlist As PopulationSet
Dim Parent1, Parent2, Child1, Child2 As Individual
CurrentGeneration.RandomGenerate(PopulationSize, ChromosomeSize)
For i = 0 To MaximumGeneration - 1
    CurrentGeneration.Evaluate(Objective1, Objective2, Constraint)
    ExternalPopulation = ExternalPopulation.NonDominated
    If ExternalPopulation.Count > ExternalSize Then
        Clusterlist = ExternalPopulation.DoClustering(ExternalSize)
        ExternalPopulation = Clusterlist.ClustersRepresentative()
    End If
    Calculate_External_Population_Fitness(CurrentGeneration, ExternalPopulation)
    Calculate_Main_Population_Fitness(CurrentGeneration, ExternalPopulation)
For j = 0 To PopulationSize / 2 - 1
    CombinedGeneration = ExternalPopulation + CurrentGeneration
    Parent1 = Selection.Tournament(CombinedGeneration, 2, Function(Ind As Individual) Ind.Fitness, Selection.FitnessBios.SmallerFitness)
    Parent2 = Selection.Tournament(CombinedGeneration, 2, Function(Ind As Individual) Ind.Fitness, Selection.FitnessBios.SmallerFitness)
    Crossover.SinglePoint(Parent1, Parent2, Child1, Child2)
    Mutation.Random(Child1)
    Mutation.Random(Child2)
    NextGeneration.Add(Child1)
    NextGeneration.Add(Child2)
Next
CurrentGeneration = NextGeneration.ShallowClone
NextGeneration.Clear()
Next
CurrentGeneration.Evaluate(Objective1, Objective2, Constraint)
CurrentGeneration.NonDominated.SaveToFile("C:\SPEA.txt")
End Sub

Private Sub Calculate_External_Population_Fitness(ByVal Main As Population, ByVal External As Population)
For Each I As Individual In External
    I.Fitness = 0
    For Each J As Individual In Main
        If Dominate(I, J) Then
            I.Fitness = I.Fitness + (1 / (Main.Count + 1))
Private Sub Calculate_Main_Population_Fitness(ByVal Main As Population, ByVal External As Population)
    For Each I As Individual In Main
        I.Fitness = 1
    Next
    For Each J As Individual In External
        If Dominate(J, I) Then
            I.Fitness = I.Fitness + J.Fitness
        End If
    Next
    End Sub

A.1.2.6 SPEA-II

Imports GA_for_View_Selection.GeneticAlgorithm
Public Class SPEA_II
    Inherits GA
    Public ArchiveSize As Short = 0.2 * PopulationSize
    Private K As Short

    Public Sub Run(ByVal Objective1 As Func(Of Object, Double), ByVal Objective2 As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
        Dim CurrentGeneration As New Population
        Dim NextGeneration As New Population
        Dim CombinedPopulation As Population
        Dim CurrentArchive As New Population
        Dim NextArchive As New Population

        Dim Child1, Child2, Parent1, Parent2 As Individual
        K = Math.Pow((ArchiveSize + PopulationSize), 0.5)
        CurrentGeneration.RandomGenerate(PopulationSize, ChromosomeSize)
        For i = 0 To MaximumGeneration - 1
            CurrentGeneration.Evaluate(Objective1, Objective2, Constraint)
            CombinedPopulation = CurrentGeneration + CurrentArchive
            CalculateFitness(CombinedPopulation)
            NextArchive = NonDominated(CombinedPopulation)
            If i > 0 Then
                NextArchive = NonDominated(NextArchive)
            End If
            If NextArchive.Count > ArchiveSize Then
                NextArchive = Truncate(NextArchive)
            ElseIf NextArchive.Count < ArchiveSize Then
                FillUpFromDominated(CombinedPopulation, NextArchive)
            End If
            For j = 0 To PopulationSize / 2 - 1
            Next
        Next
    End Sub
Parent1 = Selection.Tournament(NextArchive, 2, Function(Ind As Individual) Ind.Fitness, Selection.FitnessBios.SmallerFitness)
Parent2 = Selection.Tournament(NextArchive, 2, Function(Ind As Individual) Ind.Fitness, Selection.FitnessBios.SmallerFitness)
Crossover.SinglePoint(Parent1, Parent2, Child1, Child2)
Mutation.Random(Child1)
Mutation.Random(Child2)
NextGeneration.Add(Child1)
NextGeneration.Add(Child2)

CurrentArchive = NextArchive.ShallowClone()
CurrentGeneration = NextGeneration.ShallowClone()
NextArchive.Clear()
NextGeneration.Clear()

CurrentArchive.Evaluate(Objective1, Objective2, Constraint)
CurrentArchive.SaveToFile("C:\SPEA2.txt")

Private Sub CalculateFitness(ByVal P As Population)
CalculateRAWFitness(P)
CalculateFULLFitness(P)
End Sub

Private Function CalculateDistances(ByVal P As Population) As Array
Dim Dlist(P.Count) As List(Of Double)
Dim D As Double
For i = 0 To P.Count - 1
For j = i + 1 To P.Count - 1
D = GA.Distance(P(i), P(j), DistanceCalculationType.Objectives)
If IsNothing(Dlist(i)) Then
Dlist(i) = New List(Of Double)
End If
Dlist(i).Add(D)
If IsNothing(Dlist(j)) Then
Dlist(j) = New List(Of Double)
End If
Dlist(j).Add(D)
Next
Next
For i = 0 To P.Count - 1
Dlist(i).Sort()
Next
Return Dlist
End Function

Private Sub CalculateRAWFitness(ByVal P As Population)
Dim Strength(P.Count) As Short
Dim Sum(P.Count) As Short
For i = 0 To P.Count - 1
For j = 0 To P.Count - 1
If GA.Dominate(P(i), P(j)) And i <> j Then
Strength(i) += 1
End If
Next
Next
End If
Next
Next

For i = 0 To P.Count - 1
For j = 0 To P.Count - 1
If GA.Dominate(P(j), P(i)) And i <> j Then
    P(i).Fitness += Strength(j)
End If
Next
Next

End Sub

Private Sub CalculateFULLFitness(ByVal P As Population)
    Dim D As Double
    Dim DList() As List(Of Double)
    DList = CalculateDistances(P)
    For i = 0 To P.Count - 1
        D = 1 / (DList(i)(K - 1) + 2)
        P(i).Fitness = P(i).Fitness + D
    Next
End Sub

Private Function NonDominated(ByVal P As Population) As Population
    Dim ND As New Population
    For i = 0 To P.Count - 1
        If P(i).Fitness < 1 Then
            ND.Add(P(i))
        End If
    Next
    Return ND
End Function

Private Function Truncate(ByVal P As Population) As Population
    Dim i As Integer
    Dim DistanceList As New List(Of PairDistance)
    Dim PD As PairDistance
    Dim MinDistance As Double = Double.MaxValue
    For i = 0 To P.Count - 1
        For j = i + 1 To P.Count - 1
            PD = New PairDistance
            PD.Source = i
            PD.Destination = j
            PD.Distance = Distance(P(i), P(j), DistanceCalculationType.Objectives)
            DistanceList.Add(PD)
        Next
    Next
    DistanceList.Sort(AddressOf PairDistance.Compare)
    i = 0
    While P.Count > ArchiveSize
        P.RemoveAt(DistanceList(i).Source)
        i += 1
    End While
    Return P
End Function

Private Structure PairDistance
    Dim Source As Short
    Dim Destination As Short
    Dim Distance As Double
End Structure
Public Shared Function Compare(ByVal ItemA As PairDistance, ByVal ItemB As PairDistance) As Integer
    Return ItemA.Distance < ItemB.Distance
End Function
End Structure

Private Function FillUpFromDominated(ByVal Combined As Population, ByVal Archive As Population)
    Dim i As Short = 0
    While i <= Combined.Count - 1 And Archive.Count < ArchiveSize
        If Combined(i).Fitness > 0 Then
            Archive.Add(Combined(i))
        End If
        i = i + 1
    End While
    Return Archive
End Function
End Class

A.1.2.7 NSGA

Imports GA_for_View_Selection.GeneticAlgorithm

Public Class NSGA
    Inherits GA_for_View_Selection.GeneticAlgorithm.GA

    Public Sub Run(ByVal Objective1 As Func(Of Object, Double), ByVal Objective2 As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
        Dim CurrentGeneration As New Population
        Dim NextGeneration As New Population
        Dim SigmaShare As Double
        Dim ClassifiedPopulation As PopulationSet
        Dim NC As Double
        Dim Fmin As Double
        Dim a As Individual
        Dim Parent1 As Individual
        Dim Parent2 As Individual
        Dim Child1 As Individual
        Dim Child2 As Individual

        CurrentGeneration.RandomGenerate(PopulationSize, ChromosomeSize)

        For i = 0 To MaximumGeneration - 1
            CurrentGeneration.Evaluate(Objective1, Objective2, Constraint)
            ClassifiedPopulation = CurrentGeneration.Classify()
            Fmin = PopulationSize + 0.01
            For t1 = 0 To ClassifiedPopulation.Count - 1
                SigmaShare = ClassifiedPopulation(t1).SuggestSigmaShare()
For \( t2 = 0 \) To \( \text{ClassifiedPopulation}(t1).\text{Count} - 1 \)
\[
\begin{align*}
a &= \text{ClassifiedPopulation}(t1)(t2) \\
a.\text{Fitness} &= F_{\text{min}} - 0.01 \\
NC &= \text{ClassifiedPopulation}(t1).\text{NicheCount}(a, 1, \text{SigmaShare}) \\
a.\text{Fitness} &= a.\text{Fitness} / NC
\end{align*}
\]
Next

\[ F_{\text{min}} = \text{ClassifiedPopulation}(t1).\text{FindMin}(\text{Function} \ (\text{ind As Individual}) \ \text{ind.\text{Fitness}}).\text{Fitness} \]
Next

\[
\text{CurrentGeneration} = \text{ClassifiedPopulation}.\text{Merge}()
\]
For \( j = 0 \) To \( (\text{PopulationSize} / 2) - 1 \)
\[
\begin{align*}
\text{Parent1} &= \text{Selection}.\text{Tournament}(\text{CurrentGeneration}, 2, \text{Function(individual) individual.\text{Fitness}, Selection.\text{FitnessBios}.\text{BiggerFitness}}) \\
\text{Parent2} &= \text{Selection}.\text{Tournament}(\text{CurrentGeneration}, 2, \text{Function(individual) individual.\text{Fitness}, Selection.\text{FitnessBios}.\text{BiggerFitness}})
\end{align*}
\]
\[
\text{Crossover}.\text{SinglePoint}(\text{Parent1, Parent2, Child1, Child2})
\]
\[
\text{Mutation}.\text{Random}(\text{Child1}) \\
\text{Mutation}.\text{Random}(\text{Child2})
\]
\[
\text{NextGeneration}.\text{Add}(\text{Child1}) \\
\text{NextGeneration}.\text{Add}(\text{Child2})
\]
Next
\[
\text{CurrentGeneration} = \text{NextGeneration}.\text{ShallowClone} \\
\text{NextGeneration}.\text{Clear}()
\]
Next
\[
\text{CurrentGeneration}.\text{Evaluate}(\text{Objective1, Objective2, Constraint})
\]
\[
\text{CurrentGeneration}.\text{Non-dominated}.\text{SaveToFile}("C:\NSGA.txt")
\]
End Sub

End Class

A.1.2.8 NSGA-II

Imports GA_for_View_Selection.GeneticAlgorithm
Public Class NSGA_II
    Inherits GA
    Public Sub Run(ByVal Objective1 As Func(Of Object, Double), ByVal Objective2 As Func(Of Object, Double), ByVal Constraint As Func(Of Object, Double))
        Dim Parents As New Population
        Dim Childs As New Population
        Dim CombinedPopulation As Population
        Dim ClassifiedPopulation As PopulationSet
        Dim i, j, index, t As Short
        Parents.RandomGenerate(PopulationSize, ChromosomeSize)
        For \( i = 0 \) To \( \text{MaximumGeneration} - 1 \)
Parents.Evaluate(Objective1, Objective2, Constraint)
Parents.AssignCrowdingDistance()
Childs = ProduceChilds(Parents)
Childs.Evaluate(Objective1, Objective2, Constraint)
CombinedPopulation = Parents + Childs
Parents.Clear()
ClassifiedPopulation = CombinedPopulation.Classify
index = 0
    Parents = Parents + ClassifiedPopulation(index)
    index = index + 1
End While
ClassifiedPopulation(index).AssignCrowdingDistance()
ClassifiedPopulation(index).Sort(Function(individual) => individual.CrowdingDistance)
t = 0
While Parents.Count < PopulationSize
    Parents.Add(ClassifiedPopulation(index).Member(t))
    t = t + 1
End While
Next
Parents.NonDominated().SaveToFile("C:\NSGA2.txt")
End Sub
Public Function ProduceChilds(ByVal P As Population) As Population
    Dim Childs As New Population
    Dim Parent1, Parent2, Child1, Child2 As Individual
    Dim Classified As PopulationSet
    Classified = P.Classify()
    P = Classified.Merge()
    For i = 0 To PopulationSize / 2 - 1
        Parent1 = Selection.CrowdedTournament(P)
        Parent2 = Selection.CrowdedTournament(P)
        Crossover.SinglePoint(Parent1, Parent2, Child1, Child2)
        Mutation.Random(Child1)
        Mutation.Random(Child2)
        Childs.Add(Child1)
        Childs.Add(Child2)
    Next
    Return Childs
End Function
End Class
A.2 Classes in Problem Domain

A.2.1 View

Namespace ViewSelection
    Public Class View
        Public Size As Double
        Public HierarchyNodes As List(Of HierarchyNode)
        Public HierarchyLevels As String
        Public IsTopView As Boolean
        Public QueryFrequency As Double
        Public UpdateFrequency As Double
        Public Id As Integer
        Public Function MaximumSize() As Integer
            Dim Product As Integer = 1
            For i = 0 To HierarchyNodes.Count - 1
                Product *= HierarchyNodes(i).Cardinality
            Next
            Return Product
        End Function
    End Class
End Namespace

A.2.2 Lattice

Imports System.IO

Public Class Lattice
    Public AdjacencyMatrix(100, 100) As Boolean
    Private ItemList As New List(Of Object)
    Public Property Connections() As DataTable
        Get
            Dim dt As New DataTable
            Dim Dc As DataColumn
            Dim Dr As DataRow
            For i = 0 To Count - 1
                Dc = New DataColumn
                Dc.DataType = GetType(Boolean)
                dt.Columns.Add(Dc)
                Dr = dt.NewRow()
                dt.Rows.Add(Dr)
            Next
            For i = 0 To Count - 1
                For j = 0 To Count - 1
                    dt.Rows(i).Item(j) = Edge(i, j)
                Next
            End For
            Return dt
        End Get
    End Property
End Class
Public Sub Add(ByVal Item As Object)
    If ItemList.Contains(Item) = False Then 
        ItemList.Add(Item)
    End If
End Sub
End If

End Sub

Public Function Indexof(ByVal O As Object) As Integer
    Return ItemList.IndexOf(O)
End Function

Public Sub Clear()
    For i = 0 To ItemList.Count - 1
        For j = 0 To ItemList.Count - 1
            AdjacencyMatrix(i, j) = False
        Next
    Next
    ItemList.Clear()
End Sub

Public Property Edge(ByVal index1 As Integer, ByVal index2 As Integer)
    Get
        Return AdjacencyMatrix(index1, index2)
    End Get
    Set(ByVal value)
        AdjacencyMatrix(index1, index2) = value
        AdjacencyMatrix(index2, index1) = value
    End Set
End Property

Public Function ParentsOf(ByVal Item As Object) As List(Of Object)
    Dim j As Integer
    Dim ParentsList As New List(Of Object)
    Dim Index As Short = ItemList.IndexOf(Item)
    For j = 0 To Index - 1
        If AdjacencyMatrix(j, Index) = True Then
            ParentsList.Add(ItemList(j))
        End If
    Next
    Return ParentsList
End Function

Public Function AncestorsOf(ByVal Item As Object) As List(Of Object)
    Dim AncestorsList = New List(Of Object)
    Dim ParentsList As New List(Of Object)
    Dim Q As New Queue(Of Object)
    Dim Item2 As Object = Item
    Q.Enqueue(Item2)
    While Q.Count > 0
        Item2 = Q.Dequeue()
        Index = ItemList.IndexOf(Item2)
        For i = 0 To Index - 1
If AdjacencyMatrix(i, Index) = True And AncestorsList.Contains(ItemList(i)) = False Then
    AncestorsList.Add(ItemList(i))
    Q.Enqueue(ItemList(i))
End If
Next
End While
Return AncestorsList

End Function
Public Function IsAncestorOf(ByVal i As Integer, ByVal j As Integer)
    Dim A As List(Of Object)
    A = AncestorsOf(ItemList(j))
    If A.Contains(ItemList(i)) Then
        Return True
    Else
        Return False
    End If
End Function
Public Function ChildsItems(ByVal O As Object) As List(Of Object)
    Dim ChildList As New List(Of Object)
    Dim n As Integer
    n = IndexOf(O)
    For i = n + 1 To Count
        If AdjacencyMatrix(i, n) = True Then
            ChildList.Add(ItemList(i))
        End If
    Next
    Return ChildList
End Function
Public Function ChildsIndexes(ByVal n As Integer) As List(Of Short)
    Dim ChildList As New List(Of Short)
    For i = n + 1 To Count
        If AdjacencyMatrix(i, n) = True Then
            ChildList.Add(i)
        End If
    Next
    Return ChildList
End Function
Public Sub DrawLattice(ByRef GBox As GroupBox)
    Dim visited(Count) As Boolean
    Dim positions(Count) As Point
    Dim Level(Count) As Queue
    Dim q As New Queue
    Dim g As System.Drawing.Graphics
    Dim p As New Pen(Color.Black, 2)
    Dim drawFont As New Font("Arial", 9)
    Dim drawBrush As New SolidBrush(Color.Black)
    Dim drawFormat As New StringFormat()
    If ItemList.Count > 0 Then
        g = GBox.CreateGraphics()
        g.Clear(GBox.BackColor)
        positions = DeterminesPositions(GBox.Height, GBox.Width)
        DrawAllCircles(positions, g, Pens.Black, 12)
        DrawLines(positions, g, p)
        DrawCircleNumbers(positions, g, drawFont)
    Else
        GBox.Refresh()
Private Sub DrawACircle(ByRef g As Graphics, ByRef center As Point, ByVal radius As Integer)
    Dim rect As New Rectangle(center.X - radius, center.Y - radius, 2 * radius, 2 * radius)
    g.FillEllipse(Brushes.Black, rect)
End Sub

Private Function DeterminesPositions(ByVal hight As Short, ByVal width As Short) As Array
    Dim Positions(Count - 1) As Point
    Dim i, t, s As Short
    Dim Q As New Queue
    Dim L As List(Of Short)
    Dim Li As New List(Of Short)
    Dim n As New node
    Dim m As New node
    Dim sum(Count - 1) As Short
    Dim MaxLevel As Short
    Dim NodeLevel(Count - 1) As Short
    Dim Level(Count - 1, Count - 1) As Boolean
    Dim nodeorder(Count - 1) As Short
    n.Id = 0
    n.Level = 0
    Q.Enqueue(n)
    MaxLevel = 0
    While (Q.Count > 0)
        n = Q.Dequeue()
        sum(n.Level) += 1
        NodeLevel(n.Id) = n.Level
        nodeorder(n.Id) = sum(n.Level)
        Level(n.Level, n.Id) = True
        If (MaxLevel < n.Level) Then
            MaxLevel = n.Level
        End If
        L = ChildsIndexes(n.Id)
        For i = 0 To L.Count - 1
            m.Id = L(i)
            m.Level = n.Level + 1
            If Q.Contains(m) = False Then
                Q.Enqueue(m)
            End If
        Next
    End While
    For i = 0 To Count - 1
        t = NodeLevel(i)
        s = nodeorder(i)
    End If
End Sub
Positions(i).X = (width / (sum(t) + 1)) * (s)
Positions(i).Y = (hight / (MaxLevel + 2)) * t + 100
Next

Return Positions
End Function

Private Sub DrawAllCircles(ByVal positions() As Point, ByVal g As System.Drawing.Graphics, ByVal p As Pen, ByVal radius As Short)
    For i = 0 To positions.Length - 1
        DrawACircle(g, positions(i), radius)
    Next
End Sub

Private Sub DrawLines(ByVal positions() As Point, ByVal g As System.Drawing.Graphics, ByVal p As Pen)
    Dim i As Short = 0
    For i = 0 To Count - 1
        For j = i + 1 To Count - 1
            If AdjacencyMatrix(j, i) = True Then
                g.DrawLine(p, positions(i), positions(j))
            End If
        Next
    Next
End Sub

Private Sub DrawCircleNumbers(ByVal positions() As Point, ByVal g As System.Drawing.Graphics, ByVal drawFont As Font)
    For i = 0 To positions.Length - 1
        g.DrawString(i.ToString, drawFont, Brushes.White, positions(i).X - 8, positions(i).Y - 8)
    Next
End Sub

Private Structure node
    Public Id As Integer
    Public Level As Integer
End Structure
End Class

A.2.3 VSP

Imports GA_for_View_Selection.General
Imports GA_for_View_Selection.ViewSelection
Imports System.Math
Namespace ViewSelection
    <Serializable()> Public Class VSP
        Public DiskSpaceLimitValue As Double
    End Class
End Namespace
Private MinQ As Double
Private MaxQ As Double
Private MinU As Double
Private MaxU As Double

Private _TheLattice As Lattice
Private AncestorList() As List(Of View)

Public Sub New(ByVal ViewLattice As Lattice)
    _TheLattice = ViewLattice
    ReDim AncestorList(_TheLattice.Count)
    For i = 0 To _TheLattice.Count - 1
        AncestorList(i) = AncestorsOf(i)
    Next
    MinQ = q(All)
    MaxQ = q(Null)
    MinU = U(Null)
    MaxU = U(All)
End Sub

Public Property Thelattice() As Lattice
    Get
        Return _TheLattice
    End Get
    Set(ByVal value As Lattice)
        _TheLattice = value
    End Set
End Property

Public ReadOnly Property View(ByVal i As Integer) As View
    Get
        Return Thelattice.Item(i)
    End Get
End Property

Public ReadOnly Property NumberOfViews() As Integer
    Get
        Return Thelattice.Count
    End Get
End Property

Public Function CubeSize() As Double
    Return Space(All)
End Function

Public Function U(ByVal M As VSPPPhenotype) As Double
    Dim i As Integer
    Dim Sum As Double = 0
    For i = 1 To NumberOfViews - 1
If M(i) = 1 Then
    Sum = Sum + View(i).UpdateFrequency * u(View(i), M)
End If

Next

U = Sum

End Function

Public Function NormalizedU(ByVal M As VSPPhenotype) As Double
    Dim i As Integer
    Dim Sum As Double = 0
    Dim NU As Double
    For i = 1 To NumberOfViews - 1
        If M(i) = 1 Then
            Sum = Sum + View(i).UpdateFrequency * u(View(i), M)
        End If
    Next
    NU = (Sum - MinU) / (MaxU - MinU)
    NU = NU * 100
    Return NU
End Function

Public Function Space(ByVal M As VSPPhenotype) As Double
    Dim i As Integer
    Dim Sum As Double
    For i = 1 To NumberOfViews - 1
        If M(i) = 1 Then
            Sum = Sum + Thelattice(i).Size
        End If
    Next
    Space = Sum
End Function

Public Function DiskSpaceConstraint(ByVal M As VSPPhenotype) As Double
    Dim C As Double
    C = Space(M) - DiskSpaceLimitValue
    Return C
End Function

Public Function q(ByVal v As View, ByVal M As VSPPhenotype) As Double
    Dim LCMV As View
    If M(v.Id) = 1 Or v.Id = 0 Then
        LCMV = v
    Else
        LCMV = v
    End If

LCMV = LeastCostMaterializedAncestor(v, M)
End If

Return LCMV.Size

End Function

Public Function Q(ByVal M As VSPPhenotype) As Double
Dim i As Integer
Dim sum As Double

For i = 0 To NumberOfViews - 1
    sum += View(i).QueryFrequency * Q(View(i), M)
Next
Q = sum

End Function

Public Function NormalizedQ(ByVal M As VSPPhenotype) As Double
Dim i As Integer
Dim sum As Double
Dim NQ As Double

For i = 0 To NumberOfViews - 1
    sum += View(i).QueryFrequency * q(View(i), M)
Next
NQ = (sum - MinQ) / (MaxQ - MinQ)
NQ = NQ * 100
Return NQ

End Function

Public Function u(ByVal v As View, ByVal M As VSPPhenotype) As Double
Dim Min As Double = Double.MaxValue
Dim SmallestAncestor As Integer
Dim index As Integer
Dim Size As Double
Dim TheAncestors As List(Of View)

index = Thelattice.IndexOf(v)
TheAncestors = AncestorList(index)
M(0) = 1

For i = 0 To M.Count - 1
    If M(i) = 1 And i <> index And TheAncestors.Contains(View(i)) = True Then
        Size = View(i).Size
        If Size < Min Then
            Min = Size
            SmallestAncestor = i
        End If
    End If
Next

End Function

Public Function AncestorsOf(ByVal i As Short) As List(Of View)
Dim A As New List(Of View)
For n = 0 To i - 1
    If ISAncestor(View(n), View(i)) = True Then
        A.Add(View(n))
    End If
Next
Return A
End Function

Private Function AncestorsOf(ByVal v As View) As List(Of View)
    Return AncestorsOf(TheLattice.Indexof(v))
End Function

Private Function ISAncestor(ByVal V1 As View, ByVal V2 As View)
    If V1.HierarchyLevels = V2.HierarchyLevels Then
        Return False
    End If
    For i = 0 To V1.HierarchyLevels.Count - 1
        If V2.HierarchyLevels(i) < V1.HierarchyLevels(i) Then
            Return False
        End If
    Next
    Return True
End Function

Private Function All() As VSPPhenotype
    Dim A As New VSPPhenotype(NumberOfViews)
    For i = 0 To NumberOfViews - 1
        A(i) = 1
    Next
    Return A
End Function

Private Function Null() As VSPPhenotype
    Dim A As New VSPPhenotype(NumberOfViews)
    For i = 0 To NumberOfViews - 1
        A(i) = 0
    Next
    Return A
End Function

Public Function LeastCostMaterializedAncestor(ByVal v As View, ByVal M As VSPPhenotype) As View
    Dim MaterializedAncestorsList As List(Of View)
    Dim Minimum As View
    Dim n As Short
    MaterializedAncestorsList = MaterializedAncestors(v, M)
    If MaterializedAncestorsList.Count = 0 Then
        Return TheLattice.TopNode()
    End If
    Minimum = MaterializedAncestorsList(0)
    n = MaterializedAncestorsList.Count
    For i = 1 To n - 1
        If Minimum.Size > MaterializedAncestorsList(i).Size Then
            Minimum = MaterializedAncestorsList(i)
        End If
Private Function MaterializedAncestors(ByVal V As View, ByVal M As VSPPhenotype) As List(Of View)
    Dim MaterializedAncestorsList As New List(Of View)
    Dim AncestorsList As New List(Of View)
    Dim c As View
    Dim n As Short
    c = Thelattice.TopNode()
    AncestorsList = AncestorList(V.Id)
    n = AncestorsList.Count
    For Each MyView In AncestorsList
        If M(MyView.Id) = 1 Or MyView.Id = 0 Then
            MaterializedAncestorsList.Add(MyView)
        End If
    Next
    Return MaterializedAncestorsList
End Function

Public Function SearchSpaceSize() As Long
    If NumberOfViews > CInt(Log(Long.MaxValue, 2)) Then
        Return Long.MaxValue
    Else
        Return Pow(2, NumberOfViews)
    End If
End Function
End Class

End Namespace

A.2.4 VSP Phenotype

Namespace General
    Public Class VSPPhenotype
        Private _array() As Integer
        Public F1, F2 As Double

        Public Property List() As Integer()
            Get
                Return _array
            End Get
            Set(ByVal value() As Integer)
                _array = value.ToArray()
            End Set
        End Property
    End Class
End Namespace
Public Sub New(ByVal Size As Short)
    ReDim _array(Size - 1)
    _array(0) = 1
End Sub
Public Sub New(ByVal o1 As Double, ByVal o2 As Double)
    F1 = o1
    F2 = o2
End Sub

Default Public Property A(ByVal i As Short) As Integer
    Get
        If i < _array.Count Then
            Return _array(i)
        Else
            Return Nothing
        End If
    End Get
    Set(ByVal value As Integer)
        _array(i) = value
    End Set
End Property

Public ReadOnly Property SearchSpaceSize() As Double
    Get
        Return Math.Pow(2, Count)
    End Get
End Property
Public ReadOnly Property Count()
    Get
        Return _array.Count
    End Get
End Property

Public Function Clone() As VSPPhenotype
    Dim S2 As New VSPPhenotype(Count)
    S2._array = _array.Clone()
    S2.F1 = F1
    S2.F2 = F2
    Return S2
End Function

Public Sub Clear()
    For i = 0 To Count - 1
        _array(i) = 0
    Next
End Sub

Public Overrides Function ToString() As String
    Return _array.ToString()
End Function

End Class
End Namespace
A.3 Performance metrics

'Imports GA_for_View_Selection.General
Imports System.IO
Imports System.Math
Imports Microsoft.Office.Interop

Public Class PerformanceMetric
    Private Approximation1 As New List(Of ObjectiveSpacePoint)
    Private Approximation2 As New List(Of ObjectiveSpacePoint)

    Private Filename As String

    Private Function Coverage(ByVal Approx1 As List(Of ObjectiveSpacePoint), ByVal Approx2 As List(Of ObjectiveSpacePoint))
        Dim A, B As Short
        Dim C As Double
        A = 0
        B = Approx2.Count
        For Each y As ObjectiveSpacePoint In Approx2
            If Dominate(Approx1, y) = True Then
                A += 1
            End If
        Next
        C = A / B
        Return C
    End Function

    Private Function Dominate(ByVal P1 As ObjectiveSpacePoint, ByVal P2 As ObjectiveSpacePoint) As Boolean
        If P1.X > P2.X Then
            Return False
        End If
        If P1.Y > P2.Y Then
            Return False
        End If
        If P1.Y < P2.Y Then
            Return True
        End If
        Return True
    End Function

    Private Function Dominate(ByVal Approximation As List(Of ObjectiveSpacePoint), ByVal P As ObjectiveSpacePoint) As Boolean
        For Each D As ObjectiveSpacePoint In Approximation
            If Dominate(D, P) = True Then
                Return True
            End If
        Next
        Return False
    End Function

Private Function Dominate(ByVal Approximation As List(Of ObjectiveSpacePoint), ByVal P As ObjectiveSpacePoint) As Boolean
    For Each D As ObjectiveSpacePoint In Approximation
        If Dominate(D, P) = True Then
            Return True
        End If
    Next
    Return False
End Function
Private Function DominatedPoints(ByVal Approximation As List(Of ObjectiveSpacePoint)) As List(Of ObjectiveSpacePoint)
    Dim Dominated As New List(Of ObjectiveSpacePoint)
    For i = 0 To Approximation.Count - 1
        For j = i + 1 To Approximation.Count - 1
            If Dominate(Approximation(i), Approximation(j)) = True And Dominated.Contains(Approximation(j)) = False Then
                Dominated.Add(Approximation(j))
            End If
            If Dominate(Approximation(j), Approximation(i)) = True And Dominated.Contains(Approximation(i)) = False Then
                Dominated.Add(Approximation(i))
            End If
        Next
    Next
    Return Dominated
End Function

Private Function HyperVolume(ByVal Approx As List(Of ObjectiveSpacePoint), ByVal ReferencePoint As ObjectiveSpacePoint) As Double
    Dim Volume As Double = 0
    Dim Rectangular As Double = 0
    Dim width, height As Double
    width = 0
    Height = 0
    Approx.Sort(AddressOf Xcompare)
    For i = Approx.Count - 1 To 0 Step -1
        If i = Approx.Count - 1 Then
            width = ReferencePoint.X - Approx(i).X
        Else
            width = Approx(i + 1).X - Approx(i).X
        End If
        height = ReferencePoint.Y - Approx(i).Y
        Rectangular = width * height
        Volume = Volume + Rectangular
    Next
    Return Volume
End Function

Private Function MaximumSpread(ByVal Approx1 As List(Of ObjectiveSpacePoint))
    Dim A, B As Double
    Dim MS As Double
    A = Approx1.Max(Function(objectivespacepoint) objectivespacepoint.X) - (Approx1.Min(Function(objectivespacepoint) objectivespacepoint.X))
    B = Approx1.Max(Function(objectivespacepoint) objectivespacepoint.Y) - (Approx1.Min(Function(objectivespacepoint) objectivespacepoint.Y))
    A = Pow(A, 2)
    B = Pow(B, 2)
    MS = Pow(A + B, 0.5)
    Return MS
Private Function Mean(ByVal approximation As List(Of ObjectiveSpacePoint)) As Double
    Dim Sum As Double
    Dim MeanValue As Double
    For Each P As ObjectiveSpacePoint In approximation
        Sum += MinDistance(P, approximation)
    Next
    MeanValue = Sum / approximation.Count
    Return MeanValue
End Function

Private Function MinDistance(ByVal P As ObjectiveSpacePoint, ByVal Approximation As List(Of ObjectiveSpacePoint)) As Double
    Dim Sum As Double
    Dim MinSum As Double = Double.MinValue
    Dim MinPoint As New ObjectiveSpacePoint(0, 0)
    For Each D As ObjectiveSpacePoint In Approximation
        If (D.X <> P.X Or D.Y <> P.Y) Then
            Sum = Abs(P.Y - D.Y) + Abs(P.X - D.X)
            If Sum > MinSum Then
                MinSum = Sum
                MinPoint = D
            End If
        End If
    Next
    Return MinSum
End Function

Private Function NonDominated(ByVal Approximation As List(Of ObjectiveSpacePoint)) As List(Of ObjectiveSpacePoint)
    Dim flag As Boolean
    Dim ND As New List(Of ObjectiveSpacePoint)
    For i = 0 To Approximation.Count - 1
        flag = False
        For j = 0 To Approximation.Count - 1
            If Dominate(Approximation(j), Approximation(i)) = True And i <> j Then
                flag = True
                Exit For
            End If
        Next
        If flag = False Then
            ND.Add(Approximation(i))
        End If
    Next
    Return ND
End Function

Private Function OpenFile() As Boolean
    OpenFileDialog1.DefaultExt = "Input Approximation"
    If OpenFileDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
        ' Open file logic here
    End If
End Function
Return True
Else
    Return False
End If

End Function

Private Function ReadData() As List(Of ObjectiveSpacePoint)
    Dim R As StreamReader = New StreamReader(OpenFileDialog1.FileName)
    Dim Line As String
    Dim x, y As String
    Dim Point As ObjectiveSpacePoint
    Dim i, j As Short
    Dim Approximation As New List(Of ObjectiveSpacePoint)

    Line = R.ReadLine()
    While R.EndOfStream = False
        Point = New ObjectiveSpacePoint
        Line = R.ReadLine()
        x = ""
        y = ""
        If Line = "]" Then
            Exit While
        End If
        i = 0
        While Line(i) <> " "
            x = x + Line(i)
            i = i + 1
        End While
        For j = i + 1 To Line.Length - 1
            y = y + Line(j)
        Next
        Point.X = CDbl(x)
        Point.Y = CDbl(y)
        Approximation.Add(Point)
    End While
    R.Close()
    Return Approximation
End Function

Private Function Spacing(ByVal Approximation As List(Of ObjectiveSpacePoint))
    Dim Sum As Double
    Dim MeanValue As Double
    Dim S As Double
    MeanValue = Mean(Approximation)
    For Each P As ObjectiveSpacePoint In Approximation
        Sum = Pow((MinDistance(P, Approximation) - MeanValue), 2)
    Next
    S = Sum / (Approximation.Count - 1)
    S = Pow(S, 0.5)
Private Function Xcompare(ByVal A As ObjectiveSpacePoint, ByVal B As ObjectiveSpacePoint) As Integer
    If A.X > B.X Then
        Return 1
    End If

    If A.X = B.X Then
        Return 0
    End If

    If A.X < B.X Then
        Return -1
    End If
End Function

End Class
Appendix B. Box Plots

Figure B.1 Box Plot Showing Two Set Coverage, C(A,B). Algorithm A Refers to Algorithm In row And Algorithm B Refers to Algorithm In Column.
Figure B.2 Box Plot for Metric of hypervolume
Figure B.3 (continued) Box Plot for Metric of hypervolume
Figure B.4 Box Plot for Metric of Spacing
Figure B.5(Continued) Box Plot for Metric of Spacing
Figure B.6 Box Plot for Metric of Maximum Spread
Figure B.7 (Continued) Box Plot for Metric of Maximum Spread
Appendix C. Publications


