DOCTORAATSPROEFSCHRIFT

2009 | Faculteit Wetenschappen

A Performance Analysis of Multi-Objective Evolutionary Algorithms for Optimization

Proefschrift voorgelegd tot het behalen van de graad van
Doctor in Wetenschappen, te verdedigen door:

Jose Maria PANGILINAN

Promotor: prof. dr. Gerrit K. Janssens
Copromotor: prof. dr. Koen Vanhoof

D/2009/2451/48

universiteit hasselt

Acknowledgement

My gratitude and appreciation go to my promotor, Prof. dr. Gerrit Janssens for his guidance throughout my doctoral program, his invaluable contribution to this research, and his persistence in reminding me to write my dissertation with clarity and correctness.

I wish to thank Prof. dr. Koen Vanhoof for his willingness to act as my co-promotor, and the evaluation committee, Prof. dr. Frank Neven, Prof. dr. Jan Van Den Bussche, Prof. dr. ir. Tom Dhaene, Prof. dr. Marc Pirlot, and Prof. dr. ir. Patrick Beullens for their time and thoughtful criticism.

I wish to thank the Vlaamse Interuniversitaire Raad (VLIR) through Prof. dr. Joseph Leunis and Prof. dr. ir. Dirk De Waele for funding my scholarship, Saint Louis University and Fr. Paul Van Parijs for giving me the opportunity to pursue a doctoral degree, and Hasselt University for granting my scholarship.

I wish to thank Ms. Martine Dekonink and the International Office staff, Ms. Sarah Verlackt, Ms. Erika Vandersmissen and Mr. Marc Thoelen for assisting me with my visa application and accommodation every year, and the Finance Office and the Secretariat for assisting me with the financial and administrative tasks necessary during my doctoral program.

Finally, I would like to thank my family, Teresa and Michael for their support during the years of my study.

Jose Maria Pangilinan
Diepenbeek, September 2009

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Many real-world optimization problems are multiobjective by nature and the objectives are in conflict with each other. Mathematical techniques are available to find best-compromise solutions by aggregating multiple objectives into a single function. They have their drawbacks as they have difficulty dealing with concave and discontinuous Pareto fronts. Stochastic local search algorithms (also called metaheuristics) are often used when exact or mathematical methods are infeasible to employ and are used to solve difficult optimization problems. These algorithms provide approximations to quality solutions based on a randomized search process using knowledge from a neighborhood of solutions. The more often used metaheuristics are Simulated Annealing, Tabu Search, Ant Colony Optimization, and Evolutionary Algorithms. Evolutionary Algorithms (EA) seem particularly suitable to solve multiobjective optimization problems. There is a vast collection of published work on multiobjective evolutionary algorithms (MOEA) and their applications. An increasing number of research papers report comparative findings of several evolutionary algorithms in terms of computing speed and Pareto optimality as tested on various multiobjective problem instances or applications with known Pareto sets. However, in many cases insufficient explanation is given as to why and how an MOEA succeeds or fails in terms of the performance of its genetic operators.

1.2 Purpose and Significance of the Study

The scientific objective of the dissertation is to improve the understanding of how multiobjective evolutionary algorithms work in finding efficient solutions to multiobjective optimization problems through experimental research. The objective of the study is twofold: (1) to describe the performance of evolutionary algorithms in terms of *stability*, *computational complexity*, *diversity* and *optimality* of solutions in different multiobjective optimization problems (MOOP), and (2) to describe their strengths and weaknesses in each of the MOOP considered in the study and describe how the MOEA succeeded or failed in terms of the effect of its genetic operators.

The *stability* of an algorithm is concerned with the sensitivity of the results to changes in the MOEA parameters settings. *Computational complexity* refers to the solution run-time complexity in terms of the size of the problem. *Diversity* measures the spread of solutions in the nondominated set in order to provide the decision maker a true picture of trade-off solutions. *Optimality* measures the proximity of the best nondominated set to the Pareto-optimal set.

This study is a significant undertaking in promoting the use of evolutionary algorithms in multiobjective optimization and will be beneficial to other researchers and practitioners when they use evolutionary algorithms in finding solutions to different problems related to multiobjective optimization. This research provides proposals on how to evaluate the performance of an MOEA in terms of its sensitivity to genetic operators and the "goodness" of its solutions in terms of diversity and optimality. The strengths and weaknesses of the genetic operators of the EAs in multiobjective optimization are determined, which is helpful to other researchers and practitioners of EA and multiobjective optimization in developing new MOEAs. In addition, the sensitivity analyses of the genetic operators are clearly described, which is significant in setting proper parameter values for the genetic operators to drive the search for nondominated solutions more efficiently.

1.3 Methodology

The performance of MOEAs is investigated on several test problems and in some cases on problems with precise solutions and known Pareto-optimal sets. Such

test problems are used to compare MOEA results with known Pareto-optimal solutions and obtain more information on the behavior of MOEAs in terms of their proximity to the Pareto-optimal set and the diversity of their efficient solutions. In the absence of known Pareto-optimal-solutions, performance measures are utilized to describe and compare nondominated sets. Similarly, a selection of real-world case studies in multiobjective optimization is investigated. Testing MOEAs on application problems is necessary to demonstrate the performance and usefulness of MOEAs in practice. In order to study the performance of an MOEA, it needs to be tested on different optimization problems that require different search spaces, data types, structure of solutions, cardinality of solution sets, and distribution of solutions among others (see Figure 1.1).

The research method is for the greater part experimental in nature. The experimental activity is structured along four multiobjective optimization problems that are classified according to data structure, data representation, and search space. The problem sets are specifically selected to evaluate the performance and usefulness of MOEAs in different settings. Results are compared, when possible, with other standard or other EA algorithms over a variety of instances. Sensitivity of results to changes in parameter setting is tested and the type of test problems on which the MOEA fails or succeeds is investigated.

The *first* problem is a bi-objective problem that has a continuous (real-parameter) search space. A solution is represented as a fixed-length string of real-valued decision variables. The cardinality of the set of efficient solutions is known to be fixed. The application under study for this problem class is the *competitive facility location problem*.

The *second* problem is a combinatorial optimization problem that has a discrete search space. A solution is coded as a fixed-length string of discrete decision variables and the cardinality of the search space is exponential. The application for this problem class is the *container storage location problem*.

The *third* problem is an NP-complete problem and has a discrete search space. A solution is represented as a variable-length string of discrete decision variables. The cardinality of the solution set is exponential to the number of vertices in the network. The application under study for this problem class is the *shortest path problem*.

The *fourth* problem is an NP-hard problem and has a continuous search space. A solution is a tree structure with each node representing a string of continuous decision variables. The cardinality of the set of efficient solutions is exponential to the size of the tree. The application for this problem class is applied on *decision trees* or *classification trees*.

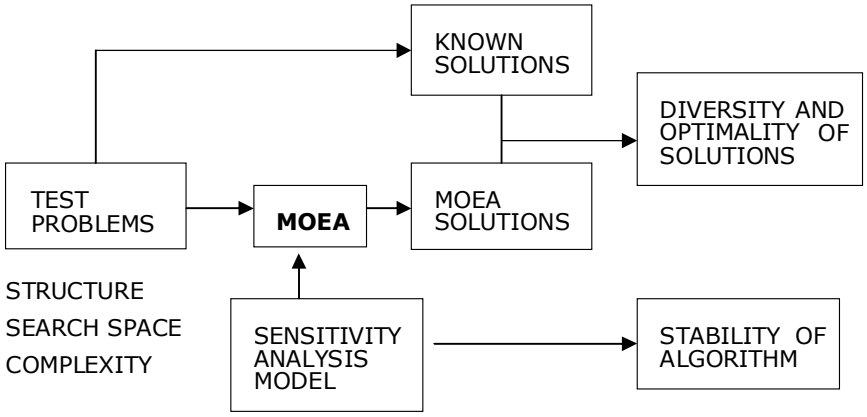

Figure 1.1 Conceptual Framework

Chapter 2 presents an introduction and literature review of multiobjective optimization methods, evolutionary algorithms, and multiobjective evolutionary algorithms. Chapters 3, 4, 5, and 6 discuss the application of MOEAs on different test problems. The literature review, the design of the genetic operators, the experiments, and the sensitivity analysis are discussed in each application chapter. Chapter 7 presents a multicriteria performance analysis for the solution sets generated in the application chapters. Finally, Chapter 8 presents the conclusions and future research directions.

Chapter 2

Literature Review on Multiobjective Optimization and Evolutionary Algorithms

A number of books have been published over the years that present and analyze multiobjective optimization methods. Ehrgott (2005) provides the necessary mathematical foundation of multicriteria optimization to solve nonlinear, linear and combinatorial problems with multiple criteria. Miettinen (1998) provides an extensive survey and review of literature of the theories and methods in nonlinear (deterministic) multiobjective optimization. Collette and Siarry (2003) systematically present a comprehensive analysis of multiobjective optimization methods from scalar methods to metaheuristics. Deb (2001) introduces the use of evolutionary algorithms in multiobjective optimization. Other books that provide a comprehensive treatment on the design of multiobjective evolutionary algorithms and their applications are by Tan et al. (2005), Abraham et al. (2005), and Coello et al. (2002).

Chapter 2 does not present a detailed theoretical enumeration and analysis of all multiobjective optimization methods but instead presents an overview of multiobjective optimization and a discussion of well-known deterministic methods and heuristic techniques. Section 2.1 presents the basic concepts of multiobjective optimization in terms of Pareto-optimality and an overview of

scalarization and non-scalarization techniques. Section 2.1.2 presents various scalarization techniques, three deterministic methods are presented namely, the *weighted sum*, the *ε-constraint*, and the *goal attainment* and two metaheuristics namely *simulated annealing* and *tabu search*. The advantages and disadvantages of these methods are also discussed. Other scalarization methods such as the *hybrid* method (Guddat et al. 1985), *the elastic constraint* method (Ehrgott and Ryan 2002) and *Benson's* method (Benson 1985) are not discussed in this chapter since these are derivatives of other techniques. The *hybrid* method combines the *weighted-sum* and the *ε-constraint* methods. The *elastic constraint* method relaxes the *ε-constraints* by using penalty functions. Engau and Wiecek (2007) derived that the *Benson* method can be considered a special case of the *hybrid* method. The details and theoretical discussion of scalarization techniques and similar methods are found in Ehrgott (2005) and in Deb (2001). Section 2.1.2 briefly presents lexicographic ordering as a non-scalarization method.

Section 2.2 introduces the fundamentals of an evolutionary algorithm, its framework, its operators, its models and the implementation guidelines for its proper use. Section 2.3 enumerates several multiobjective evolutionary algorithms from their early implementations to the modern and shows the advantages and disadvantages of each algorithm. The section provides a review of comparative studies of modern MOEAs to show where each MOEA performs best in relation to multiobjective optimization.

2.1 Multiobjective Optimization

2.1.1 Problem Definition

Multiobjective optimization is concerned with the minimization or maximization of a vector of objectives $\boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), f_3(\boldsymbol{x}) \ldots f_m(\boldsymbol{x}))$, $f(\boldsymbol{x}) \in \mathfrak{R}^m$. Its general form can be stated as follows:

$$
\begin{aligned}
&\text{Minimize/Maximize} \quad \boldsymbol{f}(\boldsymbol{x}) \\
&\text{subject to} \\
&\qquad g_j(\boldsymbol{x}) = 0 \qquad j = 1, \ldots, J \\
&\qquad h_k(\boldsymbol{x}) \geq 0 \qquad k = 1, \ldots, K \\
&\qquad \boldsymbol{x}_l \leq \boldsymbol{x} \leq \boldsymbol{x}_u
\end{aligned}
\tag{2.1}
$$

Note that there are m objective functions to be maximized or minimized and $\boldsymbol{x}$ is a vector of n decision variables with J equality and K inequality constraints. The constraints $\boldsymbol{x}_l \leq \boldsymbol{x} \leq \boldsymbol{x}_u$ are upper and lower bounds on the decision variable $\boldsymbol{x}$. If any of the objectives of $\boldsymbol{f}(\boldsymbol{x})$ are conflicting, no single solution to the problem exists. Instead, the outcome is a set of solutions and the concept of nondomination or Pareto optimality must be used to characterize the objectives. A nondominated solution is one in which an improvement in one objective requires a decline in another. To explain this further, consider a feasible region, the decision space $\mathcal{D}$, in the parameter space $\boldsymbol{x} \in \Re^n$ and the objective space $\mathcal{Z}=\{\boldsymbol{y} \in \Re^m\}$ wherein $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ subject to $\boldsymbol{x} \in \mathcal{D}$, the multiple objective function, $f_m(\boldsymbol{x})$ maps the decision space into an objective function space as shown in Figure 2.1.

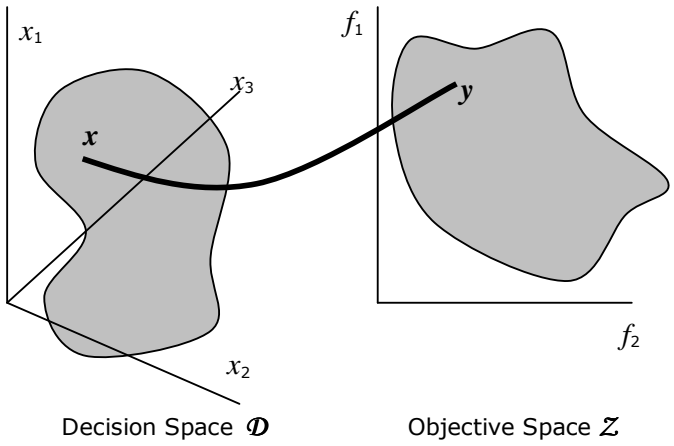

Figure 2.1 Decision space maps to objective space

In a two-dimensional representation of a minimization problem, an objective space and a feasible region are shown in Figure 2.2. The set of nondominated solutions lies on the bold curve (Pareto-optimal front). Points **A** and **B** represent specific nondominated points. The points **A** and **B** are nondominated solution points because an increase in one objective f_1 requires a decline in the other objective f_2.

The concepts can be formalized as follows (Bosman and Thierens 2003):

1) Pareto dominance:
   A solution $\boldsymbol{x}^1$ is said to *dominate* a solution $\boldsymbol{x}^2$ (denoted by $\boldsymbol{x}^1 \preceq \boldsymbol{x}^2$) if and only if
   $$((\forall i \in \boldsymbol{M} \mid f_i(\boldsymbol{x}^1) \leq f_i(\boldsymbol{x}^2)) \wedge (\exists i \in \boldsymbol{M} \mid f_i(\boldsymbol{x}^1) < f_i(\boldsymbol{x}^2)))$$
   where $\boldsymbol{M}=\{1,2,\ldots,m\}$

2) Pareto-optimality:
   A solution $\boldsymbol{x}^1$ is said to be Pareto-optimal if and only if $\neg\exists \boldsymbol{x}^2 : \boldsymbol{x}^2 \preceq \boldsymbol{x}^1$

3) Pareto-optimal set:
   The set $\boldsymbol{P}_S$ of all Pareto-optimal solutions: $\boldsymbol{P}_S = \{ \boldsymbol{x}^1 \mid \exists \boldsymbol{x}^2 : \boldsymbol{x}^2 \preceq \boldsymbol{x}^1 \}$

4) Pareto-optimal front: The set of all objective function values corresponding to the solutions in $\boldsymbol{P}_S$:
   $$\boldsymbol{P}_F = \{ \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), f_3(\boldsymbol{x}) \ldots f_m(\boldsymbol{x})) : \boldsymbol{x} \in \boldsymbol{P}_S \}$$

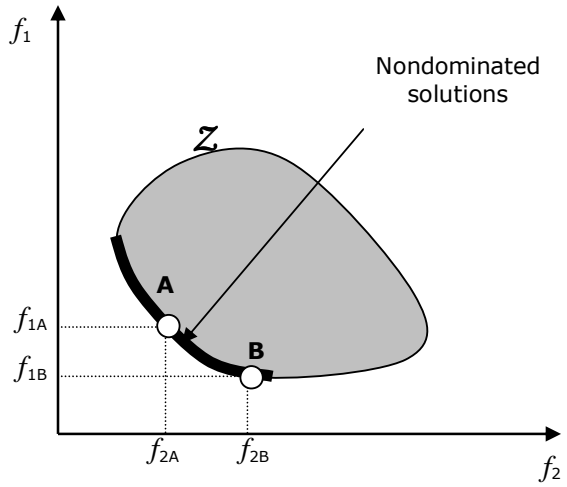

Figure 2.2 Nondominated solutions

Konak et al. (2006) state that a practical approach to multiobjective optimization is to investigate a set of solutions (the best-known Pareto set) that

represent the Pareto-optimal set as much as possible. A multiobjective optimization approach should achieve the following three conflicting goals:

1. The best-known nondominated front should be as close possible as to the Pareto-optimal front. Ideally, the best-known nondominated set should be a subset of the Pareto-optimal set.
2. Solutions in the best-known nondominated set should be uniformly distributed and diverse over of the Pareto front in order to provide the decision maker a true picture of trade-offs.
3. In addition, the best-known nondominated front should capture the whole spectrum of the Pareto-optimal front. This requires investigating solutions at the extreme ends of the objective function space (ends of the bold line as shown in Figure 2.2).

2.1.2 Scalarization methods

Optimization methods are termed traditional or classical in the sense that these algorithms combine multiple objectives into a single function. They are more commonly known as *aggregating functions*. They are easy to implement by transforming the multiobjective problem into a single-objective nonlinear programming problem or into a multiobjective goal-programming problem. However, several disadvantages have been noted: (1) these methods may miss some optimal solutions; (2) they are influenced by the shape of search spaces; and (3) they are time-consuming methods because they should be performed in a series of separate runs to obtain the Pareto-optimal solutions (Das and Dennis 1997). Three scalarizing strategies namely, *weighted sum*, *ε-constraint*, and *goal attainment* are presented to show how aggregating functions work as applied to multiobjective optimization. In addition to the traditional methods, this section presents two global optimization algorithms namely, *simulated annealing* and *tabu search*. Although they are considered global optimization techniques, they search for solutions using scalarized functions as the traditional methods. The following sections present the mechanics of each method and their specific disadvantages in finding Pareto-optimal solutions.

WEIGHTED SUM STRATEGY

The *weighted sum strategy* converts the multiobjective problem $\boldsymbol{f}(\boldsymbol{x})$ into a scalar problem by constructing a weighted sum of all the objectives. This means that the multiobjective optimization problem is transformed into a scalar optimization problem of the form

$$\min_{x \in D} f(\boldsymbol{x}) = \sum_{i=1}^{m} w_i f_i(\boldsymbol{x}) \tag{2.2}$$

where $w_i \geq 0$ are the weighting coefficients representing the relative importance of the objectives. The weighting coefficients usually correspond to the relative importance of the objectives but this is not always necessary. The problem reduces to determining the weight coefficients for each objective.

As an illustration, consider the bi-objective case as shown in Figure 2.3. A line S is drawn below the objective function space. The optimization finds the value of **B** for which S touches the boundary of $\mathcal{Z}$ as it proceeds outwards from the origin. The weights w_1 and w_2 define the slope of S, which leads to the solution point **B** where S touches the boundary of $\mathcal{Z}$.

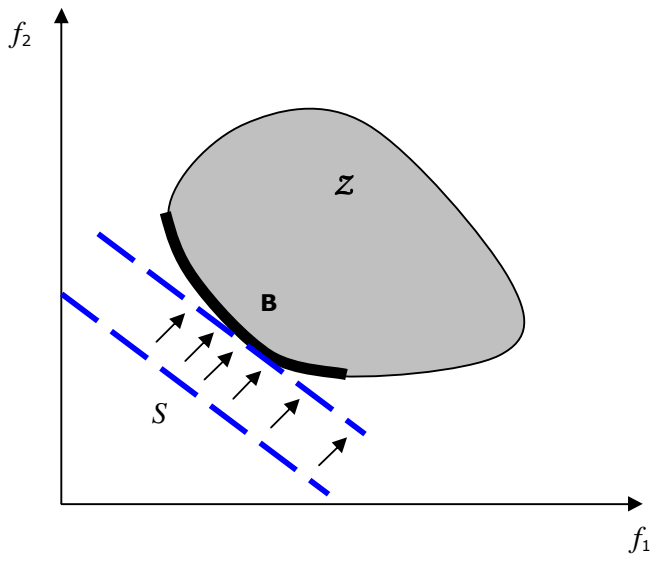

Figure 2.3 Weighted Sum Method

This approach is the simplest way to solve problems with convex Pareto-optimal fronts but a uniform spread of the vector of weights $\boldsymbol{w}=\{w_1, w_2, \ldots, w_m\}$ does not

produce a uniform spread of points on the Pareto curve. If the Pareto curve is not convex, no $\boldsymbol{w}$ exists for which the solution S lies in the non-convex part. Figure 2.4 illustrates a non-convex objective space and shows why the weighted sum approach cannot capture the set of Pareto-optimal solutions in a non-convex objective space.

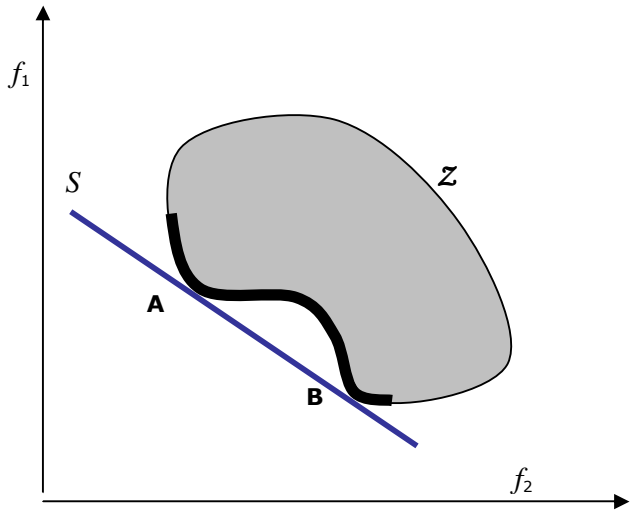

Figure 2.4 Non-convex problem

ε-CONSTRAINT METHOD

The ε-Constraint Method is based on minimizing one (primary) objective function, by treating the other objectives as constraints bounded by some acceptable levels ε (Haimes et al. 1971). A single objective minimization is carried out for the most relevant objective function f_p subject to additional constraints on the other objective functions. The ε-constraint method reformulates the multiobjective problem by keeping a primary objective, f_p, and expressing the other objectives in the form of inequality constraints

$$\min_{x \in D} f_p(\boldsymbol{x}) \qquad (2.3)$$

$$\text{subject to} \quad f_i(\boldsymbol{x}) \le \varepsilon_i; i = 1,...,m; i \ne p$$

Figure 2.5 illustrates the ε-constraint method for a bi-objective problem where f_1 is treated as the primary objective and f_2 as a constraint such that $f_2(\boldsymbol{x}) \leq \varepsilon_2$. The value ε_2 divides the feasible objective space into two parts and the left portion becomes the feasible region. This approach then tries to find the solution that reduces or minimizes the new feasible region, in the above example, at the solution point $f_1 = f_{1s}$ and $f_2 = \varepsilon_2$.

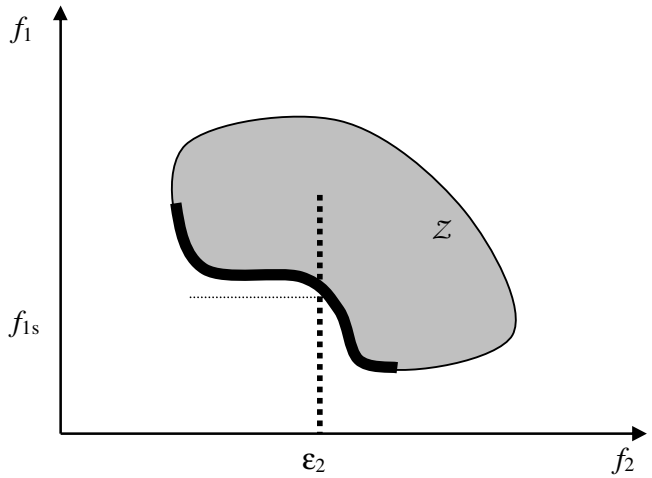

Figure 2.5 ε-Constraint Method

This method faces the problem that the new feasible region depends on the value of ε and finding a suitable value of ε to ensure a feasible solution is not straightforward. A further disadvantage of this approach is it requires more information from the user as the number of objectives increases. The obvious drawback is that it is time consuming, and the coding of the objective functions may be difficult or even impossible for problems with too many objectives (Coello 2000).

GOAL-ATTAINMENT METHOD

The goal attainment method is a variation of goal programming wherein targets or goals are assigned for each objective and the objective function minimizes the sum of the absolute value of differences between target values and computed values. Although the method works on vectors, it is still considered an

aggregating approach. The method requires the user to assign a vector of weighting coefficients, $\boldsymbol{w}=\{w_1,w_2, \ldots,w_m\}$ relative to the degree of under- or over- achievement of the objectives. This involves expressing a set of design goals, $\boldsymbol{p} = \{p_1,p_2,\ldots,p_m\}$ that is associated with a set of objectives, $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), f_3(\boldsymbol{x}) \ldots f_m(\boldsymbol{x}))$. The optimization problem is to find a solution $\boldsymbol{x}^*$ by minimizing a scalar α and is formulated as

$$\min_{\substack{\alpha\in\Re,\\ x\notin D}} \alpha \qquad (2.4)$$

$$\text{subject to} \quad f_i(\boldsymbol{x}) - w_i\alpha \le p_i \; ; \; i = 1, 2, \ldots, m$$

The term $w_i\alpha$ introduces an element of "slackness" or soft constraints into the problem. The slack variable α is an argument that simultaneously minimizes the vector of objectives $\boldsymbol{f}(\boldsymbol{x})$. The goal-attainment method provides a convenient intuitive interpretation of the design problem that is solvable using standard optimization procedures such as nonlinear programming. The goal attainment method is represented geometrically for the two-dimensional problem in Figure 2.6.

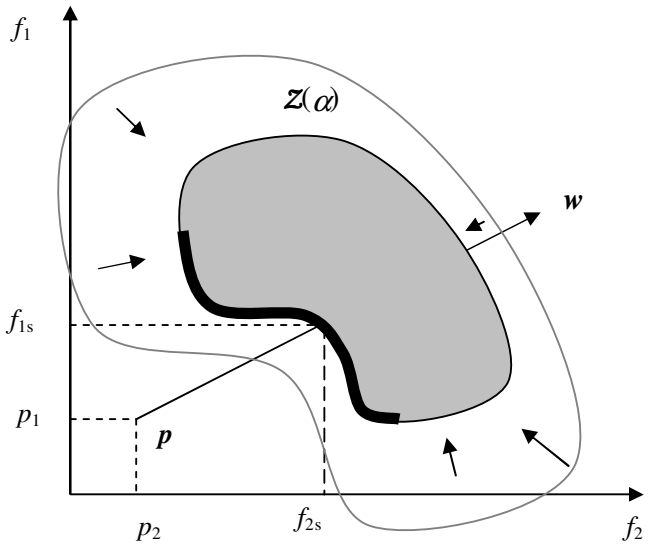

Figure 2.6 Goal Attainment method

The weighting vector $\boldsymbol{w}$ defines the direction of search from $\boldsymbol{p}$ to the feasible function space, $Z(\alpha)$. Specification of the goals p_1 and p_2 defines the goal point,

$\boldsymbol{p}$. The term α, which changes the size of the feasible region is altered during the optimization. The optimal solution is the first point where the vector $\boldsymbol{p} + \boldsymbol{w}\alpha$ intersects the feasible region $Z(\alpha)$. This intersection is the unique solution point (f_{1s}, f_{2s}). The main problem with the method is that it requires some problem knowledge from the decision maker, such as setting logical goals and target values for each objective, and finding a suitable weight vector $\boldsymbol{w}$. Improper values may mislead the search direction towards non-optimal solutions.

Other classical methods that aggregate a multiobjective optimization problem into a scalar optimization problem exist similarly suffer from the difficulties stated above, that is, all algorithms require some problem knowledge to find suitable parameters such as weights, value functions or user-specified targets or goals to find a single Pareto-optimal solution. Deb (1999) explicitly points out the limitations of the classical approaches.

SIMULATED ANNEALING

Figure 2.7 shows a pseudocode for simulated annealing. Simulated annealing was originally intended for combinatorial optimization. The basic idea in simulated annealing is to reduce the possibility of getting trapped in local optima by allowing local search moves from a current solution to its inferior neighbors. The algorithm generates local movement in the neighborhood of the current state and accepts a new state based on a function depending on temperature *t*. The temperature *t* changes as the search progresses.

Serafini (1994) developed a multiobjective simulated annealing algorithm (MOSA) for solving multiobjective combinatorial problems. Serafini's MOSA is a single-point method that optimizes one weighted scalarizing function at each step. Ulungu et al. (1999) suggested a population-based MOSA, which optimizes multiple scalarizing functions separately. Each of them is optimized by a single simulated annealing run. To maintain the diversity of resultant nondominated solutions, a set of fixed evenly-distributed weight vectors are used. Unlike Serafini's MOSA, Ulungu's MOSA has the equal chance to optimize each weighted scalarizing function. Czyzak et al. (1998) proposed a MOSA with adaptive search directions. It also uses a population of solutions to optimize multiple weighted scalarizing functions at the same time. To find the solutions in the unexplored area of the Pareto-optimal front, this approach adaptively tunes the weight vector of each solution during the search according to the closeness

to its neighbors. There are many variations of MOSA in literature and a comprehensive survey can be found in Coello et al. (2002).

Simulated annealing is a robust technique that can handle arbitrary cost functions, is a good option when heuristics are unavailable, and is easy to code. However, repeatedly annealing with a schedule is very slow, especially if the cost function is expensive to compute. Since it uses scalarized functions to find optimal solutions, it also suffers from the disadvantages of scalarization techniques.

```
Simulated_Annealing{
  x0 = initial solution
  t0 = initial temperature (>0)
  while( t > 0 ){
    x' = pick a random neighbor to x        derive a new solution x' by
                                              randomly changing current
                                              solution

    c= f(x) – f(x')                         difference of acceptance
                                              function
    if( c > 0 ) x = x'                      keep the better solution
    else {
      r = random number in range [0…1]
      m = exp(-c/t )
      if( r < m ) x = x'                          take the worse
solution
                                              to avoid local optimum.
    }
    t = reduced (t);                        decrease the temperature
  }
}
```

Figure 2.7 Simulated annealing pseudocode.

TABU SEARCH

The general algorithm of tabu search (Glover 1989) is shown in Figure 2.8. The basic idea of tabu search is to create a subset $\boldsymbol{T}$ from $\mathcal{N}$ whose elements are called tabu moves form historical information of the search process. Membership in $\boldsymbol{T}$ is awarded either by a historical list of moves detected as unproductive or by a set of tabu conditions. The subset $\boldsymbol{T}$ limits the search and

keeps tabu search from becoming a simple hillclimber. At each step of the algorithm, a best movement defined by the function *opt*() is chosen.

Hansen (1997) proposed a multiobjective tabu search (MOTS*) that generates random solutions as starting points of the algorithm. A weight vector is determined for each of these solutions based on a weighted metric that distributes the solutions uniformly along the Pareto front. Gandibleux et al. (1997) proposed a MOTS based on the use of a utopian reference point. The utopian point used is the best objective function value for each objective from the solution in the neighborhood of current solutions. Weights are used in the aggregating function and are changed periodically to promote diversity.

```
Tabu_Search {
1:  x_0 = initial solution
2:  x* = x                                   x* is the best solution
3:  c = 0                                    initialize iteration counter
4:  T = ∅                                    T is the set of tabu moves
    if (N(x) − T = ∅)  goto step 4           N(x) is the neighborhood
                                              function
    else {c= c+1
        Select n_c ∈ N(x) − T  such that
        n_c(x)= opt(n(x): n ∈ N(x) − T )     opt() is a user-defined
                                              evaluation function
    }
    x = n_c(x)
    if (f(x) < f(x*) then x* = x
    if (stop) N(x) − T = ∅                   check stopping condition
    else {update T
      goto step 4
    }
}
```

Figure 2.8 Tabu search pseudocode

Two tabu lists are used: (1) a list of normal attributes considered tabu that prevents the algorithm from returning to visited solutions, and (2) a list to variate the weights. Other authors have proposed hybrids of the basic tabu search that combine tabu search with other search methods such as hillclimbing, simulated annealing, and population-based methods to find solutions to MOOPs.

A survey of MOTS algorithms and their performance comparisons can be found in Coello et al. (2002). Tabu search has been widely used in combinatorial optimization but its use in continuous search spaces is not extensive. The basic problem with tabu search with multiple objectives is how to generate diverse solutions. In addition, if other search techniques are used for exploration such as EAs, extra computational cost is added to the algorithm. Since tabu search uses weights to find optimal solutions, it also suffers from the disadvantages of scalarization techniques.

2.1.3 Non-scalarization Methods

LEXICOGRAPHIC ORDERING

In lexicographic ordering the decision maker must arrange the objective functions according to their absolute importance. This ordering means that a more important objective is infinitely more important than a less important objective. After ordering the most important objective function is minimized subject to original constraints. An optimal solution $\boldsymbol{x}^*$ in lexicographic ordering is called lexicographically optimal and $f(\boldsymbol{x}^*)$ is a lexicographically a minimal vector in the objective space. The lexicographic optimization problem can be written as

$$\operatorname*{lexmin}_{x \in X} \left(f_1(\boldsymbol{x}),\ f_2(\boldsymbol{x}),\ldots,\ f_m(\boldsymbol{x}) \right) \tag{2.5}$$

The objective functions are arranged according to lexicographic order from the most important f_1 to the least important fm. A feasible solution $\boldsymbol{x} \in \boldsymbol{X}$ is lexicographically optimal if there is no $\boldsymbol{x}' \in \boldsymbol{X}$ such that $f(\boldsymbol{x}) <_{\text{lex}} f(\boldsymbol{x})$. Miettinen (1998) proves that the solution to the lexicographic problem (2.5) is Pareto-optimal. The advantage of lexicographic ordering is its simplicity but the decision maker may have difficulties in ranking the objectives in terms of their importance. In addition to its drawbacks, the priority ranking implies the absence of tradeoffs between criteria (Ehrgott 2005).

2.2 Evolutionary Algorithms

Different strategies can be used for solving multiobjective optimization problems. On one hand, the decision-making is reduced to a single-objective

function and scalar optimization is used to find the corresponding solution. This approach often requires knowledge of the optimization problem in order to assign proper weights for each criterion. Such approach entails repetition of the optimization procedure until a satisfactory solution is found. On another hand, the decision-making is applied at the end of the optimization to avoid repetition, in which an optimization run generates a set of solutions. The optimal solution in this case is the Pareto-optimal set. However, the size of the Pareto-optimal set may be infinite in some instances and is impossible to find with a finite number of solutions. In such a case, the preferred result is a subset of the Pareto-optimal set and such a subset of solutions can be generated using evolutionary algorithms. Bosman and Thierens (2003) state that searching a space by maintaining a finite population of solutions is characteristic of EAs, which makes them natural candidates for multiobjective optimization aiming to find a good approximation to the Pareto-optimal front.

Evolutionary algorithms (EA) represent a subset of generic population-based metaheuristic optimization algorithms in Artificial Intelligence. They are stochastic methods that use techniques motivated by natural evolution such as, random variation, recombination, selection, and competition of individuals in a population. Genetic algorithms, Evolutionary Programming, and Evolution Strategies are the mainstream computational models of EA whereas Genetic Programming and Learning Classifier Systems are related evolutionary techniques.

Evolutionary algorithms work on a population of solutions. With the application of evolutionary or genetic operators, they produce improving approximations of solutions to a problem as the generation progresses. The candidate solutions in a population are referred to as chromosomes or individuals and depending on the EA model, individuals are coded as bit strings, real-valued vectors, trees, graphs, or matrices. A new set of chromosomes is created by selecting individuals according to their level of fitness in the problem domain. The fitness of an individual is a computed measure that is a function of the objective value, i.e. if the problem is to minimize an objective function value the individual with the smaller fitness value is the better solution. At each generation, the genetic operators lead to the creation of a population of new individuals that are fitter than their parents. The process of reproduction and selection continues until the algorithm satisfies a termination criterion. Figure 2.7 shows a template of a general evolutionary algorithm.

At the start of the procedure, a number of individuals (the population $\boldsymbol{P}_t$; t=0) are randomly initialized or may be seeded. Then the objective function is evaluated for these individuals. If the optimization criteria are not met (termination condition), creation of a new generation starts. Individuals are selected according to their fitness for the production of offspring. Parents are recombined and/or mutated with a certain probability to produce offspring. The fitness of the offspring is then computed. Competent offspring are inserted into the population replacing the parents, producing a new generation $\boldsymbol{P}_{t+1}$. The loop is repeated until the termination criteria are met.

```
procedure EA; {
    t = 0;
    initialize population P_t
    evaluate P_t
    while (termination condition not satisfied)
        {
        select parent P_t
        variate P_t
        evaluate P_t
        select environment P_{t+1}
        }
}
```

Figure 2.9 Evolutionary algorithm template

An object or an individual that forms a solution in the solution space (phenotype space) of the problem is called a *phenotype* or a candidate solution, whereas an individual in the EA search space is called a *genotype* or chromosome and the search space as the search space (genotype space). A chromosome is composed of genes and the location of a particular gene in a chromosome is referred to as the *locus*. A gene may assume different values and each value of a gene is called an *allele*. The terms 'chromosome', 'individual', and 'solution' are treated the same since a chromosome represents only a single solution.

The following sections introduce the significant elements and genetic operators used in evolutionary algorithms. The last subsection section presents a discussion of the interaction between the genetic operators and other important issues in implementing evolutionary algorithms to solve problems of varying levels of difficulty.

2.2.1 Representation

An EA solves a particular problem by first creating an encoding function from the solution space of the problem (phenotype space) to the search space of the EA (genotype space). For instance, if the solution space is composed of integers, an EA may encode the integers by their binary equivalent, which makes the search space binary coded. Apparently, the search process is done in the genotype space and as such may be very different from the problem's solution space. A solution is obtained by decoding the fittest chromosomes after the EA has terminated. However, there are problems wherein the use of binary encoding is not feasible, specifically in optimization problems that have decision variables that take continuous values. In such a case, real-valued or real-parameter encoding is necessary – each solution is represented as a real-valued vector. Intuitively, the phenotype space is the same as the genotype space.

It is important to understand the strength of an evolutionary algorithm depends on a robust encoding or representation scheme. The representation scheme also dictates the type of variation operators, which controls the exploitation and exploration of the search space. Discussion on this is presented in Section 2.2.3.

2.2.2 Population size and its initialization

The basic issues that arise initially in the use of evolutionary algorithms are determining the *size of the population* and the manner of inserting individuals in the initial population. The choice of the population size has been studied in different perspectives and it seems natural that the population size can be determined in terms of string length. Goldberg (1985), through a schemata paradigm for binary strings, shows that the population size increases as an exponential function of the string length. A later study (Goldberg et al. 1992) shows a linear dependence of population size on string length is adequate. Reeves (1993) derives an expression to determine population size from q-ary alphabets on different confidence levels on the assumption that the initial population is generated by a random sample with replacement and that at least one allele is present at each locus can be found.

In multiobjective optimization, it is known that the population size of solutions increases exponentially with the number of objectives (Deb 2001). There are two common options to respond to this problem: (1) use a large population or (2) integrate a dynamic population sizing procedure in the GA. Implementations for single-objective EAs has shown promising results, but dynamic sizing has remained a challenge to multiobjective optimization. Hence, there is no better alternative but to estimate the size of the initial population as a function of the number of objectives. Deb (2001) provides an approximation chart for finding the minimum population size as a function of the number of objectives.

The second problem is in the choice of a method by which the *initial population* is to be filled up. Usually the population is initialized randomly but this does not necessarily envelop the search space uniformly. Seeding the initial population with a known solution from other heuristics can facilitate an EA to find better solutions but on the contrary, the study by Surry and Radcliffe (1996) find that seeding the initial population may reduce the quality of solutions found.

2.2.3 Variation of the population

The purpose of *variation operators* is to improve the diversity of the population by applying recombination and mutation to selected chromosomes of the preceding generation. Variation in the genotype space similarly finds new candidate solutions in the phenotype space. The recombination operator creates new individuals by combining genes of two or more parents whereas mutation creates new individuals by variation of a single parent. In a binary-coded EA, the crossover operator is mainly responsible for the search aspect, whereas mutation introduces variability and keeps diversity in the population. Variation methods are grouped into two types: binary-coded variation and real-valued variation operators.

BINARY RECOMBINATION

This section describes recombination methods for individuals with binary variables or binary strings. Commonly, these methods are called *crossover*.

During the recombination of binary variables, only parts of the parents are exchanged between them depending on the number of parts (the number of crossover points) the individuals are partitioned before the genes are swapped. Crossover replaces some of the alleles in one parent by alleles of the corresponding genes of the other.

Single/Multipoint Crossover

In a single-point crossover, one crossover position is selected uniformly at random and the genes exchanged between the individuals about this point produces two new offspring as shown in Figure 2.8. In a double-point crossover, two crossover positions are selected uniformly at random and genes are exchanged between the individuals between these points, produce two new offspring.

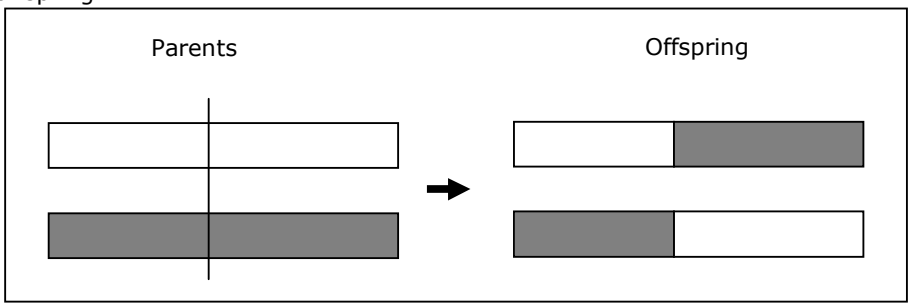

Figure 2.10 Single-point crossover

For a multipoint crossover, m crossover positions are chosen at random with no duplicates and then are sorted in ascending order. The genes between successive crossover points are exchanged from both parents and produce two new offspring. The genes before the first crossover point are not exchanged between the parents as shown in Figure 2.9. Single and multipoint crossover techniques define cross points as places between loci where an individual can be split. Uniform crossover generalizes this scheme to make every locus a potential crossover point. An offspring is created by choosing every gene with a probability p (usually equal to 0.5) from either parent.

Eshelman et al. (1989) state that the single-point crossover has considerable positional bias. It favors substrings of contiguous bits of a chromosome without being sure if the chromosome is moving towards a good solution. The idea

behind the multipoint crossover and many of the variants of the crossover operator is that parts of the chromosome that contribute most to the performance of the chromosome may not necessarily be located in adjacent substrings. Multipoint crossover appears to encourage the exploration of the search space, which makes the search more robust rather than favoring the convergence to fit individuals early in the search.

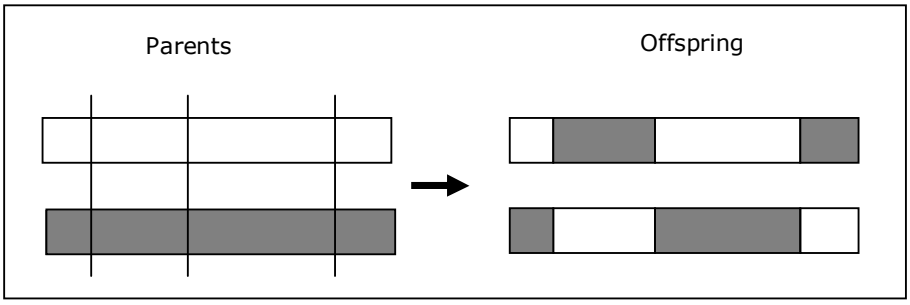

Figure 2.11 Multipoint crossover

REAL-VALUED RECOMBINATION

The crossover operator with one or more crossover points is easy to apply on binary strings but is not suitable for real-valued vectors. The main task in real-valued recombination is how to create new offspring in a logical manner. Recombination cannot perform search in real-valued variables with respect to each gene. Hence, it is necessary to practice caution in the usage of recombination in real-parameter optimization. Many different real-valued crossover operators exist and the issue of which crossover operator is better is context-dependent or problem-specific. Three of the many real-valued crossover operators are presented below and these are discrete recombination, intermediate recombination, and line recombination.

Discrete Recombination

Discrete recombination is analogous to uniform crossover with binary encoding. For each position, the parent who contributes its genes to the offspring is chosen randomly with equal probability. Discrete recombination can be used with any data type variable (binary, integer, real or symbols). In a two-dimensional real-

search space, the offspring can be either the parents themselves or the other two diagonal solutions as shown in Figure 2.10. This crossover operator has insufficient search power since the locations of new offspring are limited to the variable boundaries.

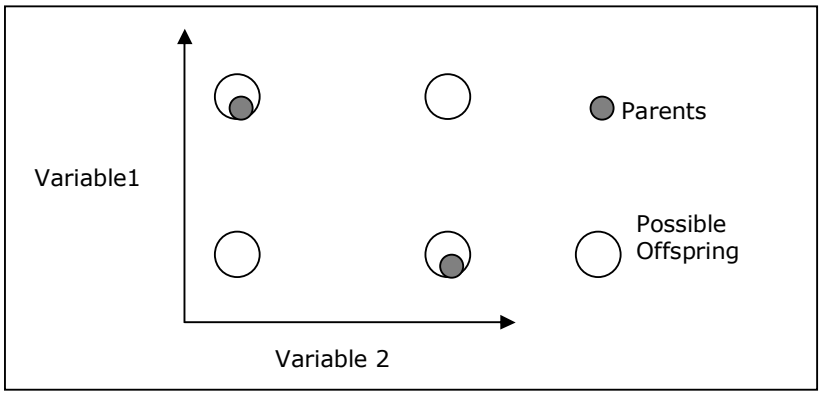

Figure 2.12 Discrete recombination

Intermediate recombination

Intermediate recombination is a method only applicable to real variables and not to binary variables. The genes of the offspring are chosen somewhere around and between the genes values of the parents. Intermediate recombination is capable of producing any point within a hypercube slightly larger than that defined by the parents. Figure 2.11 shows the possible area of offspring after intermediate recombination.

The gene of an offspring at position i is produced according to the rule (Dumitrescu et al. 2000):

$$z_i = x_i\alpha_i + y_i(1 - \alpha_i) \tag{2.6}$$

where α is a scaling factor chosen uniformly at random over an interval [$-d$, $1+d$]. In intermediate recombination d=0, for extended intermediate recombination $d > 0$. A good choice is d=0.25. Each variable in the offspring is the result of combining the variables according to the above expression with a new a chosen for each variable.

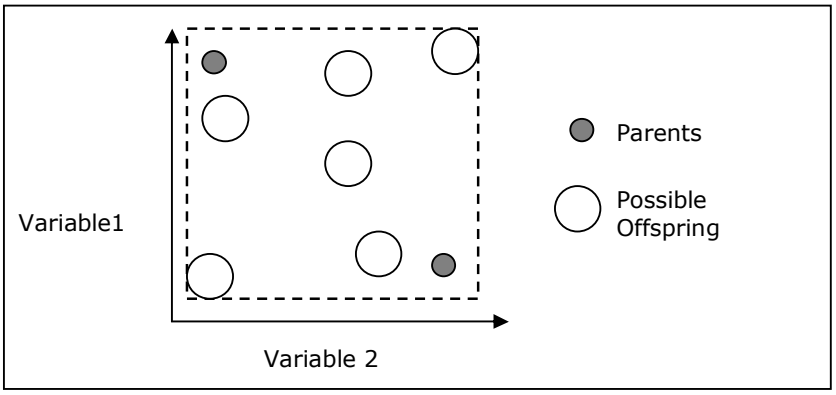

Figure 2.13 Intermediate recombination

Line recombination

Line recombination is similar to intermediate recombination except that one value of α is used for all the genes. Line recombination can generate any point on the line defined by the parents as shown in Figure 2.12. The line recombination operator and its variants possess a feature that may constitute an adaptive search.

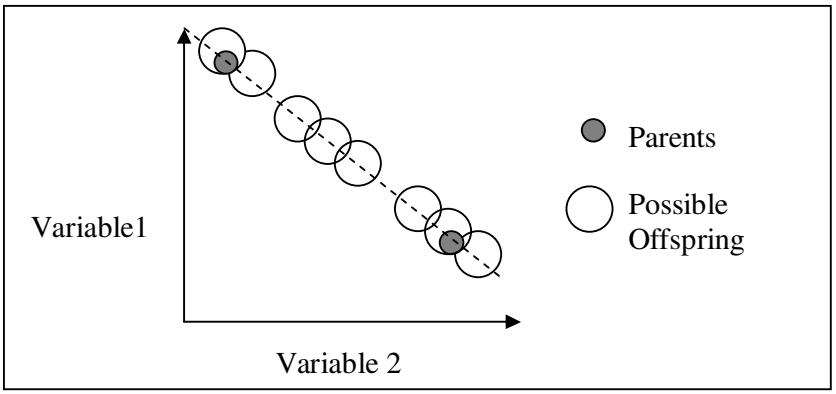

Figure 2.14 Line recombination

Consider Figures 2.11 and 2.12, if the difference between the parent solutions is small, the difference between offspring solutions is also small, and if the difference between the parent solutions is large, the difference between offspring solutions is large. If the initial population is randomly chosen over the

entire search space, the intermediate recombination operator at the early stages of the EA allows search of the entire search space and continues to converge to solutions in some region as the generation progresses. Beyer and Deb (2000) found similarity in the different real-parameter crossover operators and they postulate that the crossover operator should (1) keep the mean objective values of the offspring population the same, and (2) increase the population diversity in general.

MUTATION

For binary valued individuals mutation means the flipping of gene values, because every gene has only two states. Thus, the size of the mutation step is always one.

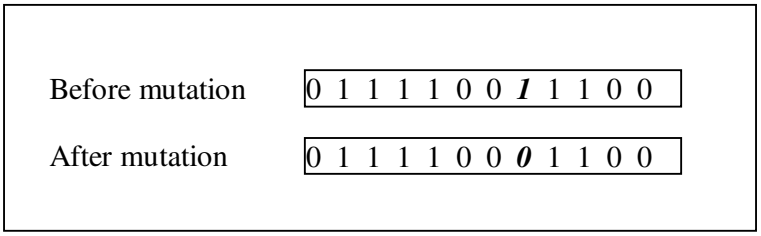

Figure 2.15 Binary mutation

Mutation of real variables means that randomly generated values are added to the variables with a low probability. The probability of mutating a variable is inversely proportional to the number of genes. The more genes one individual has, the smaller is the mutation probability. Mühlenbein and Schlierkamp-Voosen (1993) write that a mutation rate of $1/n$ (n: number of genes of an individual) produces good results for a wide variety of test functions. This means that per mutation only one variable per individual is mutated. Thus, the mutation rate is independent of the size of the population.

The size of the mutation step is usually difficult to choose. The optimal step-size depends on the problem considered and may even vary during the optimization process. It is known that small steps are often successful, especially when the individual is already well adapted. However, larger changes can produce good results much faster. Thus, a good mutation operator should

often produce small step-sizes with a high mutation probability and large step-sizes with a low probability.

The proper choice of the recombination and mutation operators is crucial in the current study since the individuals in the study's problem classes can only be represented as either real or discrete numbers. The MOOP in Chapters 3 and 6 require a real-parameter representation whereas the MOOP in Chapters 4 and 5 are combinatorial problems and require a discrete-parameter representation. Consequently the recombination and mutation operators became dependent on the coding or representation of an individual in each problem class.

2.2.4 Evaluation of an individual

Associated with the selection operation is the evaluation function, more commonly known as the *fitness function*. The evaluation function is the basis of control of the search progress and forms the foundation for the *selection* operator. Its task is to assign a quality measure to evaluate the relevance of each chromosome. The quality measure may be derived from the objective function of the optimization problem through some transformation or in some cases, the fitness function may be identical to the objective function. For most numerical and combinatorial optimization problems, the fitness function corresponds to the problem's objective function. For example, in a minimization problem the *selection* operator chooses individuals with the lowest fitness values whereas in a maximization problem the *selector* chooses the highest fitness values.

In addition to the computation of the fitness of a chromosome, it may be relevant for most decision-makers to have knowledge about the fitness landscape generated by the fitness function. The fitness landscape shows the surface created by the chromosomes and may help the decision-maker to visualize the search progress of the EA.

2.2.5 Selection

The aim of selection is to focus the search process on the most promising regions of the search space. It is based on the quality of the individuals as

defined by their fitness. Selection involves two tasks: (1) the selection of individuals for variation (parent selection) and, (2) the selection for replacement (environmental selection) which selects new individuals and parents to be kept in the population. The selection of individuals for variation promotes high reproductive probability of the fittest individuals and preserves the diversity of solutions in the population whereas the selection for replacement forms the population of the fittest individual for the next generation.

Selection techniques are grouped into three schemes: proportional selection, scaling, ranking procedures, and tournament selection. Proportional selection is commonly known as the Monte Carlo selection or roulette wheel selection. Variants of proportional selection include stochastic sampling with replacement, stochastic universal sampling, and truncation selection. In proportional selection schemes, the fittest individuals will dominate parent selection and may mislead the search process. Scaling and ranking mechanisms avoid the problems of proportional selection by reducing the domination of fittest individuals in parent selection (Dumitrescu et al. 2000).

SELECTION PRESSURE

Selection pressure is defined as the degree to which highly fit individuals are allowed to produce offspring in the next generation. It may also be defined as the ratio of the probability of selecting the fittest individual and the probability of selecting an average individual. High values of the selection pressure strongly favor the best individuals in the population. In effect, this drives the parent selection to just a few individuals making it more *elitist*. To avoid such a phenomenon, higher recombination and mutation probabilities need to be introduced to the EA (Dumitrescu et al. 2000).

SCALING

In most EAs, using the objective function scores in selection as in proportional selection may be insufficient because the scale wherein a chromosome is measured is important. Fitness scaling is a method that converts the raw fitness scores computed by the fitness function to values in a range that is suitable for the selection function. The selection function uses the scaled fitness values to select the parents of the next generation. The selection function assigns a higher

probability of selection to individuals with higher scaled values. The transformation may either be static (independent of time) or dynamic. Common *static scaling* procedures include linear scaling, logarithmic scaling, and power law scaling. It has been shown that the use of static scaling may force the EA to rapidly converge to a local domain and therefore it limits the exploration of the search space. *Dynamic scaling* alleviates this problem by changing the scaling transformation at each generation. The method is cumbersome due to the repetitive rescaling of the fitness values. The ranking method and tournament selection are alternative techniques among dynamic scaling that provide better solutions (Reeves and Rowe 2003).

THE RANKING METHOD

The ranking method ranks the individuals in the population according to their fitness and the selection probability of an individual is defined as function of its rank, usually a linear function. Ranking is simple and more efficient than scaling and avoids premature convergence. However, computing the ranks of each individual, which is computationally more expensive than linear scaling, requires sorting the whole population according to their rank. An elegant property of ranking is that it maintains a constant selection pressure without rescaling at each generation. In the ranking method, the worst fit individuals have the lowest reproduction rate and the best-fit individuals have the highest reproduction rate (Blicke and Thiele, 1995)

TOURNAMENT SELECTION

Tournament selection is similar to linear ranking in the sense that it reduces selection pressure. Binary tournament selection is the most commonly used procedure under this scheme. This type of selection chooses two chromosomes at random for which the fitness is calculated. The fitter individual is selected to become a parent in the next generation. The procedures iterates until the population of parents (mating pool) is filled up. The q-tournament selection is a generalization of the binary tournament selection where q, the tournament size, defines the number of chromosomes that compete in the tournament. There exist several variants of tournament selection, like probabilistic tournament, Boltzmann tournament, and score-based tournament. The potential advantage of tournament selection over the other schemes is that it does not require a

global fitness comparison in a population. It only needs a preference ordering between a set of individuals (Reeves and Rowe 2003).

SELECTION FOR REPLACEMENT

The discussion above is focused on parent selection. The issue of selection of offspring for the next generation remains to be addressed. It answers the question whether the EA allows all offspring to replace the parents. The environmental selection distinguishes the survivors among the parents and their offspring. The selection strategies discussed previously may be applied to the environmental selection phase. The only difference between parent selection and environmental selection is the phase wherein they are evaluated in the evolutionary cycle. The question that remains to be answered is the number of offspring and parents to be kept in the new generation.

In a *generational* model, a new population of offspring replaces the whole parent population, which means that selection is for mating only. A variant of this type is the generational elitist strategy – it keeps the best individuals of a generation and allows it to survive in the succeeding generation. The elitist strategy preserves good individuals to stay for more than one generation unless it is not chosen for mating and variation. Elitist strategies increase the speed of convergence of an evolutionary algorithm. An alternative to the *generational* model is the *steady-state* model wherein a number of parents (μ) are selected based on their fitness and a number of offspring (λ) are generated through variation functions. All of the offspring are inserted into the new population and μ parents are discarded by fitness ranking selection or tournament selection. The concept of generation in its strictest sense is no longer clear since the new population is composed of both parents and offspring.

There are other replacement strategies but it is suggested that generational and elitist strategies work better than pure generational strategies in the context of the better usage of solutions and convergence to an optimum solution. The reader is referred to Dumitrescu et al. (2000) and Deb (2001) for further reading on other replacement strategies.

2.2.6 Models of evolutionary algorithms

GENETIC ALGORITHMS

The genetic algorithm (GA) was developed by Holland in the early 1960's in his work on natural and adaptive systems. Genetic algorithms are instrumental in defining the fundamental concepts of evolutionary algorithms and they embody the main paradigm of evolutionary computation (see the above discussion on EA concepts).

The traditional representation of a chromosome is a binary string of fixed length, but arrays of other data types and structures can be used similarly in the same way. The search and variation operators are typically mutation and recombination but recombination is deemed more important in searching for solutions. Mutation on the other hand is used to induce variability and prevent premature convergence. However, in recent literature, there is no reason to presume that recombination must be more important that mutation. A strategy of recombination-and-mutation is not always necessary. It is also possible to use recombination-or-mutation (Deb and Agrawal, 1999b).

EVOLUTIONARY STRATEGIES

The method of Evolutionary Strategies (ES) is due to Ingo Rechenberg (1973). Evolutionary strategies use real-valued vectors to represent individuals and use mutation as the primary search operator. An individual is usually represented as a pair, a vector $\boldsymbol{x}$ that corresponds to a point in the search space, and $N(0, \sigma)$, the standard deviation vector, is a vector of instances created using a zero-mean normal distribution with a standard deviation σ.

The *replacement strategy* is deterministic and is based on the fitness rankings, not on the actual fitness values. The first and simplest model, the (1+1)-ES, operates on the current individual x_t (parent) and its descendant x_{t+1}. The better of x_t and x_{t+1} is selected to become parent of the next generation, whereas the lesser is discarded. In the $(\mu+1)$ –ES, μ parents may generate one descendant at a time and the least fit individual is discarded. The $(\mu+\lambda)$-ES and (μ, λ)-ES where $\lambda \geq \mu$, use multiple populations of parents and descendants to

increase the population sizes. This process increases the convergence rate. In the (μ+λ)-ES, a number of parents μ are used to generate λ descendants and all the μ+λ individuals compete for survival in the intermediate generation. In the (μ, λ)-ES, the entire parent population is replaced by survivors of λ descendant. This makes the lifespan of a solution to be limited to only one generation. This strategy is well suited in cases where the search is affected by noise.

The simplest *mutation* is performed by adding a normally distributed random value to each vector component – the amplitude given by the standard deviation vector. The mutation step size or mutation strength is often governed by self-adaptation depending on the strategy used. In the (1+1)-ES, the mutation step size is equal to the standard deviation. In the other strategies, the individual step sizes for each point or correlations between points are governed either by standard self-adaptation or by correlated adaptation (Dumitrescu et al. 2000).

EVOLUTIONARY PROGRAMMING

The method was introduced by Fogel in the 1960's in his work on artificial intelligence – generation of intelligent behavior in a machine as regards to prediction of the environment. Intelligent behavior in evolutionary programming (EP) is generated and described by deterministic finite-state machines. Mutation is the only search and variation operator since recombination of two finite-state machines or automata does not seem to be useful in exploring solutions. EP is a mutation-based EA applied to discrete search spaces. In real-valued problems, evolutionary programming is very similar to evolutionary strategies in the sense that distributed mutations are performed on each decision variable and a self-adapting rule is used to update the mutation strengths. Variants and extensions of EP were suggested by Fogel (1992).

Two main differences can be identified between EP and ES. EP typically uses stochastic selection via a tournament in which selection eliminates those solutions with the least wins, whereas ES typically uses deterministic selection in which the worst solution are purged from the population depending on their function evaluation. In EP, no recombination is used whereas ES variants introduce recombination operators.

GENETIC PROGRAMMING

The genetic programming (GP) model was introduced by Koza (1989) and deals with automatic programming. It is a technique used to optimize a population of computer programs according to a fitness function with respect to a given problem. The GP searches for the fittest computer program. The fundamental structures considered for evolution are tree structures, graph structures, and linear structures. GP favors the use of programming languages that naturally embody tree structures since trees can easily represent mathematical structures that can be evaluated in a recursive manner.

In a tree-based representation, *crossover* is applied on an individual by simply switching a subtree with another subtree from another individual in the population. The expressions resulting from crossover are very much different from their initial parents. Several *mutation* techniques can be applied within genetic programming. In a tree structure, mutation can replace a node of an individual and its subtree (macromutation) or replace just the node's information (micromutation). Mutation and crossover are applied separately. In other words, either crossover, mutation, or neither is applied to each individual. Both crossover and mutation are not applied to the same individual.

GP does not require a special replacement strategy and a parent-selection operator but tournament selection is preferred for large problem classes (Dumitrescu et al. 2000).

LEARNING CLASSIFIER SYSTEM

A Learning Classifier System (LCS), is a machine learning system that evolves rules to adapt to a given classifier system – a learning system using evolutionary algorithms for rule discovery. It consists of a population of binary rules on which an evolutionary algorithm is utilized to generate the best rules. Learning classifier systems process rules in parallel and are grouped into two types depending upon the learning standpoint. Classifiers or rules consist of a pair of attributes, a condition, and an action, and are usually binary-coded. Two approaches exist: the Pittsburgh approach, and the Michigan approach. A Pittsburgh-type LCS has a population of separate rule sets, where the evolutionary algorithm evolves a population of rule bases and finds the best rule set. In a Michigan-type LCS, there is a single population of rules and the

algorithm focuses on selecting the best set of rules in the population. The Pittsburgh approach can be implemented with minimal modifications to an evolutionary algorithm whereas the Michigan approach requires reinforcement and learning procedures to determine the fitness of rules, to discover new rules, and to show how rules interact with other rules.

The Michigan approach introduced by Holland (1976) learns by interacting with an environment from which it receives feedback in the form of a numerical reward. Learning is achieved by trying to maximize the amount of reward received. There are many models of LCSs and many ways of defining what a learning classifier system is. Most LCS models consist of four main components. First, a finite population of condition-action rules (classifiers)that represent the current knowledge about the system. Second, a performance component that governs interaction in the environment. Third, a distribution component that distributes the reward received from the environment to the classifiers responsible for the rewards obtained. Fourth, a discovery component that is responsible for discovering of better rules and improving existing ones through a genetic algorithm.

2.2.7 Interaction among genetic operators

The balance between exploitation and exploration is an important issue in the usage of GAs. If the selection operator, which exploits fit individuals, applies excessive selection pressure, the population loses its diversity rapidly. In order to maintain the diversity of solutions, the use the recombination and mutation operators should be high. The variation operators must generate offspring that are reasonably different from the parent solutions. Otherwise, the population may converge to a suboptimal solution. On the contrary, if the selection pressure is insufficient and does not exploit good solutions, the GA's search process behaves like random search.

The balance of exploitation and exploration issues was studied by Goldberg et al. (1993). Goldberg and Deb (1991) calculated the *takeover time* of a number of selection operators. The takeover time is defined as the number of generations required for the best solution to occupy all of the population slots by repetitive application of the selection operator only. The takeover time provides information about the speed of how the best solution in a population is emphasized. They observed that binary tournament selection and linear ranking

selection have the same takeover times and that proportionate selection is much slower than tournament selection.

Deb and Agrawal (1999) evaluated a series of experiments with different operators and parameter setting as applied to problems of different difficulties and concluded the following:

1) For simple problems such as uni-modal and linear problems, a genetic algorithm with a selection and a crossover or mutation operator, or a combination of crossover and mutation operators, can all work satisfactorily. For a selector-mutation combination GA, a small population size provides the optimum performance. Since the mutation operator behaves similar to local search, it may require more iterations. For a selector-crossover GA without a mutation operator, the population requirement is high but the number of generations required may be smaller.

2) For difficult problems with multimodality and high dimensionality of the search space, selector-mutation GAs do not work successfully in finding the optimum solution whereas selector-crossover GAs with adequate population sizes find the correct optimum.

While there is a vast collection of empirical and theoretical studies on GAs there is no singular formula for setting the parameters of a GA. Reeves and Rowe (2003) tentatively suggest the following recommendation in implementing a GA:

1) An initial population of about 50 should contain sufficient alleles for the GA to make successful progress
2) Prior knowledge should be used along with randomization in choosing the initial chromosomes.
3) Tournament selection is more efficient than roulette wheel selection. Two-point or uniform crossover have less positional bias the one-point crossover.
4) For a guarantee that an algorithm will eventually converge to the optimum, incremental reproduction and replacement usually make better use of resources than the generational approach.
5) The role of crossover is still not well understood. Although the crossover operator can jump over certain gaps in the search space or speed up search, design of good recombination operators is difficult unless there is sufficient knowledge of which properties are being preserved during the process.

6) An adaptive mutation rate appropriate to the application should be used, but if in doubt, a fixed rate of $1/L$ is a reasonable choice; where L is the length of the chromosome.
7) Diversity maintenance should be prominent in any implementation.
8) Hybridization should be used wherever possible.
9) GAs are stochastic by nature; several replicate runs are required for application.

The recommendations above are from single-objective studies. The current study investigates the effect of different parameter settings on MOEAs in multiobjective optimization problems through sensitivity analysis. The sensitivity analysis will verify whether the recommended values given above are applicable to multiobjective optimization problems.

2.3 Multiobjective evolutionary algorithms

The remarkable property of an EA is that it processes a population of solutions in one simulation run whereas classical methods process only a single solution in one optimization run. This feature of maintaining a finite population of solutions, which aims to find a good approximation of the Pareto-optimal front makes an EA an appropriate solution method for multiobjective optimization (Bosman and Thierens 2003). Classical methods require some knowledge in the assignment of weight vectors, target values, and ε-vectors to transform a multiobjective problem into a single-objective optimization problem. EAs eliminate such requirement or transformation since they generate and work on a population of solutions. There is a huge amount of literature on MOEA methods and similarly their application to real-word problems is also numerous. For example Pangilinan and Janssens (2007b) introduce MOEAs and its application to two optimization problems of differing search spaces. A list of references can be found at http://www.lania.mx/~ccoello/EMOO/EMOObib.html.

Section 2.2 described the important elements and operators of an EA. The discussion on the selection of fit individuals was limited to the fitness evaluation of individuals based on a single-objective problem. In order for a basic evolutionary algorithm to work and find Pareto-optimal solutions to multiobjective optimization problems (MOOP), modifications in its evolutionary operators are necessary. Section 2.3.1 describes early modifications to the basic EA and Section 2.3.2 describes improved MOEAs that use elite-preserving

mechanisms (elitist MOEAs). Finally Section 2.3.3 presents recent comparative studies of modern MOEAs that are known to obtain good approximations of the Pareto-optimal front.

2.3.1 Early implementations of an MOEA

VECTOR EVALUATED GENETIC ALGORITHM

The first implementation of an MOEA was suggested by Schaffer (1984). Schaffer's *vector evaluated* GA (VEGA) extends the basic GA to a multiobjective optimization method by dividing the population of size N into m equal subpopulations randomly. Each subpopulation is assigned a fitness based on a different objective function. The variable m denotes the number of objective functions. The advantage of VEGA is that it is easy to implement, only minor changes are needed to convert the basic GA algorithm into a multiobjective GA. The translation does not change the computational complexity of a simple GA, which is $O(N)$.

VEGA selection procedure

Input:	$\boldsymbol{P}_t$	(population of N individuals)
Output:	F	(fitness value)
	$\boldsymbol{P}'$	(mating pool)

Step 1: Set the objective counter i = 1 and mating pool $\boldsymbol{P}' = \varnothing$.
Step 2: For each individual $\boldsymbol{x} \in \boldsymbol{P}_t$ the fitness is computed as $F(\boldsymbol{x}) = f_i(g(\boldsymbol{x}))\cdot g(\boldsymbol{x})$ is a mapping function.
Step 3: For j=1 to (N/m) select individual $\boldsymbol{x}$ from $\boldsymbol{P}_t$ using proportionate selection and copy it to the mating pool $\boldsymbol{P}' = \boldsymbol{P}' + \{ \boldsymbol{x} \}$.
Step 4: Set $i = i+ 1$. If $i \leq m$ go to Step 2 else stop.

VEGA fitness results correspond to a linear combination of the objectives where the weights depend on the distribution of the population at each generation. This means that VEGA tends to find solutions near the best solution of each objective (champion solutions). It was shown that certain points in a concave surface would not be found by this optimization algorithm (Richardson et al. 1989).

Schaffer (1984) also observed that the crossover between champion solutions could not find diverse solutions even in a convex search space.

WEIGHT-BASED GENETIC ALGORITHM

The weight-based genetic algorithm (WBGA) combines the weighted sum approach and the population feature of genetic algorithms. Hajela and Lin (1992) proposed that each individual in a population is assigned a different weight vector in order to find several Pareto-optimal solutions in one optimization run instead of only one Pareto-optimal solution that is associated to only one particular weight vector. Hence, an individual is represented as a string of all decision variables x_i with their corresponding weights w_i and each objective function f_i is multiplied by the weight w_i. The fitness of a solution is computed as the sum of its weighted objective functions. To maintain diversity of weight combinations, the WBGA uses fitness sharing in the objective space by a niching method applied to the substring of the weight vector. The computational complexity of WBGA is $O(mN^2)$.

WBGA fitness assignment

Input: $\boldsymbol{P}_t$ (population)
Output: F (fitness value)

Step 1: For each individual $\boldsymbol{x} \in \boldsymbol{P}_t$ do
Extract weights $w_i (i = 1,2, \ldots, m)$ from $\boldsymbol{x}$
Set $F(\boldsymbol{x}) = w_1 \cdot f_1(g(\boldsymbol{x})) + w_2 \cdot f_2(g(\boldsymbol{x})) \ldots + w_m \cdot f_m(g(\boldsymbol{x}))$

The advantage of this method is its simplicity in implementation. However, it is inherently biased towards convex portions of the Pareto-optimal front. It may create a very high selection pressure if certain combinations of weights are produced at early stages of the search (Coello 1996). As in any weight-based approach, the WBGA fails to find solutions in non-convex Pareto-optimal regions.

MULTIOBJECTIVE GENETIC ALGORITHM

Fonseca and Fleming (1993) are the first researchers to propose a multiobjective genetic algorithm (MOGA) that uses Pareto-based ranking to find

nondominated solutions. It uses a niching strategy to maintain the diversity of solutions while finding nondominated solutions. For a solution $\boldsymbol{x}$, the rank is computed as one plus the number of solutions $\boldsymbol{y}$ that dominate it. After ranking is performed, each solution is assigned a raw fitness according to its rank and afterwards the raw fitness of all solutions for each rank is averaged. This average fitness is then assigned to each solution of the rank so that each solution is sampled at the same rate. The averaging procedure ensures that solutions with higher ranks have higher assigned fitness in the population. Diversity of solutions in MOGA is maintained by following a niching strategy and a sharing function σ_{share}. The shared fitness value of a solution is computed by dividing its assigned fitness by its niche count. It follows that the solutions in lesser-crowded regions will have better shared fitness. This means that solutions in such regions will have a higher selection pressure. The computational complexity of MOGA is $O(mN^2)$.

MOGA fitness assignment

Input: $\boldsymbol{P}_t$ (population of N individuals)
Output: F (fitness value)

Step 1: For each $\boldsymbol{x} \in \boldsymbol{P}_t$ calculate its rank by counting the number of solutions that dominates $\boldsymbol{x}$: $r(\boldsymbol{x}) = 1 + |\{\boldsymbol{y} | \boldsymbol{y} \in \boldsymbol{P}_t \wedge \boldsymbol{y} \prec \boldsymbol{x}\}|$. The symbol | | denotes the cardinality of a set and $\prec$ denotes the Pareto dominance relation. In this case, $\boldsymbol{y} \prec \boldsymbol{x}$ means $\boldsymbol{y}$ dominates $\boldsymbol{x}$.

Step 2: Sort a population according to the ranking. Assign each $\boldsymbol{x} \in \boldsymbol{P}_t$ a raw fitness $F'(\boldsymbol{x})$ by interpolating from the best ($r(\boldsymbol{x})$=1) to the worst individual ($r(\boldsymbol{x}) \leq N$); linear ranking is used.

Step 3: Calculate fitness values $F(\boldsymbol{x})$ by averaging and sharing the raw fitness values $F'(\boldsymbol{x})$ among individuals $\boldsymbol{x} \in \boldsymbol{P}_t$ having identical ranks $r(\boldsymbol{x})$

MOGA is a good approach, efficient and relatively easy to implement but is highly dependent on an appropriate selection of σ_{share} (Coello 2000). Fitness sharing and niching are performed in the objective space, which makes the MOGA applicable to other optimization problems such as combinatorial optimization problems but the shared fitness computation does not assure that a solution in a poorer rank will always have a worse scaled fitness than every solution in a higher rank (Deb 2001). This may introduce bias towards solutions

that may evolve from only one region of the trade-off surface. This may slow down convergence to the Pareto-optimal front in higher rank solutions.

NONDOMINATED SORTING ALGORITHM

Srinivas and Deb (1994) implemented Goldberg's (1989) idea of a nondominated sorting genetic algorithm (NSGA) that favors diverse nondominated solutions by using a sharing strategy similar to MOGA. The advantage of NSGA over MOGA is that it avoids the problem of slow convergence and poor spread of better ranked solutions in the trade-off front by assigning values front-wise i.e. solutions in a better front are assigned larger shared fitness values. The first step in NSGA is to sort the population $\boldsymbol{P}_t$ according to nondomination, which produces several mutually exclusive nondominated sets or classes. The fitness assignment begins from the first class (best nondominated set) and proceeds to the other dominated sets. The individuals of the first class are assigned fitness values equal to N and subsequently their shared fitness values are computed. The minimum shared fitness value is then used to compute the shared fitness values of the next dominated set and this process continues until all fitness values of the remaining dominates have been calculated. The computational complexity of NSGA is $O(mN^2)$.

NSGA selection

Input:	$\boldsymbol{P}_t$	(population of N individuals)
	σ_{share}	(niche radius)
Output:	$\boldsymbol{P}_s$	(nondominated set)

Step 1: Set an initial population $\boldsymbol{P}_r = \boldsymbol{P}_t$ and initialize the minimum fitness value $F_{min}= N$. Set s = 1.

Step 2: Determine the set $\boldsymbol{P}_s$ of individuals in $\boldsymbol{P}_r$ whose decision vectors are nondominated regarding $g(\boldsymbol{P}_r)$. Remove the members of $\boldsymbol{P}_s$ from $\boldsymbol{P}_r$ i.e. $\boldsymbol{P}_r= \boldsymbol{P}_r - \boldsymbol{P}_s$ (multiset subtraction).

Step 3: Set the raw fitness of individuals in $\boldsymbol{P}_s$ to F_{min} and perform fitness sharing in the decision space within $\boldsymbol{P}_s$ only.

Step 4: Decrease the minimum fitness value F_{min} such that it is lower than the smallest shared fitness in $\boldsymbol{P}_s$:$0<F_{min} <\min\{F(\boldsymbol{x}) \mid \boldsymbol{x}\in \boldsymbol{P}_s\}$. Set $s = s + 1$.

Step 5: If $\boldsymbol{P}_r \neq \varnothing$ then go to step 2 else stop.

The sharing in NSGA is done in the decision values and not in the objective values, which ensures a better distribution of solutions in the trade-off front. However, this technique is less efficient (computationally) than MOGA, and is more sensitive to the sharing function σ_{share} (Srinivas and Deb 1994).

NICHED PARETO GENETIC ALGORITHM

Horn and Nafpliotis (1993) introduced the niched Pareto genetic algorithm (NPGA) that differed from previous Pareto-dominance methods by using binary tournament selection rather than proportionate selection. In a NPGA, there is no need to compute a fitness value for each solution. The selection procedure favors nondominated solutions ($\boldsymbol{P}_{dom}$) and if dominance cannot be established, niching and fitness sharing is performed. Solutions with lower niche counts win, which means that parents in less crowded regions are chosen in the offspring population. The computational complexity of NPGA is $O(mN^2)$.

NPGA selection

Input:	$\boldsymbol{P}_t$	(population of N individuals)
	$\boldsymbol{P}_{dom}$	(subpopulation t_{dom} individuals)
		t_{dom} is the domination pressure
Output:	$\boldsymbol{P}$'	(mating pool)

Step 1: Set i = 1 and mating pool $\boldsymbol{P}$' = $\varnothing$.

Step 2: Select two competitors $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{P}_t$ and a comparison set $\boldsymbol{P}_{dom} \subseteq \boldsymbol{P}_t$ of t_{dom} individuals at random without replacement.

Step 3: If $g(\boldsymbol{x})$ is nondominated regarding $g(\boldsymbol{P}_{dom})$ and $g(\boldsymbol{y})$ is dominated, then $\boldsymbol{x}$ is the winner of the tournament: $\boldsymbol{P}' = \boldsymbol{P}' + \{\boldsymbol{x}\}$. Else if $g(\boldsymbol{y})$ is nondominated regarding $g(\boldsymbol{P}_{dom})$ and $g(\boldsymbol{x})$ is dominated then $\boldsymbol{y}$ is the winner of the tournament: $\boldsymbol{P}' = \boldsymbol{P}' + \{\boldsymbol{y}\}$.

Step 4: Else decide tournament by fitness sharing: calculate the number of individuals in the partially filled mating pool that are in σ_{share}-distance to $\boldsymbol{x}$: $n(\boldsymbol{x}) = |\{\boldsymbol{k} | \boldsymbol{k} \in \boldsymbol{P}' \wedge d(\boldsymbol{x}, \boldsymbol{k}) < \sigma_{share}\}|$.
Do the same for $\boldsymbol{y}$.
If $(n(\boldsymbol{x}) < n(\boldsymbol{y}))$ then $\boldsymbol{P}' = \boldsymbol{P}' + \{\boldsymbol{x}\}$ else $\boldsymbol{P}' = \boldsymbol{P}' + \{\boldsymbol{y}\}$.

Step 5: Set $i = i + 1$. If $(i \leq N)$ then go to Step 2 else stop.

The basic advantage of this approach is it does not require any fitness assignment, which removes any bias in the fitness assignment procedure. Since this approach does not apply Pareto selection to the entire population but to a subpopulation in each run, the technique is very fast and produces good nondominated runs that can be kept for a large number of generations. However, the approach requires an appropriate value for the sharing factor σ_{share} and a good choice of the value t_{dom} in order to perform well. The dependence of NPGA to the two variables σ_{share} and t_{dom} complicates its appropriate use in practice (Coello 2000).

2.3.2 Modern implementations of an MOEA

STRENGTH PARETO EVOLUTIONARY ALGORITHM

Zitzler et al. (2002) introduced the Strength Pareto Evolutionary Algorithm 2 (SPEA2), which is an extension and improvement of the original work by Zitzler and Thiele (1999). SPEA2 integrates a fitness assignment strategy, which considers the number of individuals that an individual dominates and the number of its dominators. It uses a nearest-neighbor density estimation technique that guides the search more efficiently and avoids the formation of new solutions in only a few clusters in the search space. SPEA2 has a truncation procedure that preserves the best solutions when the number of nondominated individuals exceeds the external population size.

SPEA2 first assigns a strength value $S(\boldsymbol{x})$, to each individual $\boldsymbol{x}$ from the archive ($\overline{N}$) and population size (N) representing the number of solutions $\boldsymbol{x}$ dominates. Then the raw fitness $R(\boldsymbol{x})$, which measures the strength of $\boldsymbol{x}$'s dominators of each solution $\boldsymbol{x}$, is calculated. The raw fitness acts as a niching mechanism but poorly performs when most paths in $N+\overline{N}$ are nondominated, i.e. the population forms new solutions in only a few clusters, in effect compromising exploration of the search space. Genetic drift is the term for this phenomenon. SPEA2 introduces a density estimator, a fitness sharing mechanism to avoid genetic drift. The density estimator is the inverse of the distance of an individual in objective space to the k-th nearest neighbor. The density value and the raw fitness value are combined to give the final fitness function. SPEA2 offers two selection procedures: environmental and mating selection.

The environmental selection is concerned with choosing individuals that will have to move on to the next generation archive from the current archive $\overline{\boldsymbol{P}}_t$ and population $\boldsymbol{P}_t$. SPEA2 maintains an archive $\overline{\boldsymbol{P}}_t$ in each generation and is composed of the "best" individuals of a fixed size $\overline{N}$, which is equal to the population size N. Two usual situations may occur in selection. First, the number of nondominated solutions in $\overline{\boldsymbol{P}}_{t+1}$ is less than $\overline{N}$. SPEA2 resolves this by adding the "best" dominated individuals from $\overline{\boldsymbol{P}}_t + \boldsymbol{P}_t$ to $\overline{\boldsymbol{P}}_{t+1}$. Second, the number of nondominated solutions for the next generation is greater than $\overline{N}$. SPEA2 uses a truncation procedure whereby the individual with the minimum distance to another individual is truncated until $|\overline{\boldsymbol{P}}_{t+1}| = \overline{N}$. SPEA2 implements binary tournament selection with replacement to fill in the mating pool. This type of mating selects two solutions at a time in each tournament. Their fitness values are evaluated and the better solution is placed in the mating pool. The runtime complexity of SPEA2 is $O(mN^2 \log N)$.

SPEA2 fitness assignment

Input:	$\boldsymbol{P}_t$	(population with N individuals)
Output:	F	(fitness values)

Step 1: Calculate the strength values of individuals in $\boldsymbol{P}_t$ and $\overline{\boldsymbol{P}}_t$. The strength of individual $\boldsymbol{x}$ is computed as

$$S(\boldsymbol{x}) = |\{\boldsymbol{y} \mid \boldsymbol{y} \in \boldsymbol{P}_t + \overline{\boldsymbol{P}}_t \wedge \boldsymbol{x} \prec \boldsymbol{y}\}|$$

Where the symbol $|\ |$ denotes the cardinality of a set, + stands for multiset union and the symbol $\prec$ corresponds to the Pareto dominance relation extended to individuals i.e., the term $\boldsymbol{x} \prec \boldsymbol{y}$ means $\boldsymbol{x}$ dominates $\boldsymbol{y}$.

Step 2: Compute the raw fitness

$$R(\boldsymbol{x}) = \sum_{\boldsymbol{y} \in \boldsymbol{P}_t + \overline{\boldsymbol{P}}_t, \boldsymbol{y} \prec \boldsymbol{x}} S(\boldsymbol{y})$$

For each individual $\boldsymbol{x}$ the distances (in objective space) to all individuals $\boldsymbol{y}$ in archive and population are calculated and stored in a list. After sorting the list in increasing order, the *k-th* element gives the distance sought, denoted as σ_x^k.

Step 3: Compute the density of each individual $\boldsymbol{x}$

$$D(\boldsymbol{x}) = \frac{1}{\sigma_x^k + 2}; k = \sqrt{N + \overline{N}}$$

Step 4: Compute the fitness values

$$F(\boldsymbol{x}) = R(\boldsymbol{x}) + D(\boldsymbol{x})$$

Since SPEA2 is an elite-preserving evolutionary algorithm, it preserves the good solutions of a population by directly carrying them over to the next generation. It makes sure that the fitness of the population does not deteriorate by storing the best solutions in an archive. It also ensures a good spread of Pareto-optimal solutions and prevents boundary solutions from removal in the population by introducing an enhanced clustering technique and a truncation method. A more detailed discussion of SPEA2 is found in Appendix B.

ELITIST NONDOMINATED SORTING GA

Deb et al. (2002) introduced an elitist nondominated genetic algorithm (NSGA-II) that uses not only an elite-preserving strategy but also an explicit-diversity preserving mechanism. Initially, NSGA-II creates a random parent population $\boldsymbol{P}_0$ $_s$orts the population based on nondomination and assigns each solution a fitness value equal to its nondomination level (as in NSGA). Thereafter NSGA-II creates an offspring population $\boldsymbol{Q}_0$ of size N by binary tournament selection and recombination operators. After the initial populations are created they are combined in one population $\boldsymbol{R}_t$ of size $2N$. Then, the population is sorted according to nondomination - solutions belonging to the best nondominated set $\boldsymbol{F}_1$ must be favored more than any other solution in the combined population. If the size of $\boldsymbol{F}_1$ is smaller then *N*, then all members of the set are chosen for the new population $\boldsymbol{P}_{t+1.}$ The new population $\boldsymbol{P}_{t+1}$ are then filled with members from subsequent nondominated fronts $\boldsymbol{F}_{2,}$ $\boldsymbol{F}_{3,\ldots,}\boldsymbol{F}_n$ in the order of their ranking. This procedure continues until no more sets can be accommodated in $\boldsymbol{P}_{t+1}$. The new population $\boldsymbol{P}_{t+1}$ is used for selection, crossover, and mutation to create a new offspring population $\boldsymbol{Q}_{t+1}$ of size *N*. The selection criterion, which requires both the rank and the crowded distance of each solution in the population, is dependent on the crowded-comparison operator. The computational complexity of NSGA-II is $O(mN^{2})$.

NSGA-II selection procedure

Input:	$\boldsymbol{P}_t$	(population with N individuals)
Output:	$\boldsymbol{Q}_t$	(offspring population with N individuals)

Let the symbol $|\ |$ denote the cardinality of a set, + stands for multiset union.

Step 1: Create a new population $\boldsymbol{R}_t = \boldsymbol{P}_t + \boldsymbol{Q}_t$.
Step 2: Perform nondominated sorting to $\boldsymbol{R}_t$ and generate different fronts $\boldsymbol{F}_i$
Step 3: Set counter i = 1; the new population $\boldsymbol{P}_{t+1} = \varnothing$.
While $|\boldsymbol{P}_{t+1}| + |\boldsymbol{F}_i| \leq N$ do
$\boldsymbol{P}_{t+1} = \boldsymbol{P}_{t+1} + \boldsymbol{F}_i$
$i = i + 1$
Step 4: Perform crowding-sort algorithm and include most widely spread solutions using crowding distance values in $\boldsymbol{F}_i$ to $\boldsymbol{P}_{t+1}$.
Step 5: Create $\boldsymbol{Q}_{t+1}$ from $\boldsymbol{P}_{t+1}$ by using crowded tournament selection and recombination operators

The crowding comparison algorithm of NSGA-II eliminates the need for a niching parameter such as σ_{share} by allowing solutions to compete using their crowding distances (Deb 2001). Removing the niching parameter also removes the problems associated with it such as the proper estimation of its value and its influence of the search process. A more detailed discussion of NSGA-II is found in Appendix B.

PARETO-ARCHIVED EVOLUTIONARY STRATEGY

Knowles and Corne (2000a) introduced the Pareto-Archived Evolutionary Strategy (PAES) using a (1+1)-ES. It comprises of three parts: the candidate solution generator, the acceptance function, and the nondominated solutions (NDS) archive. Viewed in this way, (1+1)-PAES represents the simplest non-trivial approach to a multiobjective local search procedure. The candidate solution generator is similar to simple random mutation i.e., it maintains a single current solution then produces a single new offspring via random mutation. In the simple (1+1)-PAES, the *test*($\boldsymbol{x}_t, \boldsymbol{y}_t, \overline{\boldsymbol{P}}_t$) function is a density-based procedure that determines whether the offspring solution should be rejected or accepted and if it should be archived or not. In the density calculation, each objective is divided into 2^d divisions where d is a user-defined parameter. The archived solutions are placed in $(2^d)^m$ hypercubes according to their location in the objective space and the number of solutions in each hypercube is counted. If the offspring is in a less crowded hypercube than its parent then the offspring

becomes the parent of the next generation otherwise the parent continues to the next generation. Similarly, solutions residing in the least crowded areas get preference to the archive. The complexity of (1+1)-PAES is $O(mN^2)$.

(1+1) -PAES procedure

Input: $\boldsymbol{x}_t$ (a random solution)
Output: $\overline{\boldsymbol{P}}_t$ (nondominated set)

Step 1: Generate an initial random solution $\boldsymbol{x}_t$ and add it to the archive $\overline{\boldsymbol{P}}_t$ of size N
Step 2: Mutate $\boldsymbol{x}_t$ to generate an offspring $\boldsymbol{y}_t$
Step 3: If ($\boldsymbol{x}_t \prec \boldsymbol{y}_t$) discard $\boldsymbol{y}_t$
else if ($\boldsymbol{y}_t \prec \boldsymbol{x}_t$) then $\boldsymbol{x}_t = \boldsymbol{y}_t$ and $\overline{\boldsymbol{P}}_t = \overline{\boldsymbol{P}}_t + \{\boldsymbol{y}\}$
else if ($\boldsymbol{z}_t \prec \boldsymbol{y}_t$), $\forall z \in \overline{\boldsymbol{P}}_t$, then discard $\boldsymbol{y}_t$
else apply *test*($\boldsymbol{x}_t$, $\boldsymbol{y}_t$, $\overline{\boldsymbol{P}}_t$) to determine which solution becomes the current solution or if $\boldsymbol{y}_t$ should be added to $\overline{\boldsymbol{P}}_t$
Step 4: If the termination criterion is reached then stop
else $t = t + 1$ and go to step 2

The (1+1)-PAES serves as a good, simple baseline algorithm for multiobjective optimization. Its performance is strong, especially given its low computational complexity, even on demanding tasks where one might expect local search methods to be at a disadvantage (Knowles and Corne 2000a). However, appropriate values for the archive size N and the depth parameter d are necessary in order to find a good set of nondominated and well-spread solutions.

Knowles and Corne (2000a) extended the (1+1)-PAES to a (1+λ)-PAES and a (μ+λ)-PAES. In the multimember (1+λ)-PAES, a parent solution is mutated λ times and an offspring is created each time. A fitness value is assigned to each offspring and compared to the archive and its hypercube location. The fittest among the parent and offspring solutions becomes the parent of the next generation. In a multimember (μ+λ)-PAES, each μ parent and each λ offspring are compared with the archive and an offspring membership is calculated as in (1+1)-PAES. The new parent population is generated based on a dominance score from the current μ parent population and λ offspring population. The proponents of PAES observed that the multimember versions of PAES do not generally perform as well as the (1+1)-PAES since offspring are compared only

with the archive and not against other offspring. This means that there is no assurance that best offspring solutions are exploited.

Knowles and Corne (2000b) developed a memetic algorithm for multiobjective optimization M-PAES, which uses the local search method of (1+1)-PAES, combined it with the use of a population and crossover. The usefulness of M-PAES was evaluated on a set of multiobjective 0/1 knapsack problems. Their results showed that M-PAES performs better than (1+1)-PAES on all problems and, compared with the SPEA, the performance of M-PAES is similar with SPEA and gives a near-best performance.

PARETO ENVELOPE-BASED SELECTION ALGORITHM

Corne et al. (2000) introduced a Pareto Envelope-based Selection Algorithm (PESA) wherein a simple hypercube scheme controls selection and diversity maintenance. The selection of a parent in PESA is dependent on the degree of crowding in the different regions of the archive. The crowding strategy divides the objective space into hypercubes. Each chromosome in the archive is assigned a *squeeze factor*, which is the total number of other chromosomes in the archive that are located in the same hyper-box. The lower the *squeeze factor* the higher the fitness of an individual. The *squeeze factor* is also used to update the archive i.e., the individual with the highest *squeeze factor* is removed when the archive exceeds in allowable number of members. The run-time complexity of PESA is $O(mN^2)$.

PESA procedure

Input: $\boldsymbol{P}_t$ (population with N individuals)
Output: $\boldsymbol{P}^*$ (Nondominated set)

Step1: Generate and evaluate N solutions in $\boldsymbol{P}_t$. Divide the normalized objective space into n^m hypercubes where n is the number of grids along a single objective axis and m is the number of objectives. Set $\overline{\boldsymbol{P}}_t = \varnothing$

Step 2: Incorporate the nondominated members of $\boldsymbol{P}_t$ into $\overline{\boldsymbol{P}}_t$

Step 3: If a termination criterion has been reached, $\boldsymbol{P}^* = \overline{\boldsymbol{P}}_t$ and stop else $\boldsymbol{P}_t = \varnothing$. Do while ($\boldsymbol{P}_t$ is not full)

with probability p_c
select 2 parents from $\overline{\boldsymbol{P}}_t$
generate a single offspring via crossover
mutate the child and add to $\boldsymbol{P}_{t+1}$
with probability (1- p_c) select one parent, mutate it to generate a child,
add child to $\boldsymbol{P}_{t+1}$

Step 4: go to Step 2

PESA is easy to implement and computationally efficient. However, its performance depends on the cell sizes (hyper-box) and prior information is needed about the objective space. Corne et al. (2001) described a region-based selection procedure (PESA-II). In the selection step, cells are selected instead of individuals and a cell that is sparsely occupied has a higher chance to be selected than a crowded cell. Once a cell is selected, solutions within the cell are randomly chosen to participate to crossover and mutation. Preliminary investigations by the researchers (Corne et al. 2001) suggest that the PESA-II results are not overly sensitive to the cell size or hyper-box dimension parameter, but they state that more investigation is required to determine if this is generally the case.

2.3.3 Comparative studies of MOEAs

A comprehensive discussion of multiobjective evolutionary algorithms (MOEA) can be found in Deb (2001). In addition, Coello (2000) gives a summary of current approaches in MOEA and emphasizes the importance of new approaches in exploiting the capabilities of evolutionary algorithms in multiobjective optimization. Zitzler et al. (1999) performed a comparative analysis of existing evolutionary algorithms in multiobjective optimization by means of well-defined quantitative performance measures. They show that elitism is an important factor in evolutionary multiobjective search. The following discussion presents a summary of relevant comparative studies of multiobjective evolutionary algorithms published from the year 2000 and onwards.

Zitzler et al. (2000) compared several multiobjective EAs on six different test functions and found the following: (1) multimodality causes the most difficulty for evolutionary approaches. However, non-convexity remains a problem for

weighted-sum based algorithms; (2) for the chosen test problems and parameter settings, a clear hierarchy of algorithms emerges regarding the distance to the Pareto-optimal front in descending order of merit: SPEA, NSGA, VEGA, WBGA, NPGA, and MOGA; (3) elitism is an important factor in evolutionary multiobjective optimization.

Zitzler et al. (2002) presented an improved version SPEA2, which uses an enhanced fitness assignment strategy and a new technique for archive truncation and density-based selection. The study compared SPEA2 with SPEA, PESA and NSGA-II on 16 continuous and combinatorial test problems. The results of the analysis were: (1) SPEA2 performs better than its predecessor SPEA on all problems; (2) PESA has the fastest convergence, which is probably due to its higher elitism intensity. However, it has difficulties keeping the boundary solutions on some problems; (3) SPEA2 and NSGA-II show the best performance overall. (4) SPEA2 seems to have advantages over PESA and NSGA-II in higher dimensional objective spaces wherein the number of nondominated solutions increases rapidly.

Deb et al. (2002) tested NSGA-II on nine difficult test problems and found that NSGA-II was able to maintain a better spread of solutions and converge better in the obtained nondominated front compared to PAES and SPEA. The diversity preserving mechanism used in NSGA-II was found to be the best among the three approaches.

Tan and Lee (2002) surveyed existing multiobjective evolutionary algorithms according to their performance on four benchmark problems. Besides considering the spread of solutions across the Pareto-optimal front and the ability to attain the global trade-offs, the uniform distribution of individuals along the Pareto-front, the computational effort, the robustness to disturbances, and the average best performance of tracking optimal regions in changing environments were evaluated. They conclude that no single algorithm excels in all performance measures. Furthermore, elitism and a sharing strategy are important for good convergence and population distribution along the discovered tradeoffs in multiobjective optimization.

Yen and Lu (2003) proposed a Rank-Density-based algorithm (RDGA) that simplifies the problem domain by converting high-dimensional multiple objectives into two objectives. Their results showed that RDGA produced statistically competitive results with the four state-of-the-art MOEAs, MOGA,

PAES, NSGA-II, and SPEA II on four types of multiobjective problems. The MOOPs were designed to exploit various complications in finding near-optimal, near-complete, and uniformly distributed Pareto-optimal fronts. RDGA was found to show better performance in keeping the diversity of the individuals along the current tradeoff surface, extending the Pareto front to new areas, and finding a well-approximated, nondominated set. However, the paper is far from representing a complete MOOP test suite to conclude that RDGA is a better algorithm than other modern MOEAs.

Yen and Lu (2003) proposed a Dynamic Multiobjective Evolutionary Algorithm (DMOEA) that simplifies computational complexity by using a *cell-based rank* and *density- fitness* estimation scheme, an *objective compression* strategy, and an adaptive-population size feature. A cell-based ranking scheme first divides the objective space into cells or hypercubes (e.g. PAES and RDGA) and ranks are given to the cells and not individually to the nondominated solutions. An objective compression strategy enhances the cell-based ranking scheme by compressing the size of the cells in the objective space. Its effects refine the Pareto front and reduce cell density. Their comparative study showed that DMOEA produces statistically competitive or even superior results with the other modern MOEAs such as PAES, NSGA-II, RDGA, and SPEA II on three multiobjective optimization benchmark problems. The problems are designed to exploit various complications in finding near-optimal, well-extended, and uniformly distributed Pareto-optimal fronts. They conclude that DMOEA can be a potential candidate in solving time-critical or on-line MOOPs due to its lower computational complexity. However, as in their study of RDGA, the test functions do not cover all challenging characteristics of MOOPs.

Bosman and Thierens (2003) argued that the quest for finding the components that result in the best EAs for multiobjective optimization is not likely to converge to a single, specific MOEA. They stated that the tradeoff between the goals of proximity and diversity preservation plays an important role in the exploitation and exploration phases of any MOEA.

Deb et al. (2005) introduced an ε-MOEA which was developed using the ε-dominance criterion by Laumanns et al. (2002). They found that the ε-MOEA was successful in finding well-converged and well-distributed solutions with a much smaller computational effort than a number of state-of-the-art MOEAs including NSGA-II, SPEA2, and PESA on 12 test problems. They suggest the use of the ε-MOEA to more complex and real-world problems due to its consistency

in achieving convergence and diversity of solutions over multiple simulation runs with less computational effort compared to other MOEAs.

Chaiyaratana et al. (2007) proposed a modified multiobjective diversity control oriented genetic algorithm or MODCGA-II, which is an improvement of its predecessor MODCGA. Their analysis on six benchmark problems described in Deb et al. (2005) indicated that the MODCGA-II produces nondominated solutions that are better than NSGA-II and SPEA-II when the number of objectives is limited to two but performs worse when the number of objectives increases to three. They recommended that their technique is the most suitable approach for both single-objective genetic algorithm and multiobjective genetic algorithm in case the number of objectives is two.

Goh and Tan (2007) performed extensive studies to examine the impact of noisy environments in evolutionary multiobjective optimization, particularly for the population dynamics of fitness and diversity. They introduced three noise-handling features that include an Experiential Learning Directed Perturbation (ELDP) operator that adapts the magnitude and direction of variation according to previous experiences for fast convergence, a Gene Adaptation Selection Strategy (GASS) that helps the evolutionary search in escaping from local optima, and an archiving model based on the concept of possibility and necessity measures. The comparative study showed that the basic algorithm incorporating the proposed features exhibits competitive or superior performance in terms of proximity, diversity, and distribution for both the noiseless and noisy benchmark problems. They found that existing MOEAs such as SPEA2 and NSGAII enhanced with the proposed features of GASS and ELDP are capable of giving better convergence and population diversity along the global tradeoff for the benchmark problems with and without the presence of noise.

The studies presented above show a variety of results and no single MOEA performs better in the different performance metrics but most of the studies compare their algorithms with either NSGA-II or SPEA2 or both. The studies above mostly evaluated the performance of the selection operators of each MOEA without investigating the effect of the parameter settings on its performance. The current study differs from the researches above. The current study does not create a new selection operator and compare its performance with well-known algorithms but investigates the effect of the parameter settings

and variation operators on the performance of NSGA-II and SPEA2 in selected multiobjective optimization problems.

Chapter 3

The Competitive Facility Location Problem: An optimization problem with a fixed-length string of continuous variables

3.1 Introduction

Facility location is the process of determining a geographic site for operations of a company or any organization in general. Managers of both service and manufacturing organizations must weigh several factors when assessing the desirability of a particular site, including proximity to customers and suppliers, labor costs, and transportation costs. Facility location is often determined by one critical factor. In the case of plant or warehouse location, economic factors usually are dominant.

A location model is said to be *competitive* when the problem of locating a new facility in the market incorporates the existence of other facilities and that the new facility has to compete for its market share (Plastria 2001). A number of facilities with known fixed locations exist in the market. That is, Competitive Facility Location (CFL) models describe how facilities capture their market share

and where a new facility should be located to maximize its market share. The models mostly start from a measure of *attraction* that a customer feels for a facility. The attraction is determined by several factors, but most CFL models represent market share as a function of the *distance* between the customer and the facility on the one hand, and on the other hand on internal characteristics of the facility, which generally can be called the *quality* of a facility. Various types of attraction functions might be formulated, but the study will deal with a multiplicative type, which leads to a gravity type attraction, given by the quality divided by some strictly positive power of the distance. Anyway, the function should be non-increasing with distance and non-decreasing with quality. The models take care of two decisions: the location of the facility and its design. The location of the site influence the distance part of the objective and the design relates to the quality. Quality is determined by a mixture of attributes of the facility like floor area, number of check-counters, product mix and price level. Raising the level of these attributes of quality involves a higher cost. The main objective of the decision is to maximize profit, which can be expressed as a function of sales and cost, like sales minus cost or sales divided by cost. The link to this chapter is made due to the results obtained by Carrizosa and Plastria (1995) who show that profit-maximization with respect to both the location and the quality of the new facility can be obtained by inspecting only a finite number of solutions. The solutions are obtained after solving a bi-objective optimization problem of finding efficient solutions that maximize "captured" consumers, while minimizing the quality costs of the new facility. The type of problem is known as a maxcovering-minquantile location problem, which arediscussed further on.

3.1.1 Problem Definition

The competitive location model involving bi-objective maximization of location and quality of a facility is due to Plastria and Carrizosa (2004). They present a general profit maximizing competitive location model with different attraction types on a consumer and limit the location of the new facility within a bounded area. Plastria and Carrizosa (2004) show that a maximal profit solution to the CFL problem through the determination of efficient solutions reduces to the bi-objective optimization of

Min	α
Max	$CW(\alpha, \boldsymbol{x})$; $\alpha \geq \alpha_0$, $\boldsymbol{x} \in \boldsymbol{S}$

where α = unknown quality cost of the new facility at an unknown site $\boldsymbol{x}$

$CW(\alpha, \boldsymbol{x})$ = captured weight of the new facility

α_0 = minimal quality cost ($\alpha_0 > 0$)

$\boldsymbol{S}$ = closed set in the plane

A finite set of *customers* is denoted by AF. Each customer $a \in AF$ has a known location $\boldsymbol{x}_a$ and a strictly positive weight w_a representing his buying power. A finite set of *competing facilities* with which the new facility is to compete is denoted by CF. Competing facility $f \in CF$ is located at site $\boldsymbol{x}_f$ and has a quality α_f considered to be known and fixed. Any customer $a \in AF$ feels an attraction $attr(\alpha,f)$ towards a facility f at $\boldsymbol{x}_f$, which depends on factors such as distance from $\boldsymbol{x}_a$ to $\boldsymbol{x}_f$ or other factors like tradition etc. Consider a new facility with unknown site $\boldsymbol{x}$ and unknown quality α, of at least some minimal quality $\alpha_0 > 0$. Its attraction on a customer $a \in AF$ can be expressed by a function $A_a(a,dist_a(\boldsymbol{x}))$, a function of the quality α and of the distance $dist_a(\boldsymbol{x})$ from the customer to the facility. The function A_a is defined on $[\alpha_0,+\infty[\times[0,+\infty[\rightarrow [0,+\infty]$. A typical example is attraction of *gravity type*, given by

$$A_a(\alpha,\boldsymbol{x}) = \frac{k_a \alpha}{\|\boldsymbol{x}-\boldsymbol{x}_a\|^p} \tag{3.1}$$

where p is any strictly positive exponent, and $k_a > 0$ represent some proportionality constant depending on a. In pure gravity models $p = 2$, which may appear in a wide variety of physical contexts. But also linear markets may be assumed (see Eiselt and Laporte, 1988 and 1989). A common feature of all attraction functions is that a larger quality cost increases the attraction of a customer to a facility whereas a larger distance between them decreases it. With a deterministic customer choice rule, the facility captures those customers attracted more to the new facility than to any competing facility in CF. The set of captured customers is given by:

$$Capt(\alpha,\boldsymbol{x}) = \left\{ a \in AF \,\middle|\, \forall f \in CF : A_a(\alpha, dist_a(\boldsymbol{x})) \geq attr(a,f) \right\} \tag{3.2}$$

The total weight captured by the new facility is given by

$$CW(\alpha,\boldsymbol{x}) = \sum_{a \in Capt(\alpha,x)} w_a \tag{3.3}$$

Profit is expressed by a *profit-indicator* function π which is strictly increasing in sales income $\sigma(CW(\alpha,\boldsymbol{x}))$ and strictly decreasing in operating costs $\gamma(\alpha)$

$$\prod(\alpha,\boldsymbol{x}) = \pi(\sigma(CW(\alpha,\boldsymbol{x})),\gamma(\alpha)) \tag{3.4}$$

It is the profit-indicator $\prod$ that needs to be optimised by an adequate choice of both the quality $a > a_0$ and the site x within some set of feasible sites $\boldsymbol{S}$:

$$max\left\{\prod(\alpha,x)\middle|\boldsymbol{x}\in\boldsymbol{S},\alpha\geq\alpha_0\right\} \tag{3.5}$$

Given a fixed quality a, profit maximisation is achieved by maximisation of the total captured weight. This optimisation problem is called a *maximal covering problem* as studied in the planar context by Drezner (1981). Drezner's objective is to maximise a weighted number of demand points (read: customers) within a given (Euclidean) distance from the facility. This constraint might be motivated as follows: the location of an emergency facility may be satisfying a bound on the distance between each demand point and the new facility, or when a shopping centre is planned, only customers within a given distance use its services. Drezner (1981) presents an algorithms of complexity $O(n^2 \log n)$ where n is the number of demand points. Since, in this problem, α is a decision variable, such a maximal covering problem needs to be solved for each possible value of α, given that only a finite number of feasible quality values are available.

When, however, optimisation needs to be done over the full range of positive α-values (with $\alpha >= \alpha_0$), another approach is to be advised. In Carrizosa and Plastria (1995) it is shown how the optimisation problem can be solved by reducing it to a bi-objective problem. In some cases the optimisation problem can be solved by inspecting a finite number of points, polynomial in the cardinality of AF. They have proven that, if maximal profit solutions exist, they are all efficient (or nondominated, or Pareto-optimal) for the bi-objective problem. The authors call the problem a minquantile/maxcovering bi-objective problem.

The cases in which the optimisation problem can be solved analytically depend on (1) the type of attraction function; (2) the characteristics of the subset; and (3) the characteristics of the search space. In terms of attraction function a

limited number of functions are feasible, including the gravity-type functions. In terms of the characteristics of the subset, the subset *s* needs to be a closed convex subset. In terms of the search space, solutions need to be found in the plane $\Re^2$, in which distances are measured by a norm. If the requirements are met then Carrizosa and Plastria (1995) state that efficient solutions to the bi-objective problem are optimal solutions to single-objective problems, i.e. single-facility minmax location problems. To explain this procedure a bit further, Theorem 8 of the article by Plastria and Carrizosa (2004) is stated as follows:

"**Theorem 8:** *Let* $(\alpha^*, \boldsymbol{x}^*)$ *be an efficient solution for the bi-objective problem. Then, one has:*

1. *When Capt*$(\alpha^*, x^*) \neq \phi$ *there exists a nonempty subset* $T \subset AF$*, with cardinality at most* 3 *such that*
   (a) $\boldsymbol{x}^*$ *solves the generalizes single-facility minmax location problem*
   $$\min_{x \in S} \max_{a \in T} D_\alpha(\boldsymbol{x})$$
   with $D_a(\boldsymbol{x})$*:* $\Re^2 \rightarrow [0, +\infty]$ *defined as* D_a*:* $x \rightarrow B_a(dist_a(\boldsymbol{x}))$ *in which* B_a *indicates, as a function of the distance d, the quality threshold above which a customer a at distance d is captured.*
   (b) α *is the optimal value of* (a)*, which is finite*
2. *In case Capt*$(\alpha, \boldsymbol{x}^*) = \phi$*, one must have* $\alpha^* = \alpha_0$*, and any other pair* $(\alpha_0, \boldsymbol{x})$ *is then also efficient for the bi-objective problem.* "

After finding, in (b) the optimal value α_T of each of the $O(n^3)$ optimisation problems in (a), and finding the set S_T of optimal solutions for the problem in (a), a list of pairs $(\alpha_T, \boldsymbol{x}_T)$ is available which contains the set of efficient solutions for the bi-objective problem.

3.1.2 Literature Review

A number of models exist that solve different competitive facility location problems. A survey of such models is discussed by Eiselt et al. (1993). The authors suggest fives major components in competitive facility location, which are space, number of players, pricing policy, rules of the game, and behavior of customers. The mostly used metric space in competitive facility location is the m-dimensional real space $\Re^m$ and, in most instances, is bounded by a convex polygon denoted by $|\Re^m|$. A pricing policy distinguishes between models with both price and location as decision variables and models that consider location

as the only decision variable. Rules of the game define the general concept of equilibrium as applied to games but most competitive facility location studies in operations research do not fall under equilibrium models because competing facilities already exist. The last component, behavior of customers, defines deterministic and probabilistic models. In a deterministic model, customers always patronize a single facility that they are most attracted to. On the other hand, probabilistic models assign probabilities of customer attraction to each facility.

Drezner (1994) proposes a solution for the location of a new facility in a continuous planar space and in an environment where competing facilities have different levels of attraction on a consumer. Drezner's (1994) algorithm solves a single-objective optimization problem that first calculates a "break-even" distance, which is the maximum distance that a consumer is willing to travel to the new facility. Afterwards, the market share is computed for the new facility relative to the break-even distance. She concludes that her model guarantees a superior location based on information about consumer preferences and facility attributes. Plastria and Carrizosa (2004) observes that most algorithms solve competitive facility location problems by either setting a fixed site and attempt to maximize profit by an adequate choice of quality or setting the quality fixed and then find an optimal site. They show that profit maximization with respect to both location and the quality of the new facility may be obtained by solving efficient solutions (Pareto-optimal) in a bi-objective location problem.

Pangilinan et al. (2005) applies an MOEA for finding efficient solutions to the CFL problem and notes that the MOEA generates inferior solutions to the deterministic procedure by Plastria and Carrizosa (2004). However, they show that the MOEA is more scalable in terms of computational runtime as the problem size grows. They recommend that further research on real-parameter operators and the integration of a local search procedure may improve the results of the MOEA.

3.2 Multiobjective Evolutionary Algorithm

The study uses the EA algorithm by Zitzler et al. (1999), the Strength Pareto Evolutionary Algorithm (SPEA). Zitzler et al. (2002) have shown that SPEA2, an improved version of SPEA, outperforms other evolutionary techniques and

seems to have performance advantages in solving optimization problems with higher dimensional objective spaces. For such reasons, this chapter employs SPEA2 to find nondominated solutions to the CFL problem as described in Plastria and Carrizosa (2004).

3.2.1 Genetic Algorithm for the CFLP

Four MOEA methods are introduced in the study. The first is a simple MOEA (S-MOEA) that uses mutation only as implemented by Pangilinan et al. (2005). The second is a variant of the simple MOEA wherein a local search procedure is added to the algorithm (MOEA-LS). The third MOEA-AX uses average crossover and non-uniform mutation. The non-uniform mutation acts as a local search operator at the latter parts of the EA run. The last MOEA-BLX uses blend crossover (Eshelman and Schaffer, 1993) and non-uniform mutation. The blend crossover preserves the mean objective function values of the population. All four MOEAs use the SPEA2 selection algorithm.

Genetic Representation. A chromosome $\boldsymbol{x}$ for the CFL problem is represented by real-parameters, which define the location of a new facility in two dimensions using (x,y)-coordinates. The objective functions are quality α, and captured weight CW of a new facility. Variation and selection operators are applied directly to these real parameter values. The main difficulty here is how to create a new pair of offspring vectors or how to mutate a facility location to a new location in a meaningful manner.

Genetic Operators. Deb (2001) argues that binary crossover techniques hardly create meaningful offspring vectors in real-parameter EAs. Given this observation, implementing a binary crossover operator in a real-parameter search space is not very useful. The problem is then reduced to the design of a crossover and a mutation operator that are strong enough to create diversified solutions and at the same time ensure convergence to optimal solutions. A simple solution would be to implement random mutation per new generation to guarantee diversity but this operator does not ensure convergence. The next step is to generate solutions not from the entire search space but from a search space near the parent solution with a uniform probability distribution. It is also possible to create a non-uniform mutation where the probability of creating solutions closer to the parent increases as the number of generations increase.

Our experiment adopts a non-uniform mutation operator, which is described as follows

$$x_i^{t+1} = x_i^t + p(x_i^l - x_i^u)h(t) \tag{3.6}$$

$$h(t) = 1 - r^{(1-t/t_{max})b} \tag{3.7}$$

where p is either -1 or 1 with probability of 0.5. The terms x_i^l and x_i^u express the lower and upper bounds that define the decision space and restrict the decision variable x_i to take a value within the decision space at generation t. The parameter r is a real random number in the interval [0,1], t is the generation counter, and t_{max} is the maximum number of generations.. The parameter b is a user-defined input that determines non-uniformity and has a value usually greater than 1. Equation 3.6 assures that mutation accomplishes uniform exploration at the first generations and search becomes local at the last generations. Equation 3.7 makes it possible to create a non-uniform mutation where the probability of creating solutions closer to the parent gets higher as the number of generations increase.

Recombination is implemented in two ways, as the arithmetic average of both parent vectors or as a blend crossover of both parents. The arithmetic crossover is defined as

$$z_i = \frac{1}{2}(x_i + y_i) \tag{3.8}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are the parents, $\boldsymbol{z}$ is the offspring, i = 1,…,m, and m is the number of dimensions of the chromosome. The offspring genes represent the arithmetic mean of the values of the parent vectors.
The blend crossover (BLX) is defined as

$$z_i = (1 - \gamma_i)x_i + \gamma_i y_i \tag{3.9}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are the parents, $\boldsymbol{z}$ is the offspring, i = 1,…,m, and m is the number of dimensions of the chromosome, $\gamma_i = (1 + 2\beta)u_i - \beta$. The term u_i is a real random number between 0 and 1. If β is zero, the crossover creates a random solution in the range (x_i, y_i). Eshelman and Schaffer (1993) report that β=0.5 performs better than any other β value for the BLX operator.

Fitness Function. The raw fitness function in this MOEA evaluates the real parameters α and $CW(\alpha, \boldsymbol{x})$ and is a multiobjective function that minimizes α and maximizes $CW(\alpha,\boldsymbol{x})$. Tournament selection as defined in SPEA2 (see Appendix B) is used for the selection of parent solutions.

Local Search. In addition to the basic genetic operators described above, the MOEA-LS uses a local search procedure that improves the convergence rate of the simple MOEA. The local search procedure iteratively searches for the lowest quality α of a facility (individual) in location $\boldsymbol{x}$. This is applied to all new offspring individuals in every generation.

3.2.2 Experiments and Results

The problem considered in this chapter is taken from Plastria and Carrizosa (2004). Specifically, a new facility must be located in the planar region S bounded by a convex polygon with corner points (0,0), (50,0), (50,20), (25,45), (0,45) as shown in Figure 3.1.

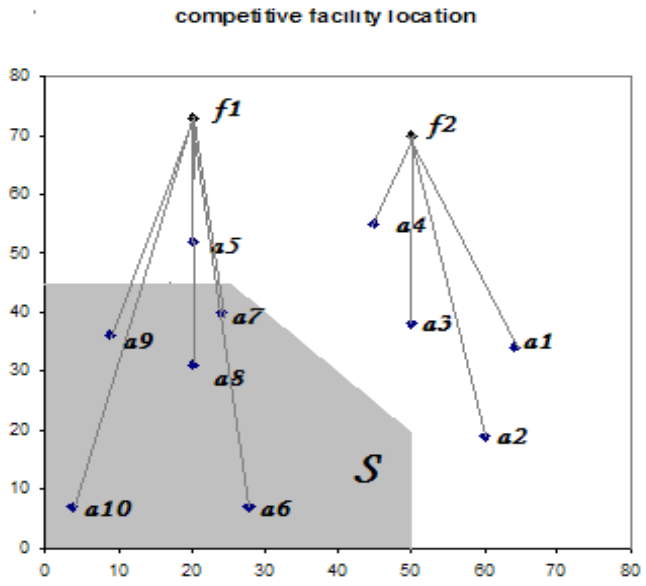

Figure 3.1 Facilities and consumers

A set of two facilities, $CF = \{f_1, f_2\}$ already exist and compete to attract a set of customers $AF = \{a_1, a_2, \ldots, a_{10}\}$. For each consumer a_i (i=1…10) the gravity-type attraction to any facility is defined in (3.1) with each $k_a = 1$ representing some proportionality constant depending on consumer a, some minimal quality $\alpha_0 \approx 0$, $\alpha_0 = 0.0000001$ and $p = 2$. The parameter p represents the sensitivity of attraction $attr$ to distance. Consumer *a* is captured by the existing facility yielding the highest attraction. The attraction $attr_a$, is determined by:

$$attr_a = max\left\{ \frac{\alpha_f}{\left\| \boldsymbol{x}_a - \boldsymbol{x}_f \right\|^2} \middle| f \in CF \right\} \tag{3.10}$$

An overview of the attraction parameters of ten potential consumers is presented in Table 3.1. The terms l_a and w_a represent the location and weight of each consumer *a* respectively.

Consumer	Location l_a	Weight w_a	Attraction $attr_a$	Facility $f(a)$
a1	(64.0, 34.0)	600	0.6702	f_2
a2	(60.0, 19.0)	100	0.3702	f_2
a3	(50.0, 38.0)	100	0.9766	f_2
a4	(45.0, 55.0)	100	4.0000	f_2
a5	(20.0, 52.0)	400	2.8345	f_1
a6	(27.8, 7.0)	300	0.2830	f_1
a7	(24.0, 40.0)	100	1.1312	f_1
a8	(20.0, 31.0)	100	0.7086	f_1
a9	(9.0, 36.0)	100	0.8389	f_1
a10	(3.8, 7.0)	600	0.2707	f_1

Table 3.1 Attraction Parameters

One difficulty of using evolutionary algorithms is its sensitivity to the input parameters such as population size, random seed, number of generations, and probability values for crossover and mutation. Determining a balanced interaction between parameter settings to find efficient solutions is not an easy task. A number of simulation runs is usually required to come up with near optimal solutions. The set of parameters shown in Table 3.2 is one of the many combinations that may be used for the competitive facility location problem.

Parameter *maxgen* stands for the maximum number of generations. The maximum quality value is indicated as *quality_max*. Parameters *xy_range* and *quality_range* signify maximum perturbation for location and quality. *Mutation_probability* is the probability that a mutation is performed. *Bit_turn_probability* represents the probability of a gene to be mutated. The *initial population* is the number of individuals at the start of the MOEA run. From an *initial population*, *a parent population* is selected to reproduce a population of *offspring individuals*. The process of selecting parents and reproducing offspring is repeated until *maxgen* is reached.

Parameter	S-MOEA	MOEA-LS	MOEA-AX	MOEA-BLX
initial population	25	25	25	25
parent population	25	25	25	25
offspring individuals	25	25	25	25
maxgen	2000	250	250	250
quality_max	2000	2000	2000	2000
xy_range	5	5	4.28	7.19
quality_range	15	15	1.25	13.13
mutation_probability	1.00	1.00	0.69	0.47
crossover_probability	0.00	0.00	0.44	0.91
bit_turn_probability	0.50	0.50	0.81	0.53

Table 3.2 Input parameters

Table 3.3 shows comparative results of Plastria and Carrizosa's algorithm (called Plastria in the column table), S-MOEA, MOEA-LS, MOEA-AX, and MOEA-BLX from the given sample problem. Plastria and Carrizosa's results show the complete set of optimal solutions. Clearly, all MOEA results show inferior solutions to that of Plastria and that the MOEA-LS, the MOEA-BLX, and the MOEA-AX perform better than the S-MOEA. The worst solutions are generated by the S-MOEA (at 250 generations) because its search operator is weak. It requires 2000 generations to have comparable results with the other MOEAs. This shows that a non-uniform local search is more effective than a uniform search. The best solutions are obtained by the MOEA-AX, which uses arithmetic crossover and non-uniform mutation.

The solutions of all MOEA algorithms are near optimal because the CFLP has a real-parameter search space. Deb (2001, p.124) explains that after a few generations, a real-parameter genetic algorithm treats a continuous search

space problem as a discrete search space problem and the global optimum is difficult to find.

CW	Plastria α	S-MOEA α 2000 generations	MOEA-LS α 250 generations	MOEA-AX α	MOEA-BLX α
600.00	0.00	0.00	0.00	0.00	0.08
900.00	39.85	40.45	39.92	40.36	40.01
1000.00	89.83	91.41	89.87	90.46	93.15
1100.00	135.27	136.69	135.84	135.51	144.72
1200.00	182.72	184.10	183.23	182.75	200.52
1300.00	359.56	362.34	360.86	361.92	363.88
1600.00	362.00	363.41	362.81	362.32	365.47
1800.00	440.48	447.06	443.73	441.31	441.46
1900.00	446.91	448.73	448.89	447.34	459.72
2000.00	566.04	568.70	566.40	566.11	589.47
2400.00	767.59	770.36	768.99	768.60	769.34
2500.00	1800.00	1812.14	1806.45	1805.07	1814.01

Table 3.3 Comparative results of efficient solutions.

In terms of runtime complexity, Plastria and Carrizosa's algorithm is able to find the list of efficient solutions in $O(n^3 log n)$ time where n is the number of consumers. On the other hand, the MOEAs takes $O(M^2)$ to evaluate the raw fitness of all candidate solutions per generation. The fitness assignment in SPEA2 per generation takes $O(M^2 log M)$ and the environmental selection per generation takes $O(M^2 log M)$ where M is the sum of the list of candidate solutions N and the archive size. Calculation of the quality α and captured weights CW per generation takes $O(nM)$. The runtime complexity of the MOEAs is dominated by its selection operator and is dependent on the initial population size N and not on the number of customers n. With regard to the MOEAs, the MOEA-LS and the MOEA-AX generates better solutions i.e. solutions nearer to the Pareto-front and finds these solutions in a much shorter time (250 generations against 2000 generations). Plastria and Carrizosa (2004) note that their geometric procedure does not extend well in finding solutions when non-Euclidean distances are used. An MOEA does not suffer from such difficulty since it does not employ a geometric procedure to find CFLP solutions. Hence, the MOEA becomes more extensible to planar problems with a non-Euclidean metric.

3.3 Sensitivity Analysis

The purpose of *uncertainty* analysis is to determine the uncertainty in estimates for dependent variables of interest (Saltelli 2000, Saltelli 1993). The purpose of *sensitivity analysis* (SA) is to determine the relationships between the uncertainty in the independent variables and the uncertainty in the dependent variables. Uncertainty analysis typically precedes SA since there is no reason to perform SA when the uncertainty in a dependent variable is below an acceptable bound or range. Sensitivity analysis is often defined as a local measure of the effect of a given input on a given output. This measure can be achieved most often by Monte Carlo methods in conjunction with a variety of sampling strategies (Saltelli et al. 2004). Monte Carlo sensitivity analysis is based on performing multiple model evaluations with probabilistically selected model inputs, and the results of such evaluations are used to 1) determine the uncertainty in model predictions and 2) identify the input variables that influence uncertainty.

SIMLAB (2004) is a program designed for global uncertainty and sensitivity analysis based on Monte Carlo methods (see Appendix A). It offers several techniques for sample generation, sensitivity analysis, and a link to external model execution. The link allows execution of complex models that can hardly be coded as simple mathematical functions such as genetic algorithms.

3.3.1 Experiments and Results

Pangilinan et al. (2008) conducted a sensitivity analysis of the MOEA-AX for the CFLP to determine which input parameters affect the output and avoid assigning arbitrary values to the input parameters. The following experiment is patterned from their study but with the addition of a sensitivity analysis for the MOEA-BLX and a robustness test for the MOEA-AX. The sensitivity analysis for the MOEA-BLX is added to compare the effect of different crossover operators on the same problem set. The robustness test is added to show that the MOEA can find near-optimal solutions without repeating a sensitivity analysis when the position of the polygon S are altered.

The five input parameters for the genetic algorithm model and their configuration are shown in Table 3.4. The factor *xyrange* defines the upper and

lower limit for searching a point in the polygon S. The factor *αrange* defines the limits for searching the quality α of the new facility. *Mrate, xover*, *bitrate* are the mutation, recombination, and bit-turn probabilities respectively.

Input Factor	Description	Probability
xyrange	Range for searching location	Uniform(0, 10)
αrange	Range for searching quality	Uniform(0, 20)
mrate	Mutation probability	Uniform(0, 1)
xover	Crossover probability	Uniform(0, 1)
bitrate	Bit-turn probability	Uniform(0, 1)

Table 3.4 Probability distribution of input parameters

The bit-turn probability is the probability that a gene will undergo mutation whereas mutation rate defines the probability that a chromosome or an individual will undergo mutation. The MOEA runs for 250 generations for each configuration. The Sobol' method in SIMLAB (see Appendix A) is used for sensitivity analysis, which generates 384 input configurations for the five input parameters of the genetic algorithm.

Table 3.5 shows the comparative results of captured weights and facility quality as computed from Plastria and Carrizosa's (2004) and the best solution set from 384 different solution sets generated by the genetic algorithms. Plastria and Carrizosa's (2004) results show the Pareto-optimal set ***P****. The genetic algorithm was able to compute the exact captured weight CW of the new facilities but show inferior solutions of quality values α.

The performance of a solution set ***Q*** from the genetic algorithm is based on three parameters namely, (1) the ratio of CW Pareto-optimal solutions in ***Q*** over the number of solutions in ***P****, (2) the ratio of CW solutions in ***Q*** that are members of ***P**** over the number of solutions in ***Q***, and (3) the sum of differences in quality values of ***Q*** and ***P****. The best solution ***Q****, is the solution that has all the CW solutions in ***P**** and the smallest sum of the differences of quality values between ***Q**** and ***P****. The quality values of the best solutions from the MOEA-AX and MOEA-BLX are shown in Table 3.5. The quality values generated by the MOEA-AX are nearer to the optimal values than values from the MOEA-BLX, which means that the arithmetic crossover performs better than the blend crossover in this CFLP problem set.

CW	Plastria (α) $\boldsymbol{P}$*	MOEA-AX ($\boldsymbol{Q}$*)	MOEA-BLX ($\boldsymbol{Q}$*)
600	0.0000	0.0003	0.0818
900	39.8488	40.3606	40.0118
1000	89.8289	90.4611	93.1452
1100	135.2698	135.5118	144.7237
1200	182.7161	182.7483	200.5216
1300	359.5603	361.9230	363.8839
1600	361.9952	362.3166	365.4698
1800	440.4785	441.3086	441.4561
1900	446.9055	447.3366	459.7156
2000	566.0434	566.1086	589.4663
2400	767.5907	768.5999	769.3388
2500	1800.0000	1805.0749	1814.0083

Table 3.5 Comparative results of best solutions

As regards the sensitivity analysis, Table 3.6 shows the Sobol first-order and total-order indices. The first-order sensitivity index shows the individual effect of an input factor on the output. More specifically a first-order index gives a measure of the direct effect of an input factor on the output variation. An input parameter having a first-order index with the least value means that it has the least influence on the output whereas a factor with the highest first-order value is most important for further investigation. If the sum of the first-order indices equals 1.0, then the model is linear. If the sum of the Sobol first-order indices does not equal 1.0 then the model is nonlinear and implies that some effects on the output are due to interactions among the input factors. First-order sensitivity measures do not capture interactions between factors whereas total-order sensitivity measures detect interactions among input factors.

The total-order indices describe the share of the output variation that is related to each input factor. This includes the direct effect as well as interactions with other factors. Removal of a factor means removal of the amount of the total-order index from the output variation. Hence, only factors with very small total-order indices can be removed to avoid significant changes in the output.

Table 3.6 has three columns that represent the performance metrics for the MOEA-AX described previously. The values represent the first-order and total-order sensitivity indices. The mutation rate has the highest first-order index and

causes nearly 30% of the variation in each of the output metrics. Similarly, the factors *xyrange* and the *bitrate* share almost 50% of the variation as shown in the third column output metric-the sum of the differences in the quality α of a facility between solutions ***Q*** and ***P**** and can be interpreted as the proximity of an α solution in ***Q*** to ***P****. The parameters α*range* and *xover* have almost no direct influence on the output metrics.

Sobol first order indices			
	Number of CW solutions in ***Q*** over |***P****|	Number of CW solutions in that are in ***P**** over |***Q***|	Sum of differences in quality
xyrange	0.047	0.115	0.298
α*range*	-0.089	-0.077	0.039
mrate	0.286	0.318	0.292
xover	-0.002	0.082	0.054
bitrate	0.065	-0.070	0.207
Sobol total order indices			
xyrange	0.602	0.642	0.407
α*range*	0.450	0.264	0.153
mrate	0.611	0.637	0.528
xover	0.200	0.300	0.137
bitrate	0.405	0.243	0.372

Table 3.6 Sobol first-order and total-order indices for MOEA-AX

The sum of the first-order indices does not add up to 1.0, which means that there is interaction among the factors. The mutation rate factor *mrate* remains the highest in the total-order indices followed by the parameters *xyrange* and the *bitrate*. The order of the remaining factors has changed implying that higher-order effects of these factors vary.

The crossover probability factor shows small values and can be a candidate for removal from the list of parameters. The results also show that the mutation rate has the greatest influence on the output but specific mutation probabilities that generated near-optimal solutions are not known. Further investigation reveals that out of the 384 input configurations 21 samples have generated near-optimal solutions i.e. they have detected all Pareto-optimal solutions for the function "captured weight", CW and have small differences in values for the function "facility quality", α. The 21 samples that generated near-optimal solutions show high values for the mutation rate, specifically an average of 0.75

and a standard deviation of 0.16. Other averages for the input factors are *bitrate*= 0.62, *xyrange* = 5.96, *αrange* = 6.46, and *xover* = 0.40.

Table 3.7 shows the performance results from the MOEA-BLX. The crossover rate has the highest first-order index, followed by the *bitrate* and mutation rate. They account for 68 % of the variation in the first column output metric and 53% of the third column output metric. The factors *xyrange* and the *αrange* have almost no direct influence on all the output metrics. However, their total-order effects are significant. There are 23 near-optimal solutions generated by the MOEA-BLX. The mutation rate averaged at 0.63, the crossover rate at 0.60 and the bit rate at 0.53. The average of *xyrange* is 5.04 and *αrange* is 9.60.

Sobol First order indices			
	Number of CW solutions in $\boldsymbol{Q}$ over $\lvert\boldsymbol{P}^*\rvert$	Number of CW solutions in $\boldsymbol{Q}$ that are in $\boldsymbol{P}^*$ over $\lvert\boldsymbol{Q}\rvert$	Sum of differences in quality d
xyrange	-0.023	-0.017	0.018
αrange	0.117	0.041	-0.013
mrate	0.222	-0.005	0.153
xover	0.240	0.035	0.229
bitrate	0.235	-0.108	0.168
Sobol total order indices			
xyrange	0.486	0.801	0.386
αrange	0.849	0.742	0.708
mrate	0.665	0.895	0.544
xover	0.631	0.694	0.687
bitrate	0.920	0.650	0.847

Table 3.7 Sobol first-order and total-order indices for MOEA-BLX

Tables 3.6 and 3.7 show the sensitivity indices of two MOEAs having different crossover operators. Their values are very different which means that the type of crossover operators while having the same mutation and selection operator changes the behavior of the MOEA in terms of the individual and interaction effects of each input parameter to the output. A common result between both algorithms is that the mutation rate and the bit-turn rate are kept high (>0.60) to get near-optimal solutions.

ROBUSTNESS

The interaction among parameters with high-valued indices should be further investigated in order to improve the solutions in the real-valued objective (quality of facility). A method that requires lesser input sample sizes will be beneficial in the investigation of sensitivity analyses of evolutionary algorithms as applied to CFL problem. To test whether a smaller input sample set is beneficial, Table 3.8 is a list of ten parameter combinations that produced the best CFL solutions. These are tested on different positions of $\boldsymbol{S}$. The succeeding experiment explores and describes the solutions that are generated by these ten parameter combinations using MOEA-AX for nine different area locations as defined by the polygon $\boldsymbol{S}$. The MOEA-AX is chosen for the robustness test since it has produced the better solutions than the other MOEAs.

	xyrange	qrange	mutation	xover	bitturn
1	4.38	1.25	0.69	0.44	0.81
2	3.13	3.75	0.94	0.06	0.44
3	5.31	0.63	0.91	0.72	0.34
4	7.81	0.63	0.78	0.59	0.34
5	3.13	3.75	0.94	0.56	0.44
6	1.88	1.25	0.94	0.19	0.56
7	7.50	5.00	0.75	0.25	0.75
8	9.22	14.69	0.95	0.55	0.83
9	7.19	13.13	0.47	0.28	0.53
10	7.81	9.38	0.78	0.72	0.34

Table 3.8 GA parameters that give best solutions to MOEA-AX

The experiment intends to determine if the same parameter combinations are able to generate near-optimal CFLP solutions on differing areas as shown in Figures 3.2-3.9. Figure 3.2 is a translation of the original area by +15 units on the *x*-axis and Figure 3.3 is a translation of the original area by +15 units on the *y*-axis. Figure 3.4 is a translation of the original area by +15 units on both the *x*- and *y*-axes. Figure 3.5, 3.6, and 3.7 are reflections of the original area on the *y*-axis, *x*-axis, and the *xy*-axes respectively. Figure 3.8 is the area rotated clockwise by 90 degrees at location (50,20) and Figure 3.9 is the original area rotated counter-clockwise by 90 degrees at location (25,45).

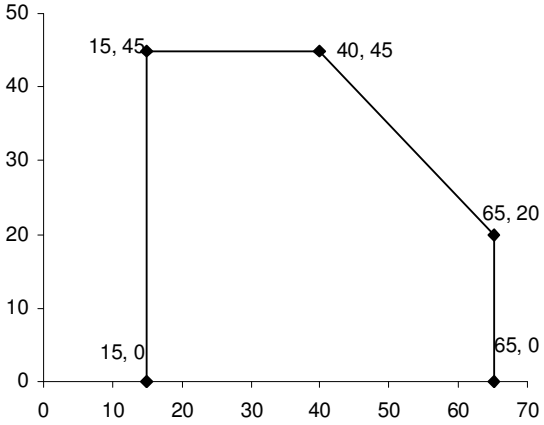

Figure 3.2 Translation of S on *x*-axis

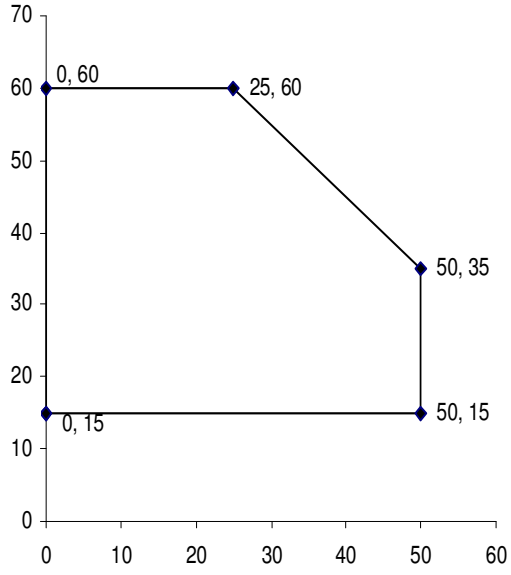

Figure 3.3 Translation of S on *y*-axis

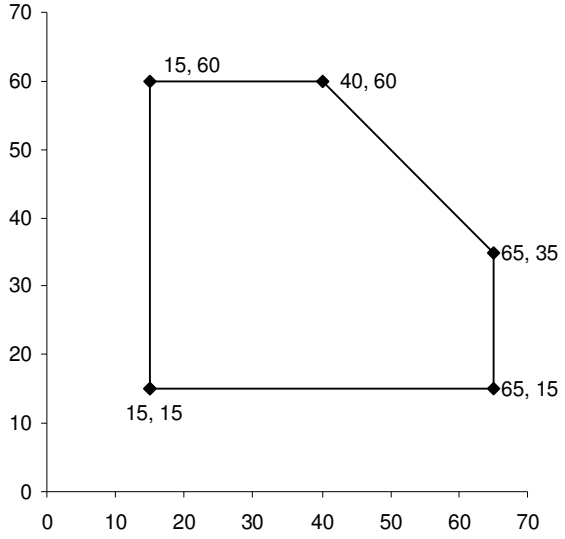

Figure 3.4 Translation of S on *xy*-axis

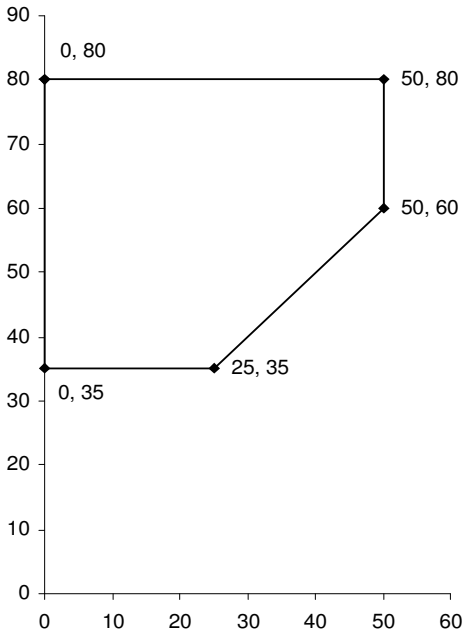

Figure 3.5 Reflection of S at *y*-axis

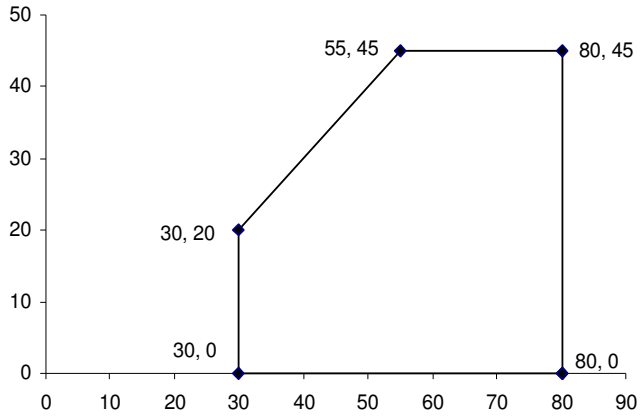

Figure 3.6 Reflection of S on *x*-axis

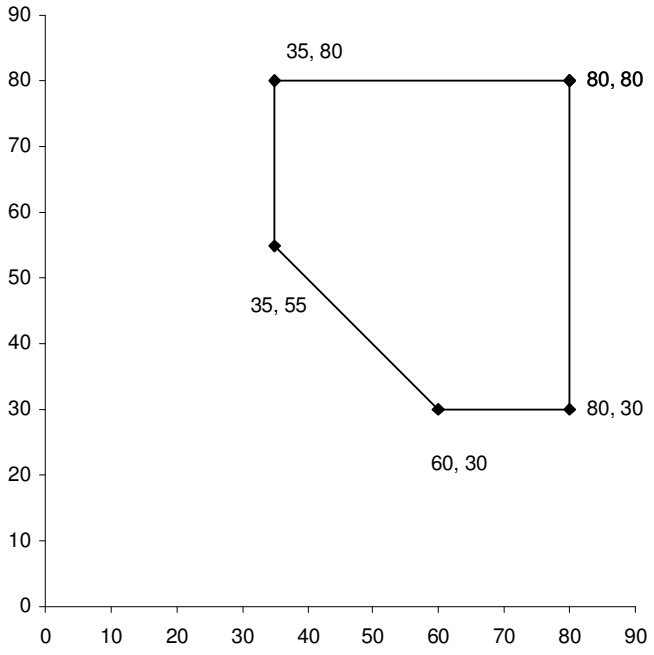

Figure 3.7 Reflection of S at *xy*-axes, *y*=80-*x*

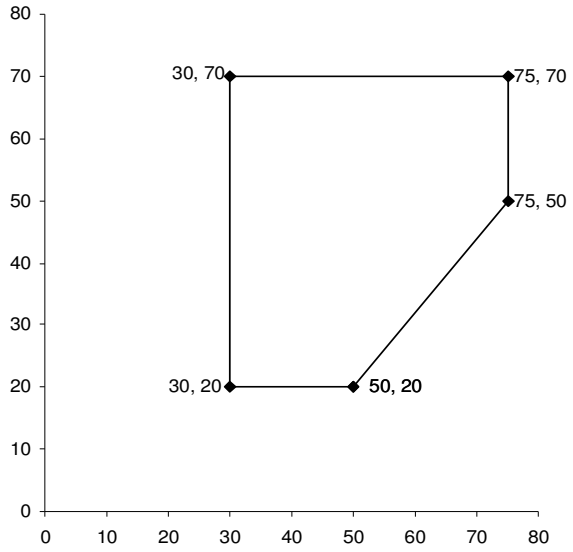

Figure 3.8 Rotation of S at (50,20) clockwise 90°

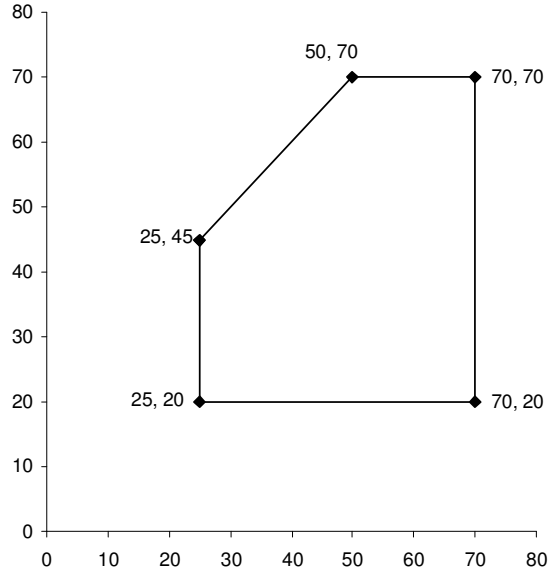

Figure 3.9 Rotation of S at (25,45) counter-clockwise 90°

Table 3.9 shows the quality values and captured weights after translation of $\boldsymbol{S}$ on the *x*-axis (see Figure 3.2). The best solution is $\boldsymbol{P}^*$, and the quality values are from nondominated sets $\boldsymbol{Q}_1$ to $\boldsymbol{Q}_{10}$. The term $\boldsymbol{Q}_i$, where $i \in \{1,2,\ldots,10\}$ represents a nondominated set of CFLP solutions generated form in an input combination listed in Table 3.7. Solutions $\boldsymbol{Q}_1$, $\boldsymbol{Q}_3$, $\boldsymbol{Q}_4$, $\boldsymbol{Q}_8$, and $\boldsymbol{Q}_{10}$ have the complete set of solutions and have mutation rates ≥ 0.69 and crossover rates ≥ 0.44. Whereas solution sets $\boldsymbol{Q}_2$, $\boldsymbol{Q}_5$, $\boldsymbol{Q}_6$, $\boldsymbol{Q}_7$, and $\boldsymbol{Q}_9$ miss one captured weight CW = 1800 and their corresponding quality values. The incomplete solutions have mutation rates ≥ 0.75 (except for $\boldsymbol{Q}_9$, which has 0.47) and crossover rates ≤ 0.56. The nearest solution to $\boldsymbol{P}^*$, according to the performance criteria described above is $\boldsymbol{Q}_8$ with a mutation rate of 0.95, bit-turn probability of 0.83, and a crossover rate of 0.55 followed by $\boldsymbol{Q}_3$ with a mutation rate of 0.91, bit-turn probability of 0.72, and a crossover rate of 0.34. The worst vector is $\boldsymbol{Q}_6$ with a mutation rate of 0.94, bit-turn probability of 0.56, and a crossover rate of 0.19.

$\boldsymbol{Q}_1$	$\boldsymbol{Q}_2$	$\boldsymbol{Q}_3$	$\boldsymbol{Q}_4$	$\boldsymbol{Q}_5$	$\boldsymbol{Q}_6$	$\boldsymbol{Q}_7$	$\boldsymbol{Q}_8$	$\boldsymbol{Q}_9$	$\boldsymbol{Q}_{10}$	CW	$\boldsymbol{P}^*$
0.077	1E-04	0.001	9E-04	2E-04	0.014	0.003	0.215	5E-04	0.016	300	1E-04
33.9	33.9	33.9	33.9	33.9	33.8	34.1	33.9	34.2	34.2	600	33.8
39.9	39.9	39.9	89.8	40.3	39.9	39.9	40.4	40.9	41.3	900	39.9
91.5	103.3	89.9	90.5	92.3	97.8	91.0	90.5	96.7	97.3	1000	89.9
135.8	139.6	135.7	135.3	137.1	135.4	135.7	135.9	135.6	140.0	1100	135.3
187.4	183.0	185.4	184.9	192.2	188.8	182.8	182.8	194.0	190.9	1200	182.8
359.8	368.7	359.9	359.6	355.7	370.6	359.8	359.8	369.9	358.7	1300	355.7
398.6	400.5	398.1	398.9	396.1	399.1	397.4	364.9	397.1	397.3	1400	364.9
441.9		441.2	441.0				442.6		443.0	1800	441.0
447.6	455.8	449.8	447.7	449.2	473.2	449.0	453.1	463.3	447.0	1900	447.0
566.3	566.8	566.5	566.5	567.0	566.4	566.3	566.4	566.6	568.1	2000	566.3
687.9	688.6	688.0	689.6	689.1	693.4	687.9	688.3	690.4	689.9	2400	687.9
789.1	789.0	789.1	791.0	789.0	788.8	789.2	789.2	790.1	790.0	2500	788.8

Table 3.9 Quality values after translation of $\boldsymbol{S}$ on the *x*-axis

Table 3.10 shows the quality values and captured weights after translation of $\boldsymbol{S}$ on the *y*-axis (see Figure 3.3). Vectors $\boldsymbol{Q}_2$ and $\boldsymbol{Q}_6$, have the complete set of solutions and have mutation rates equal to 0.94 and crossover rates ≤ 0.44. Vectors $\boldsymbol{Q}_1$, $\boldsymbol{Q}_4$ and $\boldsymbol{Q}_{10}$ miss one solution, the solution for CW=2100. Solution sets $\boldsymbol{Q}_5$, $\boldsymbol{Q}_8$, and $\boldsymbol{Q}_9$ miss two solutions each whereas $\boldsymbol{Q}_3$ and $\boldsymbol{Q}_7$ miss three solutions. The nearest solution to $\boldsymbol{P}^*$ is $\boldsymbol{Q}_2$ with a mutation rate of 0.94, bit-turn probability of 0.44, and a crossover rate of 0.06 followed by $\boldsymbol{Q}_6$ with a mutation rate of 0.94, bit-turn probability of 0.56, and a crossover rate of 0.19. The worst

vector is $\boldsymbol{Q}_3$ with a mutation rate of 0.91, bit-turn probability of 0.34, and a crossover rate of 0.72.

$\boldsymbol{Q}_1$	$\boldsymbol{Q}_2$	$\boldsymbol{Q}_3$	$\boldsymbol{Q}_4$	$\boldsymbol{Q}_5$	$\boldsymbol{Q}_6$	$\boldsymbol{Q}_7$	$\boldsymbol{Q}_8$	$\boldsymbol{Q}_9$	$\boldsymbol{Q}_{10}$	CW	$\boldsymbol{P}^*$
0.002	0.0001	0.0001	0.0001	0.001	0.001	0.001	0.095	0.003	0.0003	100	0.0001
25.1	17.3	17.4	17.4	17.7	17.4	17.4	18.0	23.8	19.2	600	17.3
59.0	57.9	57.6	57.9	57.6	58.2	57.9	62.1	58.3	59.0	900	57.6
90.3	90.2	90.5	90.4	91.0	90.0	90.0	91.7	92.1	90.5	1000	90.0
136.0	135.3	136.5	136.6	136.2	135.6	135.6	135.6	137.2	139.9	1100	135.3
182.8	183.5	182.9	183.7	185.0	182.8	182.8	183.0	183.0	182.9	1200	182.8
361.9	361.8	366.7	361.5	361.8	362.0	360.1	361.0	359.8	361.7	1300	359.8
363.3	362.1		364.5	362.2	362.3		364.2	363.0	363.4	1600	362.1
446.6	444.7		440.9		440.6			443.7	442.6	1800	440.6
447.8	449.0	452.7	448.7	451.3	459.7	447.6	450.9	447.7	449.3	1900	447.6
566.4	567.3	566.2	566.3	566.4	566.4	566.9	566.4	567.7	566.4	2000	566.2
	684.4				685.5					2100	684.4
686.4	686.9	686.5	686.7	686.1	686.2	686.3	686.9		687.8	2400	686.1
688.7	689.2	688.8	688.7	689.3	688.8	689.0	689.2	689.4	690.1	2500	688.7

Table 3.10 Quality values after translation of $\boldsymbol{S}$ on the *y*-axis

Table 3.11 shows the quality values and captured weights after translation of $\boldsymbol{S}$ on both the *x* and *y* axes (see Figure 3.4). No single solution set captures all the quality values and captured weights as listed in the column of $\boldsymbol{P}^*$. Solution sets $\boldsymbol{Q}_1$, $\boldsymbol{Q}_7$, and $\boldsymbol{Q}_{10}$ miss only one solution, CW=2100 and vectors $\boldsymbol{Q}_5$ and $\boldsymbol{Q}_8$ miss one solution, CW=1800 and CW=100 respectively. The remaining vectors $\boldsymbol{Q}_2$, $\boldsymbol{Q}_3$, and $\boldsymbol{Q}_4$ miss two solutions whereas $\boldsymbol{Q}_6$ and $\boldsymbol{Q}_9$ miss three solutions. The nearest solution to $\boldsymbol{P}^*$ is $\boldsymbol{Q}_{10}$ with a mutation rate of 0.78, bit-turn probability of 0.34, and a crossover rate of 0.72 followed by $\boldsymbol{Q}_7$ with a mutation rate of 0.75, bit-turn probability of 0.75, and a crossover rate of 0.25. The worst vector is $\boldsymbol{Q}_8$ with a mutation rate of 0.95, bit-turn probability of 0.85, and a crossover rate of 0.55.

Table 3.12 shows the quality values and captured weights of the reflection of $\boldsymbol{S}$ on the *y*-axis (see Figure 3.5). Vectors $\boldsymbol{Q}_3$ and $\boldsymbol{Q}_6$, have the complete set of solutions. The nearest solution to $\boldsymbol{P}^*$ is $\boldsymbol{Q}_6$ with a mutation rate of 0.94, bit-turn probability of 0.56, and a crossover rate of 0.19 followed by $\boldsymbol{Q}_3$ with a mutation rate of 0.91, bit-turn probability of 0.72, and a crossover rate of 0.34. The worst vector is $\boldsymbol{Q}_8$ with a mutation rate of 0.95, bit-turn probability of 0.85, and a crossover rate of 0.55.

$\boldsymbol{Q}_1$	$\boldsymbol{Q}_2$	$\boldsymbol{Q}_3$	$\boldsymbol{Q}_4$	$\boldsymbol{Q}_5$	$\boldsymbol{Q}_6$	$\boldsymbol{Q}_7$	$\boldsymbol{Q}_8$	$\boldsymbol{Q}_9$	$\boldsymbol{Q}_{10}$	CW	$\boldsymbol{P}^*$
1.868	2E-05	0.0001	0.0001	0.0008	0.0002	0.124	0.011	0.022	0.024	600	2E-05
29.8	43.1	42.6	31.2	43.6	42.6	29.7	31.5	31.9	43.7	700	29.7
58.7	58.0	58.0	57.9	57.7	74.9	57.9	58.3	59.2	57.9	900	57.7
95.1	92.6	90.1	90.6	90.6	90.1	91.8	90.6	90.6	90.4	1000	90.1
135.7	135.5	135.4	135.4	135.5	135.4	135.5	135.5	137.3	136.5	1100	135.4
183.1	183.6	183.0	183.9	185.3	183.4	182.8	184.1	189.4	184.8	1200	182.8
350.5	359.9	360.1	359.9	363.4	350.2	362.6	350.6	361.2	372.6	1300	350.2
398.1	396.5	401.3	395.3	395.7	397.0	398.5	398.3		397.3	1400	395.3
444.8		440.7	443.7			447.9	447.3	445.7	440.8	1800	440.7
447.4	449.4	454.0	448.5	448.3	452.1	448.4	455.8	448.4	449.5	1900	447.4
566.8	566.8	566.5	566.2	566.3	566.1	566.2	566.1	566.2	569.7	2000	566.1
	683.9	686.6		685.2			689.2			2100	683.9
686.3	687.2	689.2	687.3	686.1	685.8	686.0	1344.6	687.0	687.2	2400	685.8
689.0	689.0		688.9	689.0	688.7	688.7	1462.5	689.3	691.3	2500	688.7

Table 3.11 Quality values after translation of $\boldsymbol{S}$ in both x and y-axes

$\boldsymbol{Q}_1$	$\boldsymbol{Q}_2$	$\boldsymbol{Q}_3$	$\boldsymbol{Q}_4$	$\boldsymbol{Q}_5$	$\boldsymbol{Q}_6$	$\boldsymbol{Q}_7$	$\boldsymbol{Q}_8$	$\boldsymbol{Q}_9$	$\boldsymbol{Q}_{10}$	CW	$\boldsymbol{P}^*$
0.087	0.0022	0.0039	0.0028	0.0003	5E-05	0.013	0.141	0.002	0.0002	400	5E-05
73.3	69.3	68.0	68.6	68.5	69.2	70.5	69.9	69.0	73.5	500	68.03
135.7	139.7	133.1	132.8	133.2	133.0	133.6	133.0	135.7	140.3	600	132.8
143.7	144.0	143.0	143.3	143.7	142.8	143.3	144.8	146.3	145.6	700	142.8
212.2	212.6	212.2	212.5	212.4	212.2	212.3	212.5	212.5	212.8	800	212.2
225.6	226.1	225.6	226.6	226.2	225.9	226.5	227.3	230.5	226.1	900	225.6
257.0	257.0	257.1	257.3	257.1	257.3	257.4	258.1	257.8	257.7	1200	257
		361.9	361.5		361.7				362.1	1300	361.5
362.6	364.7	362.5	362.1	362.2	362.1	362.4	362.8	362.7	363.5	1600	362.1
485.7	485.7	487.7	485.1	485.3	485.1	487.6	489.4	486.2	488.6	1700	485.1
538.5	537.9	537.7	537.6	538.1	537.5	537.6	542.0	538.9	539.7	1800	537.5
649.8	649.4	649.4	651.1	650.1	649.4	651.3	649.3	652.9	650.5	2000	649.3
685.6	684.0	685.2		683.5	683.6				687.2	2100	683.5
686.6	686.7	685.8	686.6	686.5	686.4	686.2		686.5		2400	685.8
688.7	688.8	689.0	689.0	689.0	688.7	689.0	689.1	689.0	690.8	2500	688.7

Table 3.12 Quality values of the reflection of $\boldsymbol{S}$ at y=40

Table 3.13 shows the quality values and captured weights of the reflection of $\boldsymbol{S}$ on the x-axis (see Figure 3.6). Vector $\boldsymbol{Q}_2$ has the complete set of solutions and all the other vectors have two or more missed solutions. Following the performance criteria described previously the nearest solution to $\boldsymbol{P}^*$ is $\boldsymbol{Q}_2$ with a mutation rate of 0.94, bit-turn probability of 0.44, and a crossover rate of 0.06 followed by $\boldsymbol{Q}_7$ with a mutation rate of 0.75, bit-turn probability of 0.75, and a

crossover rate of 0.25. The worst vector is $\boldsymbol{Q}_3$ with a mutation rate of 0.91, bit-turn probability of 0.34, and a crossover rate of 0.72.

$\boldsymbol{Q}_1$	$\boldsymbol{Q}_2$	$\boldsymbol{Q}_3$	$\boldsymbol{Q}_4$	$\boldsymbol{Q}_5$	$\boldsymbol{Q}_6$	$\boldsymbol{Q}_7$	$\boldsymbol{Q}_8$	$\boldsymbol{Q}_9$	$\boldsymbol{Q}_{10}$	CW	$\boldsymbol{P}^*$
	4E-05		0.0003		1E-06					100	1E-06
0.04	5.0	3E-04	12.9	0.0001	0.0001	0.001	0.04	0.0002	0.0003	600	1E-04
30.7	43.0	42.6	29.7	30.3	42.6	29.8	30.6	34.4	29.9	700	29.7
67.5	66.9	69.2	69.4	66.6	66.0	66.5	69.3	66.1	69.7	800	66.0
185.1	185.2				185.1	185.4	185.3			900	185.1
	212.1									1000	212.1
212.4	212.4	220.1	212.7	212.7	217.8	212.3	213.6	212.2	215.2	1100	212.2
335.5	335.8	335.5	337.2	337.5	336.3	335.8	338.4	336.3	336.6	1200	335.5
382.9	384.5	383.5	383.2	384.8	383.4	403.8	383.0	383.8	385.1	1300	382.9
440.5	440.9	440.6	441.0	440.7	440.6	441.4	441.4	440.7	442.4	1800	440.5
451.0	447.6	448.3	448.4	447.4	447.4	452.5	448.0	448.7	448.9	1900	447.4
681.2	655.9	682.0	674.0	652.0	652.0	678.4	653.4	652.9	673.1	2000	652.0
1245	1245	1246	1246	1246	1245	1246	1246	1256	1246	2100	1245

Table 3.13 Quality values of the reflection of $\boldsymbol{S}$ at x=40

Table 3.14 shows the quality values and captured weights of the reflection of $\boldsymbol{S}$ on both the x and y axes (see Figure 3.7). Vectors $\boldsymbol{Q}_1$, $\boldsymbol{Q}_6$, $\boldsymbol{Q}_7$, and $\boldsymbol{Q}_9$ have the complete set of solutions. Vectors $\boldsymbol{Q}_2$, $\boldsymbol{Q}_3$, $\boldsymbol{Q}_5$, and $\boldsymbol{Q}_{10}$ miss one solution each. The nearest solution to P^* is $\boldsymbol{Q}_1$ with a mutation rate of 0.69, bit-turn probability of 0.81, and a crossover rate of 0.44 followed by $\boldsymbol{Q}_7$ with a mutation rate of 0.75, bit-turn probability of 0.75, and a crossover rate of 0.25.

$\boldsymbol{Q}_1$	$\boldsymbol{Q}_2$	$\boldsymbol{Q}_3$	$\boldsymbol{Q}_4$	$\boldsymbol{Q}_5$	$\boldsymbol{Q}_6$	$\boldsymbol{Q}_7$	$\boldsymbol{Q}_8$	$\boldsymbol{Q}_9$	$\boldsymbol{Q}_{10}$	CW	$\boldsymbol{P}^*$
1.307	4E-05	3E-04	0.0002	3E-05	0.003	0.001	0.022	0.0026	0.0011	600	3E-05
42.6	42.7	42.7	42.6	42.8	42.5	43.2	44.9	45.8	43.2	700	42.5
74.7	74.8	75.0	77.5	75.1	80.6	75.2	76.9	80.0	75.2	800	74.7
333.4	331.6	331.4	331.8	331.5	331.2	334.6	331.3	331.6	337.2	900	331.2
430.9	431.2	430.6	432.3	430.7	431.8	431.1	444.1	430.6	431.5	1100	430.6
482.0	481.7	482.0	483.7	482.3	482.5	482.1	482.9	482.1	483.1	1200	481.7
500.9	506.1	501.4	505.6	503.4	502.2	502.1	502.4	507.8	500.8	1300	500.8
542.8	547.5	543.5	546.8	542.8	549.0	543.0	546.2	543.1	546.0	1400	542.8
769.0			770.5		792.0	768.1		770.1	848.3	1800	768.1
848.3	848.1	848.1	848.1	848.0	848.4	848.4	848.1	848.1		2100	848.0
868.5	867.2	910.8	867.7	897.6	869.3	867.8	867.1	870.3	911.8	2400	867.1
882.0	908.5	917.6	879.5	914.8	893.0	887.6	907.4	887.0	917.9	2500	879.5

Table 3.14 Quality values of the reflection of $\boldsymbol{S}$ at y=80-x

The worst vector is $\boldsymbol{Q}_{10}$ with a mutation rate of 0.78, bit-turn probability of 0.34, and a crossover rate of 0.72.

Table 3.15 shows the quality values and captured weights after $\boldsymbol{S}$ is rotated clockwise by 90 degrees at (50, 20) (see Figure 3.8). No single vector captures all the quality values and captured weights as listed in the column of $\boldsymbol{P}^*$. Solution sets $\boldsymbol{Q}_1$, $\boldsymbol{Q}_3$, and $\boldsymbol{Q}_6$ miss only one solution, CW= 1400 and vectors $\boldsymbol{Q}_9$ and $\boldsymbol{Q}_{10}$ miss one solution each, CW=2100 and CW=2400 respectively. Vectors $\boldsymbol{Q}_4$ and $\boldsymbol{Q}_5$ miss solution CW= 2100. The remaining vectors $\boldsymbol{Q}_2$, $\boldsymbol{Q}_5$, $\boldsymbol{Q}_7$, and $\boldsymbol{Q}_8$ miss two solutions each. The nearest solution to $\boldsymbol{P}^*$ is $\boldsymbol{Q}_6$ with a mutation rate of 0.94, bit-turn probability of 0.56, and a crossover rate of 0.19 followed by $\boldsymbol{Q}_3$ with a mutation rate of 0.91, bit-turn probability of 0.34, and a crossover rate of 0.72 . The worst vector is $\boldsymbol{Q}_8$ with a mutation rate of 0.95, bit-turn probability of 0.83, and a crossover rate of 0.55.

$\boldsymbol{Q}_1$	$\boldsymbol{Q}_2$	$\boldsymbol{Q}_3$	$\boldsymbol{Q}_4$	$\boldsymbol{Q}_5$	$\boldsymbol{Q}_6$	$\boldsymbol{Q}_7$	$\boldsymbol{Q}_8$	$\boldsymbol{Q}_9$	$\boldsymbol{Q}_{10}$	CW	$\boldsymbol{P}^*$
0.001		0.0003	0.0053	0.0001	4E-05	0.024	0.039	0.0189	0.0025	100	4.E-05
4.9	2.4	3.8	2.5	4.2	3.1	2.6	2.4	2.3	5.4	600	2.3
42.9	42.7	43.2	41.6	43.7	43.6	42.8	44.5	47.8	43.1	700	41.6
70.0	67.4	66.6	65.9	66.1	66.1	66.4	66.6	67.2	67.5	800	65.9
214.7	212.5	220.0	215.5	218.0	219.1	219.7	213.9	226.1	212.5	1100	212.5
335.3	335.7	335.8	335.3	336.0	335.1	308.6	336.8	308.0	337.1	1200	308.0
350.8	350.4	350.2	358.8	350.5	354.7	359.6	353.1	361.4	351.4	1300	350.2
			395.5			395.4	395.6	398.8	397.3	1400	395.4
443.4	443.0	442.6	440.9	442.6	443.4			445.2	443.6	1800	440.9
447.4	449.7	447.8	450.2	448.9	447.4	448.1	448.6	447.4	448.8	1900	447.4
566.5	566.9	566.1	566.7	567.0	568.8	566.6	567.4	568.8	571.0	2000	566.1
683.4	684.6	684.4			683.6	683.9	683.3		688.2	2100	683.3
686.2	686.2	686.7	686.5	687.4	686.5			686.9		2400	686.2
688.7	688.7	689.0	688.9	688.9	689.1	689.0	689.6	689.6	691.0	2500	688.7

Table 3.15 Quality values after a 90° clockwise rotation at (50,20)

Table 3.16 shows the quality values and captured weights after $\boldsymbol{S}$ is rotated counter-clockwise by 90 degrees at (25, 45) (see Figure 3.9). Vector $\boldsymbol{Q}_8$ has the complete set of solutions and all the other vectors have one or more missed solutions. Solution sets $\boldsymbol{Q}_1$, $\boldsymbol{Q}_4$, $\boldsymbol{Q}_7$, and $\boldsymbol{Q}_{10}$ miss only one solution each and vectors $\boldsymbol{Q}_2$, $\boldsymbol{Q}_3$, $\boldsymbol{Q}_6$, and $\boldsymbol{Q}_9$ miss two solutions each. Vector $\boldsymbol{Q}_5$ misses three solutions. Following the performance criteria described previously, the nearest solution to $\boldsymbol{P}^*$ is $\boldsymbol{Q}_8$ with a mutation rate of 0.95, bit-turn probability of 0.83,

and a crossover rate of 0.55 followed by $\boldsymbol{Q}_1$ with a mutation rate of 0.69, bit-turn probability of 0.81, and a crossover rate of 0.44. The worst vector is $\boldsymbol{Q}_3$ with a mutation rate of 0.91, bit-turn probability of 0.34, and a crossover rate of 0.72.

$\boldsymbol{Q}_1$	$\boldsymbol{Q}_2$	$\boldsymbol{Q}_3$	$\boldsymbol{Q}_4$	$\boldsymbol{Q}_5$	$\boldsymbol{Q}_6$	$\boldsymbol{Q}_7$	$\boldsymbol{Q}_8$	$\boldsymbol{Q}_9$	$\boldsymbol{Q}_{10}$	CW	$\boldsymbol{P}^*$
0.017	0.0001	0.0195	0.0004	4E-05	0.0001	0.11	0.038	0.0014	0.0021	600	4.E-05
33.2	29.9	29.4	29.9	31.6	31.4	30.1	33.1	33.6	43.1	700	29.4
66.4	66.1	66.2	66.1	66.8	66.0	67.5	70.3	67.6	68.6	800	66.0
216.5	218.8	214.1	216.4	215.8	218.0	213.0	213.3	223.9	214.2	1100	213.0
253.7	307.8	335.3	307.8	307.7	307.7	307.9	253.7	312.4	309.6	1200	253.7
354.0	359.8	351.7	360.9	360.1	359.6	361.3	360.9	361.3	352.1	1300	351.7
401.2			396.2			395.9	399.0	400.6	401.8	1400	395.9
		443.9	443.1			444.7	443.6	442.3	441.1	1800	441.1
489.9	458.4	448.1	463.0	448.0	447.0	448.0	448.7	447.8	447.7	1900	447.0
566.2	567.1	566.2	566.4	566.1	566.1	566.6	566.6	566.2	566.7	2000	566.1
686.5	683.1				684.9		685.0			2100	683.1
686.7	685.8	686.4	687.1	686.4	686.4	686.5	686.9		686.9	2400	685.8
689.0	688.7	688.8	689.2	689.1	688.7	688.8	688.8	689.2	690.3	2500	688.7

Table 3.16 Quality values after a 90° counter-clockwise rotation (25, 45)

The results show there is no single parameter combination that dictates the generation of the best nondominated solutions. This means a parameter combination that finds the best solution in one case area does not necessarily find the best solution in another case area. There are also cases wherein not even one combination produced a complete set of solutions from all ten parameter combinations. This means the ten best parameter combinations in the original configuration does not always generate the best solutions when changes to the location of the area are introduced. However, the multiobjective evolutionary algorithm can generate the best solution $\boldsymbol{Q}^*$ from an ensemble of only ten parameter combinations by combining the best solutions for each combination.

The tables also show that higher mutation probabilities (> 60%) and lower crossover probabilities (<55%) produce the nearest nondominated solution to $\boldsymbol{P}^*$. On the contrary, high crossover rates (> 50%) often produce the worst nondominated solutions among the parameter combinations when using arithmetic crossover and non-uniform mutation.

3.4 Summary

This chapter investigates the performance of several MOEAs in discovering a set of solutions to the competitive facility location problem and a sensitivity analysis of their solutions in relation to their input parameters. In terms of the optimality of solutions, the MOEAs are able to find Pareto-optimal solutions in the discrete-valued objective (captured weights) search space but shows inferior Pareto-optimal solutions to the real-valued objective (facility quality).

In terms of runtime complexity, the MOEAs run in polynomial time but are computationally expensive in terms of repeated executions for sensitivity analysis. The advantage of using MOEAs in finding efficient solutions to the CFLP is that it does not suffer from difficulties related to geometric procedures; they are extensible to planar problems with a non-Euclidean metric.

The sensitivity analyses show that the mutation rate causes much of the variation in the output and requires a high probability value in order to generate CFL solutions near the Pareto-optimal front. When using average crossover, the crossover rate has the least influence on the output and requires low probability values, and when using blend crossover, the crossover rate has the greatest influence and requires high probability values.

Section 3.4 examined the effects of changing the area location in finding solutions to the competitive facility location problem while preserving a small set of previously effective GA-parameter combinations. The results show that there is no single GA-parameter combination that dictates the generation of best solutions in the CFLP when the area parameters are altered. However, an ensemble of a few GA-parameter combinations is sufficient to find near-Pareto-optimal solutions when area parameters are changed. The MOEA in this case, is robust as long as the mutation probability and the bit-turn probability are set to a probability of more than 60% while keeping the crossover probability low.

Chapter 4

The Container Location Model: An optimization problem with a fixed-length string of discrete variables

4.1 Introduction

Containers are in the market for international conveyance of freight for more than 40 years and have got more acceptance due to the concept of a unit-load. The breakthrough was achieved in this time period with high investments in specially designed vessels, in the availability of (purchased or leased) containers and in adaptable seaport terminals with suitable equipment.

A container terminal serves as an interface connecting container vessels on sea with trucks on land. It provides loading and unloading services for the container carriers. On top, the terminal serves as a temporary storage space for containers between two journeys on carriers. Due to increasing container traffic, sometimes thousands of trucks pass the gates of the terminal, of which probably the surface area might be relatively small. This combination requires for highly efficient handling of the operations. The need for optimization in container

terminal operations has become important because the logistics of large container terminals reaches a high level of complexity (Steenken, 2004).

An overall optimization model of the operational decisions in a container terminal is not easy, and maybe not feasible, because of the complexity and the multi-criteria nature of the problem. In fact, decisions are required on berth allocation, schedule and stowage plan of the vessels, quay crane allocation, storage space allocation, location assignment of containers in blocks, etc. It looks more feasible to split the overall problem into several sub-problems. This chapter concentrates on the storage location of outbound containers in the yard.

Quay cranes discharge inbound and transit containers from vessels and load outbound and transit containers to vessels. The storage yard is made up of blocks of containers. In a block, containers are stored, usually in six lanes side by side, with each lane including 20 or more container stacks. The level at which a container is stacked is called a tier. Container stacks typically have four or five tiers.

Performance indicators of a container terminal have been defined, measuring productivity, resource utilization and customer satisfaction. Two examples of performance indicators taken as objectives are (1) to minimize the vessel berthing time, which is a service measure of the terminal, or (2) to maximize the throughput of the quay cranes, which is a productivity measure of the terminal.

The berthing time of a vessel consists of several components like waiting, berthing, unloading, loading and departing. The times spent in each of those components are related to availability of required the resources. Quay cranes discharge and load the containers. Internal trucks provide transportation of containers between the quay cranes and the storage blocks. External trucks bring outbound containers into the yard and pick up inbound containers from the yard. Yard cranes handle the containers in the storage blocks. They load containers from trucks and stack them onto blocks, and retrieve containers from blocks and load them onto trucks. Decisions about the storage of containers in the yard directly affect the workload of yard cranes and the travelling distances of the internal trucks and indirectly affect the efficiency of the quay cranes.

An overall optimization model of the operational decisions in a container terminal is not easy, and maybe not feasible, because of the complexity and the

multicriteria nature of the problem. In fact, decisions are required on berth allocation, schedule and stowage plan of the vessels, quay crane allocation, storage space allocation, location assignment of containers in blocks, etc. It looks more feasible to split the overall problem into several sub-problems. This chapter concentrates on the storage location of outbound containers in the yard.

This chapter focuses on an improved use of the storage area by reducing the time for the yard cranes to transfer containers from the storage area to the marshalling area for loading onto the vessels. The objective is to minimise the time the vessels spend at the berth. To obtain this objective the time spent transferring containers from a storage area to the vessel (i.e. the sum of set up times and travel times for all containers) needs to be minimised.

While the model may be formulated as a mixed integer linear program, its computational complexity increases exponentially with the number of containers in the schedule and is known to be NP-hard (Kozan and Preston 2001). A genetic algorithm is formulated as a tool to generate good solutions.

4.1.1 Problem Formulation

In practice wherever possible, storage yards are grouped into two categories, import and export yard blocks. Import containers are unloaded from container ships from overseas and continue through inland transport while export containers are loaded on ships for overseas. Arrival of import containers is foreseeable but their departure is unpredictable. Because of the randomness in their departure, import containers are not stacked so high. Departure of export containers is usually connected to a ship and can be stacked more efficiently. In a generic stacking problem, items numbered from 1 to n arrive at a set of k stacks in some permutation. Eventually, all items need to leave the stacks in correct order with a minimum number of reshuffles. König and Lübbecke (2008) show that to approximate the minimum number of reshuffles is NP-hard. This chapter mainly addresses the problem of stacking export containers and shows how loading schedules affect handling time in seaport terminals.

According to Kozan and Preston (2001), the objective of the container location model (CLM) is to determine the optimal storage strategy for various

handling schedules such that the containers must be stored in a manner that minimizes the amount of handling time. They define the optimization problem as

$$\text{Minimize } \sum \left(travelling\_time_i + setup\_time_i \right) \tag{4.1}$$

Subject to the constraints

1. Only one container can be stored in a given storage position,
2. A yard machine is scheduled to handle one container at a given time,
3. A container arriving at time *t'* cannot be stored under a container arriving at time *t* such that $t < t'$.

The formulation makes use of the following definitions:

$Travelling\_time$ is the time required to transport container *i* between the storage area, marshalling area, track area and inter-modal terminal and is defined as

$$Travelling\_time = lock + \frac{x_i RW + y_i CW}{V} + lock \tag{4.2}$$

and

$Setup\_time$ is the time required to access the desired container *i* at the storage area and defined as

$$Setup\_time = \begin{Bmatrix} z_i(\ 4lock + 2move\ ) + 2lock + move \\ 0 \end{Bmatrix} \tag{4.3}$$

making use of the following variables

$lock$ is the time needed for the yard machinery to lock on to a container before picking it up and to unlock a container after moving it.

$move$ is the time required to move containers to an adjoining position to access containers below.

CW is the length of a column of the storage area

RW is the width of a row of the storage area

x_i is the row where container is stored

y_i is the column in the storage area where container is stored

$z_{i,t}$ is the vertical storage position of container

V is the velocity of the yard machine.

Figure 4.1 shows a picture of 3-tier stack and the margin between set up time and travel time. In this example, *container B* needs to be transferred from the storage area for loading. Since *container A* is on top of *container B*, the total

setup time includes the time to re-handle *container A* plus the setup time for *container B*. The re-handling time of *container A* is the time to remove it from the stack and the time to replace it on the stack. Looking at Figure 4.3 the yard machine *locks* on A, moves it to the ground, and *unlocks* A in the first step. Then the yard machine *locks* on B, moves B to a transport vehicle, and *unlocks* B. In the third step, the yard machine *locks* on A, places it on top of C, then *unlocks* A. The travelling time of *container B* starts only after the setup process is completed.

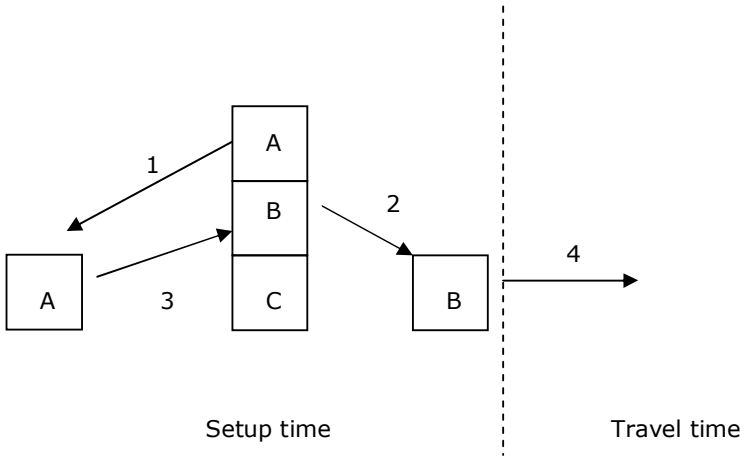

Figure 4.1 Setup time and travel time.

The conflicting objectives of maximizing the use of storage space and minimizing unproductive moves in a stacking strategy seem to be an interesting bi-objective optimization problem, but further examination shows that the efficient use of storage space is dependent on the number of containers in a yard block. For example, given a set of containers in a block, the percentage of space used by these containers does not change when they are relocated in the same block. However, proper arrangement of containers in a block may improve the total handling time during loading. The bi-objective problem reduces to a single-objective optimization problem of minimizing the total handling time of containers in a block by finding better arrangements of the containers before they are loaded. In return, this leads to a multiobjective problem of finding the minimum total handling time while minimizing the number of relocated containers (*re-handled containers*).

This chapter evaluates two experiments. The first experiment is a single-objective optimization problem of minimizing the total handling time as defined by Kozan and Preston (2001). The second experiment is a bi-objective optimization problem of minimizing *total handling time* while minimizing *re-handled containers*. The bi-objective problem is defined as:

$$\text{Minimize } \sum_i travelling\_time_i + setup\_time_i \qquad (4.4)$$
$$\text{Minimize } (re\text{-}handled\ containers)$$

4.1.2 Literature Review

Containers may be arranged according to container information such as size, weight, departure time, and destination vessel among others. In category stacking, containers of the same category are stacked together in the same yard bay or in contiguous yard bays. In the residence time strategy, a container is stacked relative to its departure time, i.e. containers that leave earlier are stacked on top of containers that leave at a later period. Steenken et al. (2004) presents two methods, (1) storage planning wherein storage space is allocated before the ship's arrival and, (2) scattered stacking wherein containers are assigned to a berthing place before a ship's arrival. Dekker et al. (2006) state that *scattered stacking* results in higher yard utilization and significantly reduces the number of unproductive moves (reshuffles).

Chen et al. (1999) provided a description of unproductive moves in port terminals, whereas Murty (2005) defines an objective function of minimizing reshuffles and maps a solution that is analogous to a bin-packing problem. In category stacking, weight is a useful criterion as heavy containers are usually stored deep in a ship. Kim et al. (2000) derived decision rules using weight groups for determining the storage slots of arriving containers through dynamic programming. Kim and Kim (1999) relate the efficiency of loading operations to a loading sequence derived from a routing solution of transfer cranes. Kim and Bae (1998) describe an approach to move containers efficiently from a current layout to an ideal layout by decomposing the container problem into two stages – a bay matching problem and a sequencing problem. They propose that heuristic techniques should be developed for calculations that are more efficient.

Kozan and Preston (2001) describe the effects of loading schedules and storage utilization in minimizing turn-around time of container ships though genetic algorithms. Kim et al. (2004) define a beam search algorithm to minimize handling time of transfer cranes and quay cranes for export containers. Other studies show the importance of improving space allocation methods of export containers in the management of port terminals (Kim et al. 2003, Zang et al. 2003, Chen et al. 2004). Pangilinan and Janssens (2009) evaluated two types of loading schemes for export containers using a mutation-only genetic algorithm. Their results show that a last-come-first-served (LCFS) loading schedule is superior to a first-come-first-served (FCFS) loading schedule and the employment of a GA in both FCFS and LCFS schedules show improvement in their handling times.

The current study differs from the earlier study by Pangilinan and Janssens (2009) and by Kozan and Preston (2001) in terms of the following: (1) the interactions of the genetic operators are evaluated, (2) the performance of two MOEAs is compared, and (3) results of both the single-objective and bi-objective CLM problems are presented.

4.2 Multiobjective Evolutionary Algorithm

4.2.1 Genetic Algorithm for the CLM

Genetic Representation. A chromosome or an individual consists of integer-valued elements that form a linear string, which contains the location of a container in the storage yard. The storage yard is represented as a 3-dimensional array of storage locations (see Fig. 4.2).

The index of each element *i* in the string represents the ID of a container *i*. The value of an element at index *i* represents the storage location of container *i* in the storage yard. The length of the chromosome is fixed and may not be greater than the number of containers, *n*. A chromosome is randomly generated and an integer with a value less than the maximum size of the storage block is assigned to each position of the chromosome. The assignment of storage locations ensures that the CLM constraints are met, e.g. a container that arrives at time *t* cannot be stacked on top of a container that arrives at time *t*+1.

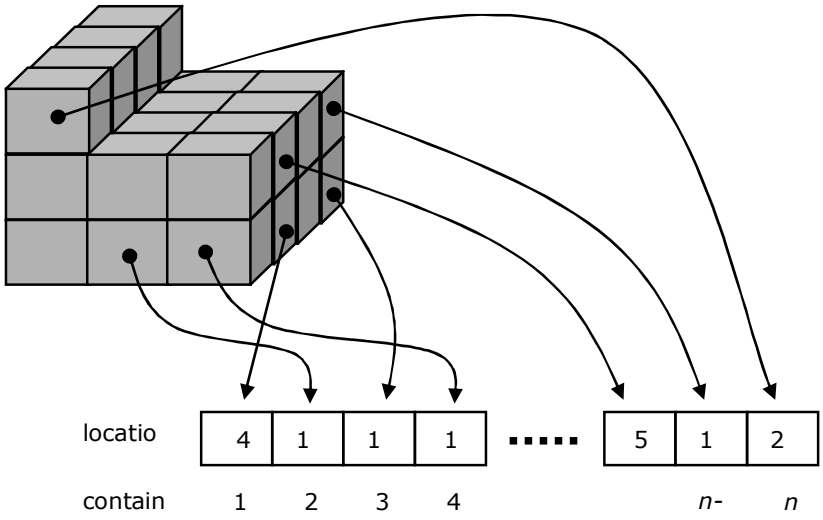

Figure 4.2 Chromosome representation

Genetic Operators. The crossover operator is an adaptation of the one-point crossover. For each pair of individuals, a locus is randomly selected and the values from the locus to the end of the chromosomes are is interchanged (see Section 2.2.3). Repairs are done to ensure that only one container is assigned to a single storage space and that a container occupies a location that is on the ground or on top of another container. In the mutation operator, a locus is randomly selected from the chromosome and the value of the locus is replaced by a new vacant location in the storage block.

Fitness and Selection. In the single-objective case, the fitness of an individual is computed based on *total handling time* of containers, i.e. the sum of the *travelling_time* and *setup_time* in loading the necessary containers on a container ship. The individual that has a smaller fitness value is the better solution. In the bi-objective case, the fitness is evaluated in terms of Pareto-dominance of *total handling time* and *re-handled containers*. The selection in both cases uses SPEA2 (Zitzler et al. 2002) and NSGA-II (Deb et al. 2002) to find nondominated solutions to the CLM problem.

4.2.2 Experiments and Results

SINGLE-OBJECTIVE CASE

The experiment compares the performance of two export-container loading schemes namely last-come-first-served (LCFS) and first-come-first-served (FCFS). The study evaluates the viability of a genetic algorithm in optimizing the handling time using the LCFS and FCFS schedules, and evaluates the effect of the genetic operators, such as crossover and mutation, on the output. The experiment has the following assumptions:

1. A single yard machine is considered.
2. All containers for loading are located in one block and near the departure ship.
3. One block is composed of five yard bays. One yard bay is 6 rows wide and 4 tiers high.
4. All containers are loaded in only one departure ship and a maximum of 120 containers can be loaded.
5. V=0.33 m/s, RW=2.4 m, CW=6.1 m, $lock$=60 sec, and $move$=60 sec. The values assigned to these variables are based on actual data.

Ten configurations for the storage utilization are used, i.e. from 10% to 100% usage. For each usage configuration, 100 instances of container arrangements are randomly generated. Figure 4.3 shows a comparison of the average handling time of both the FCFS and the LCFS loading schedule with regard to storage use from 10% to 100%. The LCFS schedule shows a linear behavior and its slope is small whereas the FCFS schedule produces a curve wherein the handling time increases considerably as the space usage increases. The performance of the LCFS loading becomes obviously better whereas the FCFS schedule becomes worse as the utilization of the storage yard increases. This is because the storage of containers emulates a stacking structure.

The storage block in this experiment contains 120 storage spaces, i.e. six rows, 5 columns, and 4 tiers. This means that one tier can accommodate 30 containers. In this context, the performance of FCFS is ideal only if the containers are placed in one tier. But if the number of containers to be stored is

more than 25% of the storage block capacity then the LCFS loading scheme will always perform better than the FCFS.

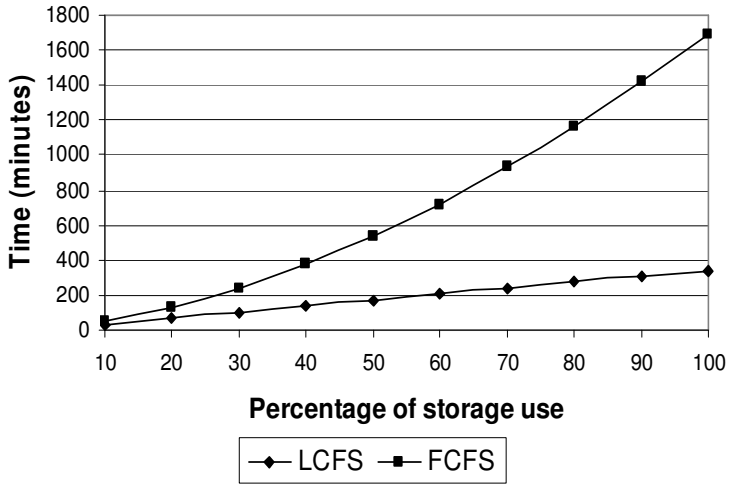

Figure 4.3 Handling time, LCFS vs. FCFS

With regard to the EAs, four configurations were used, (1) SPEA2 with crossover only, (2) SPEA2with mutation only, (3) NSGA-II with crossover only, and (4) NSGA-II with mutation only. Each of the above EA conditions was tested on both the FCFS and LCFS loading schedules. An initial population is set to 36 individuals and a run of 20 generations. The population size of 36 was estimated from Deb (2001) with the assumption that 30% of nondominated paths are present in the initial population.

Table 4.1 presents a comparison of the best-average handling times of 50 random container configurations per percentage usage. The table presents results from an FCFS schedule, an EA-mutation-only (EAM-FCFS), and an EA-crossover-only (EAX-FCFS) schedule. The results show that the EA methods have shorter handling times for each storage configuration than FCFS. This means reshuffling (re-handles) of the containers from their initial position improves the total handling time of all the containers for loading. There are no significant differences between the results of mutation-only and crossover-only schemes in both SPEA2 and NSGA-II (F-test with 5% significance level). However, there is a pattern in the solutions. The crossover-only algorithm performs better for both MOAEs when the space usage is low (≤ 60% in NSGA-II

and ≤ 30% in SPEA2). The mutation-only algorithms perform better when the space usage increases beyond these limits. This shows that SPEA2 and NSGA-II use the variation operators differently even when their parameter settings are the same.

Table 4.1 also shows that the NSGA-II results are smaller than the SPEA2 results, which implies that NSGA-II finds faster total handling times than SPEA2 in all FCFS configurations for the single-objective CLM problem. The best handling times are in **bold**.

Space Usage (%)	FCFS (min)	EAM-FCFS SPEA2 (min)	EAM-FCFS NSGA-II (min)	EAX-FCFS SPEA2 (min)	EAX-FCFS NSGA-II (min)
10	32.50	31.41	30.77	30.35	**28.96**
20	95.70	91.29	89.16	85.89	**80.27**
30	194.24	189.93	188.22	186.85	**180.00**
40	320.71	316.64	313.51	316.72	**311.89**
50	484.21	479.97	477.15	481.62	**476.27**
60	668.93	664.48	659.85	667.87	**663.64**
70	881.77	877.11	**872.35**	880.43	876.50
80	1118.08	1114.67	**1110.42**	1116.70	1114.77
90	1379.01	1377.55	**1376.26**	1378.46	1377.91

Table 4.1 Comparison of solutions in an FCFS scheme

Table 4.2 shows the EA methods have shorter handling times for each storage configuration in an LCFS schedule. Again, this shows that arranging the containers to better positions improves the total handling time. Comparing the EA results using the F-test with 5% significance level, there are no significant differences between the values of mutation-only and crossover-only schemes in both SPEA2 and NSGA-II. However, the results of NSGA-II are better than the results of SPEA2 in both mutation-only and crossover-only algorithms. This means that NSGA-II finds faster total handling times than SPEA2 in all LCFS configurations for the single-objective CLM problem.

Space Usage (%)	LCFS (min)	EAM-LCFS SPEA2 (min)	EAM-LCFS NSGA-II (min)	EAX-LCFS SPEA2 (min)	EAX-LCFS NSGA-II (min)
10	29.68	26.16	**25.51**	26.81	25.62
20	62.36	57.55	55.74	57.99	**55.70**
30	95.32	90.58	**88.10**	91.08	88.61
40	128.86	124.30	**122.09**	124.56	122.22
50	162.27	158.47	156.52	158.24	**155.68**
60	196.59	193.24	191.12	192.86	**190.27**
70	230.41	227.64	225.14	227.35	**225.04**
80	265.17	262.64	260.80	262.27	**260.41**
90	300.32	298.39	297.38	298.14	**296.93**

Table 4.2 Comparison of solutions in an LCFS scheme

MULTIOBJECTIVE CASE

The following experiment compares the performance of SPEA2 and NSGA-II in the bi-objective CLM problem as defined in (4.4). Their averages are compared in terms of handling time, re-handled containers, and the cardinality of their nondominated solutions. The averages per storage-use configuration are taken from 50 runs. The initial population is set to 36 individuals and a run is set at 20 generations. The mutation and crossover rates are set at 0.50. The individuals in the initial population in each run are identical in order to count the number of re-handled containers after each run.

Table 4.3 shows the average results of SPEA2 and NSGA-II from an FCFS loading scheme in terms of total handling time, quantity of re-handled containers, and the cardinality of the nondominated sets. The best handling times and the least number of re-handled containers are in **bold**. The second column lists the values of total handling time when there is no reshuffling of containers before loading. The results show using the F-test with 5% significance level, that there are no significant difference in the total handling times, in the number of reshuffled containers, and the size of their nondominated sets between SPEA2 and NSGA-II. The results show that NSGA-II generates lesser re-handled containers and smaller solution sets. The averages in the number of re-handled containers in both MOEAs are small and increase as the space usage increases. The size of the solution set behaves similarly.

Storage Use %	FCFS	SPEA2-FCFS			NSGA-II-FCFS		
		Handling Time (min)	Re-handled Containers	Number of Solutions	Handling Time (min)	Re-handled Containers	Number of Solutions
10	49.83	47.55	1.09	5.04	**46.35**	**0.70**	2.24
20	130.16	125.29	1.32	6.62	**124.02**	**0.88**	2.58
30	243.48	**237.66**	1.68	6.88	237.76	**1.20**	3.00
40	374.56	368.84	2.02	7.50	**368.47**	**1.28**	3.02
50	540.16	**531.23**	2.30	8.42	533.15	**1.72**	3.36
60	727.26	**721.77**	2.90	9.36	724.63	**2.56**	4.63
70	935.08	927.75	3.33	9.86	**927.59**	**2.71**	4.76
80	1163.53	1159.36	3.81	9.62	**1157.28**	**2.91**	4.70
90	1418.7	**1414.41**	4.70	8.80	1415.74	**2.57**	4.36

Table 4.3 Comparison of bi-objective solutions in an FCFS scheme.

Table 4.4 shows the averages of handling time, re-handled containers, and the cardinality of nondominated solutions in SPEA2 and NSGA-II using an LCFS loading scheme. The results show using the F-test with 5% significance level, that there are no significant difference in the total handling times, in the number of reshuffled containers, and the size of their nondominated sets between SPEA2 and NSGA-II. Unlike the results in Table 4.3, NSGA-II generates less re-handled containers, smaller solution sets, and faster handling times in all storage configurations. The size of the solution sets in SPEA2 decreases gradually as the space usage increases from 20% to 90%.

Storage Use %	FCFS	SPEA2-LCFS			NSGA-II-LCFS		
		Handling Time (min)	Re-handled Containers	Number of Solutions	Handling Time (min)	Re-handled Containers	Number of Solutions
10	34.47	31.38	3.04	15.76	**30.98**	**2.83**	6.32
20	69.26	65.53	3.57	17.94	**65.15**	**3.36**	7.46
30	103.26	99.76	3.80	17.16	**99.42**	**3.63**	7.84
40	137.56	133.99	3.97	16.06	**133.72**	**3.67**	7.34
50	171.88	168.68	3.92	14.60	**168.33**	**3.64**	7.18
60	206.34	203.44	4.06	13.00	**203.09**	**3.86**	7.00
70	240.28	237.92	4.02	11.16	**237.78**	**3.24**	5.84
80	274.63	272.32	4.87	10.30	**272.31**	**3.38**	5.42
90	309.18	307.22	5.87	9.10	307.66	**3.13**	4.78

Table 4.4 Comparison of bi-objective solutions in an LCFS scheme.

NSGA-II behaves similarly when space usage is ≥ 30%. On the other hand, the quantity of re-handled containers in SPEA2 increases as the space usage increases.

Tables 4.3 and 4.4 show that the number of re-handled containers and the size of solution sets of the LCFS loading method are larger than the values of the FCFS loading method. However, the handling times of the LCFS are far smaller than handling times of the FCFS scheme. The tables above show that NSGA-II is a better algorithm than SPEA2 in both the single-objective and bi-objective CLM cases.

4.3 Sensitivity Analysis

The sensitivity analysis investigates the effects of the recombination and mutation operators on the handling time of containers in the case of the single-objective CLM. In the bi-objective case, a sensitivity analysis on the proximity of nondominated solutions to the Pareto-optimal set is not possible since the Pareto-optimal set is unknown. However, a sensitivity analysis on the cardinality of the nondominated sets is feasible.

4.3.1 Experiments and Results

SINGLE-OBJECTIVE CASE

The sensitivity analysis for the single-objective case utilizes 12 configurations according to space usage. These configurations help illustrate the differences between SPEA2 and NSGA-II with regard to the influence of their genetic operators. The experiment investigates space usage of 30 %, 60%, and 90% in an FCFS loading scheme. The output variable is the sum of the handling time of containers. The input parameters are *mrate*, *xover*, and *maxgen*, the mutation probability, recombination probability, and number of generations respectively. The initial population is 36 with the assumption that 30% of nondominated solutions are in the initial population as estimated from Deb (2001). In each loading configuration, there are 128 combinations of the input parameters generated by Sobol's method and their distributions are listed in Table 4.5.

Input factor	Description	Probability Distribution
mrate	Mutation probability	Uniform(0,1)
xover	Recombination probability	Uniform(0,1)
maxgen	Number of generations	Uniform(1,10) Uniform(11,20)

Table 4.5 Input factors and probability distributions

Table 4.6 shows the results of the sensitivity analysis for an FCFS schedule. Two configurations in the number of generations are provided to show the changes in the sensitivity indices between generations 1 to ten and generations 11-20. The figures in **bold** are the indices of the input parameters with high main effects. The second and third column shows the sensitivity indices of the input parameters when the number of generations *maxgen* is between 1 and 10. The fourth and fifth column shows the sensitivity indices of the input parameters when the number of *maxgen* is between 11 and 20. The sensitivity indices show that the interaction of the input parameters *mrate* and *xover* is high while their main effects are low in most of the configurations. The interaction of *maxgen* with the other input parameters in the initial generations is high and diminishes as the MOEA run progresses. This means the selection operator of either SPEA2 or NSGA-II has much influence in finding nondominated solutions in the initial stages of the MOEA. However, in the latter generations, the main effect of mutation increases when the space usage is 60% or higher. This implies that when the space usage is high, the mutation operator has the largest influence in finding different handling times during the final generations of the optimization run in an FCFS schedule. The first-order indices of the crossover rate are small in most instances except when the space usage is at 30 % and *maxgen* is between 1to 10. This means that the crossover operator does not explore the search space as much as the mutation operator in an FCFS schedule.

The SPEA2 sensitivity indices are zeroes when the space usage is 30% and 60% as shown in second column. The zero values imply that there is no variation in the output during generations 1 through 10. Similarly when *maxgen* is between 11 and 20 and space usage is 30%, the SPEA2 indices are negative. Negative indices also mean that the inputs have no influence on the output. When *maxgen* is between 11 and 20 and space usage is 60%, the SPEA2 indices are small. This means that the sum of handling times generated by SPEA2 do not vary much after the first few generations when space usage is below 60%.

Single-objective case (FCFS)	1-10 generations		11-20 generations	
	SPEA2	NSGA-II	SPEA2	NSGA-II
30% space usage				
Sobol' first-order indices				
mrate	0.00	-0.053	-0.091	-0.312
xover	0.00	0.439	-0.034	**0.067**
maxgen	0.00	**0.537**	-0.06	-0.063
Sobol' total-order indexes				
mrate	0.00	0.343	0.884	0.851
xover	0.00	0.481	1.296	1.071
maxgen	0.00	1.187	0.131	0.085
60% space usage				
Sobol' first-order indices				
mrate	0.00	0.00	0.043	**0.342**
xover	0.00	-0.543	0.029	0.0038
maxgen	0.00	-0.382	0.00	0.052
Sobol' total-order indexes				
mrate	0.00	0.00	1.107	0.566
xover	0.00	0.341	1.060	0.374
maxgen	0.00	0.503	0.00	0.119
90% space usage				
Sobol' first-order indices				
mrate	-0.010	0.342	0.339	**0.820**
xover	0.238	0.118	-0.0205	0.152
maxgen	0.044	**0.500**	-0.0393	0.133
Sobol' total-order indexes				
mrate	0.145	0.609	0.564	1.007
xover	0.461	0.423	0.212	0.269
maxgen	0.204	0.897	0.006	0.325

Table 4.6 Sobol' indices for an FCFS configuration (single-objective).

Figure 4.7 shows the sensitivity indices for an LCFS schedule. The figures in **bold** are the input parameters that have the high main effects. In generations 1-10, the parameter *maxgen* has the greatest influence in the variation of handling time of containers regardless of space usage. In generation 11-20, the mutation rate has the greatest influence in the variation of the output when the

space usage is 30% whereas the crossover rate has the greatest influence when the space usage is 60% and 90%. The sensitivity indices of *maxgen* decreases as the number of generations increase whereas, the influence of the mutation and crossovers rates increase as the number of generations increase. This means that the mutation and crossover operator are responsible for the exploration of new solutions after the tenth generation.

Single-objective case (LCFS)	1-10 generations		11-20generations	
	SPEA2	NSGA-II	SPEA2	NSGA-II
30 % space usage				
Sobol first-order indexes				
mrate	0.068	0.052	**0.833**	0.736
xover	0.477	0.284	0.430	0.171
maxgen	0.840	**0.930**	0.066	0.290
Sobol total-order indexes				
mrate	0.117	0.142	0.944	1.053
xover	0.627	0.336	0.622	0.499
maxgen	0.705	1.064	0.247	0.391
60% space usage				
Sobol first-order indexes				
mrate	0.442	0.268	0.148	0.642
xover	-0.136	0.009	-0.169	**0.751**
maxgen	0.838	**1.004**	0.001	0.176
Sobol total-order indexes				
mrate	0.782	0.300	0.812	0.675
xover	0.025	0.056	0.438	0.850
maxgen	1.019	1.305	0.246	0.206
90% space usage				
Sobol first-order indexes				
mrate	0.265	-0.192	0.332	0.335
xover	0.439	0.314	0.495	**0.477**
maxgen	**0.733**	0.639	0.150	0.187
Sobol total-order indexes				
mrate	0.349	0.109	0.728	0.595
xover	0.513	0.540	0.959	0.617
maxgen	0.882	0.992	0.249	0.168

Table 4.7 Sobol' indices for an LCFS configuration(single-objective)

With regard to the changes in space usage, the effect of the mutation rate for NSGA-II decreases as the space usage increases as shown in the fifth column. This pattern is the reverse of the pattern in Table 4.6.

MULTIOBJECTIVE CASE

The sensitivity analysis investigates the effect of the input parameters on the cardinality of the nondominated sets generated by the MOEA in both LCFS and FCFS loading schemes. The input parameters are *mrate*, *xover*, and *maxgen*, which are the mutation probability, recombination probability, and number of generations respectively.

Table 4.8 lists the Sobol' sensitivity indices of SPEA2 and NSGA-II for an LCFS loading schedule. The numbers in **bold** represent the largest first-order effects among the input parameters in generations 1 to 10 and in generations 11 to 20. The values are emphasized to show the changes in the influence of each parameter between generation 1-10 and generations 11-20. In the first 10 generations, *maxgen* contributes 64% to the variation in the output and it has the highest interaction effect in both SPEA2 and NSGA-II. However, this condition changes in generations 11-20 as its value decreases. The main effect of the mutation rate is small in generations 1-10 and increases its influence from generations 11-20 as shown in column four and five. This means that the mutation operator is responsible mostly for exploring the dominated sets in generations 11-20 in an FCFS schedule. The zero indices under column five means that size of the nondominated sets generated by NSGA-II does not change from generation 11-20.

Table 4.9 lists the Sobol' sensitivity indices of SPEA2 and NSGA-II for an FCFC loading schedule. The numbers in **bold** represent the largest first-order effects among the input parameters in generations 1 to 10 and in generations 11 to 20. The figures in Table 4.9 show that the mutation rate (in SPEA2) mostly affects the variation in the size of the nondominated sets from generation 11-20 while *maxgen* has the largest first-order indices from generations 1 to 10, which are similar to the results in Table 4.8

Multiobjective case (LCFS)	1-10 generations		11-20generations	
	SPEA2	NSGA-II	SPEA2	NSGA-II
30 % space usage				
Sobol first-order indexes				
mrate	-0.015	0.348	0.450	**0.590**
xover	-0.099	0.135	-0.029	0.076
maxgen	0.508	**0.644**	0.096	0.051
Sobol total-order indexes				
mrate	0.510	0.417	0.820	0.845
xover	0.239	0.040	0.278	0.398
maxgen	0.832	0.787	0.218	0.155
60% space usage				
Sobol first-order indexes				
mrate	0.239	0.102	0.739	**0.804**
xover	0.064	0.110	0.215	-0.030
maxgen	**0.425**	0.418	0.259	-0.221
Sobol total-order indexes				
mrate	0.492	0.489	0.929	1.254
xover	0.080	0.120	0.156	0.354
maxgen	0.919	0.910	0.320	0.049
90% space usage				
Sobol first-order indexes				
mrate	0.098	0.000	**0.688**	0.000
xover	0.098	0.000	0.098	0.000
maxgen	**0.370**	1.067	0.078	0.000
Sobol total-order indexes				
mrate	0.772	0.000	0.860	0.000
xover	0.424	0.000	0.322	0.000
maxgen	1.109	1.067	0.291	0.000

Table 4.8 Sobol' indices for the LCFS configuration (multiobjective).

Multiobjective case (FCFS)	1-10 generations		11-20generations	
	SPEA2	NSGA-II	SPEA2	NSGA-II
30 % space usage				
Sobol first-order indexes				
mrate	0.100	0.071	**0.689**	0.134
xover	0.093	-0.198	0.469	-0.257
maxgen	0.029	**0.137**	0.065	0.025
Sobol total-order indexes				
mrate	0.672	0.725	1.027	1.068
xover	0.171	0.031	0.479	0.927
maxgen	0.483	0.575	0.131	0.282
60% space usage				
Sobol first-order indexes				
mrate	0.276	-0.110	**0.520**	0.438
xover	-0.040	-0.255	-0.244	0.279
maxgen	**0.391**	0.335	0.333	0.125
Sobol total-order indexes				
mrate	0.526	0.530	0.714	0.782
xover	0.115	0.339	-0.036	0.322
maxgen	0.854	0.933	0.523	0.050
90% space usage				
Sobol first-order indexes				
mrate	0.497	0.051	**0.748**	0.182
xover	-0.041	0.163	0.528	-0.512
maxgen	0.449	**0.706**	0.074	-0.285
Sobol total-order indexes				
mrate	0.833	0.412	0.943	1.331
xover	0.192	0.172	0.658	1.017
maxgen	0.681	0.919	0.132	0.413

Table 4.9 Sobol' indices for an FCFS configuration (multiobjective)

4.4 Summary

The chapter presents the performance of two modern MOEAs in relation to the container location problem. Each MOEA is evaluated in two dimensions: (1) loading scheme and (2) number of objective functions. Different results arise between an LCFS and an FCFS loading schedule in relation to their container handling times, reshuffles, storage-yard utilization. Similarly, different results appear between the single-objective and bi-objective cases.

In the single-objective case, the results show that the LCFS is superior to the FCFS loading schedule when the storage utilization is higher than 25%. In terms of the exploration and exploitation of the search space, the mutation-only GA finds better solutions when the space usage is high whereas the crossover-only GA finds better solutions when the space usage is low in an FCFS schedule. On the other hand, the results invert in an LCFS scheme, the crossover-only GA finds better solutions when the space usage is high and the mutation-only GA finds better solutions when the space usage is low. The results here contradict the recommendations of Deb and Agrawal (1999) that a selector-mutation GA does not work successfully in finding optimal solutions (see Section 2.2.7). As for comparing both MOEAs, NSGA-II performs better in both LCFS and FCFS scheduling and in the configurations of mutation-only and cross-only GA.

In the bi-objective case, the computations show similar results to the single-objective case. The handling times of the LCFS are far smaller than handling times of the FCFS scheme. However, the number of re-handled containers and the size of the nondominated sets of the LCFS loading method are larger than the results of the FCFS loading method. In both loading schemes, NSGA-II generates faster handling times, lesser re-handled containers, and smaller solution sets than SPEA2.

The sensitivity indices in the single-objective and bi-objective cases show dissimilarity between SPEA2 and NSGA-II. However, for both SPEA2 and NSGA-II, and in both scheduling schemes, the selection operator accounts for the variation in the size of the nondominated sets in the initial generations of the MOEA run and its effect diminishes in the later generations. Whereas the influence of the mutation operator is negligible at the early stages of the run but increases in the later generations of the MOEA run. The main effect of the crossover operator is small but has high interactions with the other operators.

Chapter 5

The Multiobjective Shortest Path Problem: An optimization problem with a variable-length string of discrete variables

5.1 Introduction

As in the case of the single-objective shortest path problem, the multiobjective shortest path problem has been studied extensively by various researchers in the fields of optimization, route planning for traffic and transport design (Granat and Guerriero, 2003) and information and communications network design (Gen and Lin, 2004; Kumar and Banjerie, 2003). The MSPP is an extension of the traditional shortest path problem and is concerned with finding a set of efficient paths with respect to two or more objectives that are usually in conflict. For example, the problem of finding optimal routes in communications networks involves minimizing delay while maximizing throughput or finding efficient routes in transportation planning that simultaneously minimize travel cost, path length, and travel time. The concept of optimization in the MSPP or in a multiobjective problem in general is different from the single-objective optimization problem wherein the task is to find a solution that optimizes a

single objective function. The task in a multiobjective problem is not to find an optimal solution for each objective function but to find an optimal solution that simultaneously optimizes all objectives. In addition, in most cases, no single optimal solution exists, only a set of efficient or nondominated solutions.

A variety of algorithms and methods such as dynamic programming, label selecting, label correcting, interactive methods, and approximation algorithms have been implemented and investigated with respect to the MSPP (Ehrgott and Gandibleux, 2000). The problem is known to be NP-complete (Garey, 1979). It has been shown that a set of problems exist wherein the number of Pareto-optimal solutions is exponential, which implies that any deterministic algorithm that attempts to solve it is also exponential in terms of runtime complexity at least in the worst case. However, some labeling algorithm studies (Gandibleux et al., 2006; Muller-Hannemann and Weihe, 2001) dispute this exponential behavior. They show that the number of efficient paths is not exponential in practice. Other authors avoid the complexity problem by developing new methods that run in polynomial time. For instance, Hansen (1979) and Warburton (1987) separately developed fully polynomial time approximation schemes (FPTAS) for finding approximately Pareto-optimal paths. Interactive procedures (Coutinho-Rodrigues et al., 1999; Granat and Guerriero, 2003) similarly avoid the problem of generating the complete set of efficient paths by providing a user-interface that assists the decision-maker to focus only on promising paths and identify better solutions according to preference. Pangilinan and Janssens (2007a) examined evolutionary algorithms for the MSPP to show that an MOEA runs in polynomial time.

Given the wealth of literature in multiobjective algorithms for the MSPP, there still seems to be a lack of reported review in evolutionary algorithm (EA) applications in relation to the MSPP. Several of the most recent alternative methods focus mostly on execution speed comparisons of different MSPP algorithms but analysis of the salient issues in multiobjective performance analysis such as runtime complexity, diversity, and optimality of nondominated solutions are almost omitted. In order to demonstrate a clearer picture of the advantages and disadvantages of EAs in optimization, this chapter attempts to investigate a multiobjective evolutionary algorithm as applied to the MSPP in terms of these measures. The current study extends the previous studies by Janssens and Pangilinan (2007a) by examining the performance of not only one MOEA but two MOEAs in order to compare the Pareto-optimality of their nondominated sets.

5.1.1 Problem Definition

Given a directed graph $\boldsymbol{G}$ = ($\boldsymbol{V}$, $\boldsymbol{E}$), where $\boldsymbol{V}$ is set of vertices and $\boldsymbol{E}$ the set of edges with cardinality $|\boldsymbol{V}| = n$ and $|\boldsymbol{E}| = m$ and a d-dimensional function vector $c:\boldsymbol{E}\rightarrow [\Re^{+}]^{d}$. Each edge e belonging to $\boldsymbol{E}$ is associated with a cost vector $\boldsymbol{c}(e)$. A source vertex s and a sink vertex t are identified. A path $\boldsymbol{p}$ is a sequence of vertices and edges from s to t. The cost vector $\boldsymbol{C}(\boldsymbol{p})$ for linear functions of path $\boldsymbol{p}$ is the sum of the cost vectors of its edges, that is $\boldsymbol{C}(\boldsymbol{p})=\Sigma_{e\in p}\ \boldsymbol{c}(e)$ whereas $\boldsymbol{C}(\boldsymbol{p})=\min_{e\in p}\ \boldsymbol{c}(\boldsymbol{e})$ for min-max functions. Given the two vertices s and t, let $\boldsymbol{P}(s,t)$ denote the set of all s-t paths in $\boldsymbol{G}$. If all objectives are to be minimized, a path $\boldsymbol{p}\in\boldsymbol{P}(s, t)$ dominates a path $\boldsymbol{q}\in\boldsymbol{P}(s, t)$ in and only if $C_i(\boldsymbol{p}) \le C_i(\boldsymbol{q})$, $i = 1,\ldots,d$ and we write $\boldsymbol{p}\preceq\boldsymbol{q}$. A path $\boldsymbol{p}$ is Pareto-optimal if it is not dominated by any other path and the set of nondominated solutions (paths) is called the Pareto-optimal set. The objective of the MSPP is to compute the set of nondominated solutions that is the Pareto-optimal set **P** of $\boldsymbol{P}(s, t)$ with respect to $\boldsymbol{c}$. The problem of the single-source multiobjective shortest path is to find the set of all paths from s to all other vertices t in $\boldsymbol{G}$, i.e. to find the Pareto-optimal set of $\boldsymbol{P}(s, t)$, $\forall t\in\boldsymbol{V}$ \{s}.

Figure 5.1 illustrates an example of a directed graph with three objective functions that have to be minimized simultaneously. The corresponding efficient paths from the source vertex s to all other vertices are : for t=1 that path is $s\rightarrow1$; for t =2, s$\rightarrow$1$\rightarrow$2; for t = 3, s$\rightarrow$1$\rightarrow$2$\rightarrow$3; for t = 4, $s\rightarrow1\rightarrow2\rightarrow4$; for t = 5, $s\rightarrow1\rightarrow2\rightarrow4\rightarrow5$ and $s\rightarrow1\rightarrow2\rightarrow3\rightarrow5$.

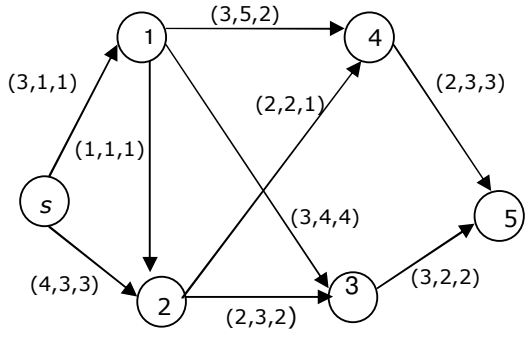

Figure 5.1 Graph with 3 objectives to be minimized.

5.1.2 Literature Review

Martins and Santos (1999) outline a labeling algorithm for the multiobjective shortest paths problem and present an analysis in terms of finiteness and optimality concepts and report that any instance of the MSPP is bounded if and only if there are no absorbent cycles in the network. They show a set of networks wherein the labeling algorithm only determines nondominated labels. On the contrary, Mooney and Winstanley (2006) state that Martins' labeling algorithm works well in theory but is prohibitive in practice in terms of its implementation due to memory costs.

A study by Gandibleux et al. (2006) reports a concise description of the MSPP and clearly narrates the most salient issues to its solution. Their study recalls Martins' labeling algorithm and attempts to improve it. Their new algorithm extends Martins' algorithm by introducing a procedure that can solve MSPP that have multiple linear functions and a max min function. Since it is an extension of Martins' algorithm, the generation of all nondominated paths remains intractable in polynomial time. However, experimental results of their study say otherwise. Their algorithm is tested on a variety of test instances and results show that in terms of size and complexity, optimizing simultaneous linear and max-min functions does not behave exponentially. They also show that their algorithm is not sensitive to different cost ranges and that density and network size increase the number of efficient solutions. An independent study by (2001) also shows that the cardinality of efficient paths in a bicriteria shortest path problem is not exponential as long as the instances are bounded by potential characteristics as defined in their experiment. They conclude with emphasis that it is still preferable to work with complete information rather than falling back on approximations.

Guerriero and Musmanno (2001) examine several label-selection and vertex-selection methods that can find Pareto-optimal solutions to the MSPP with respect to execution time. The performance of the different algorithms was measured using random and grid networks and results show that label-selection methods are generally faster than vertex-selection methods and that parallel computing is necessary in the design of efficient methods.

While some researchers focus on exhaustive solutions or on improvements thereof, other researchers are more concerned with better runtime solutions. Tsaggouris and Zaroliagis (2006) present an improved fully polynomial time approximation scheme (FPTAS) for the multicriteria shortest path problem and a new generic method for obtaining FPTAS to any multiobjective optimization problem with non-linear objectives. They show how their results can be used to obtain efficient approximate solutions to the multiple constrained path problems and to the non-additive shortest path problem. Their algorithm, which departs from earlier methods using rounding and scaling techniques on the input edge costs, builds upon an iterative process that extends and merges sets of vertex labels representing paths. The algorithm resembles the Bellman-Ford method but implements the label sets as arrays of polynomial size by relaxing the requirements for strict Pareto optimality.

Granat and Guerriero (2003) introduce an interactive procedure for the MSPP based on a reference point labeling algorithm. The algorithm converts the multiobjective problem into a parametric single-objective problem whereby the efficient paths are found. The algorithm was tested on grid and random networks and its performance was measured based on execution time. They conclude that an interactive method, from their experimental results, is encouraging and does not require the generation of the complete Pareto-optimal set (which avoids the intractability problem). Likewise, (Coutinho-Rodrigues et al., 1999) suggests an interactive method that incorporates an efficient *k*-shortest path algorithm in identifying Pareto-optimal paths in a bi-objective shortest path problem. The algorithm was tested against other MSPP algorithms on 39 network instances. They conclude that their k-shortest path algorithm performs better in terms of execution time.

From a different perspective, evolutionary algorithms (EAs) have been extensively analyzed in single objective optimization problems but only a few researchers have applied EAs to the multiobjective shortest path problem either as the main problem or as a sub-problem in relation to route planning, traffic and transport design, information systems and communications network design. Gen and Lin (2004) use a multiobjective hybrid genetic algorithm (GA) to improve solutions to the bi-criteria network design problem (finding shortest paths) with two conflicting objectives of minimizing cost and maximizing flow. The paper shows how the performance of a multiobjective genetic algorithm can be improved by hybridization with fuzzy logic control and local search. The

results show a positive effect of hybridization, that is, an improvement in the convergence of the Pareto front.

Kumar and Banerjee (2003) present an algorithm for multicriteria network design (shortest paths and spanning trees) with two objectives of optimizing network delay and cost subject to satisfaction of reliability and flow constraints. They tested an evolutionary algorithm approach, Pareto Converging Genetic Algorithm (PCGA), to design different sized networks and found that EAs scale better in larger networks than two traditional approaches namely branch exchange heuristics and exhaustive search. They conclude that the primary advantage of EAs to solve multiobjective optimization problems is their diversity of solutions generated in polynomial time. Crichigno and Baran (2004) demonstrate similar representations (spanning trees) to Kumar's for a multicast algorithm. The basic difference between both algorithms is the latter adopts the Strength Pareto Evolutionary Algorithm (SPEA) in generating efficient solutions to the multicast routing problem.

Mooney and Winstanley (2006) show the behavior of an elitist genetic algorithm as applied to the MSPP in the field of geographic information systems (GIS). The experiment compares the runtime performance (execution time) of the EA against a modified version of Dijkstra's algorithm on several artificial and real road networks. The results show that the EA competes well with the modified Dijkstra approach in terms of execution time and that the EA converges quickly to the Pareto-optimal paths.

Janssens and Pangilinan (2008) conducted a sensitivity analysis of an MOEA for the MSPP using SPEA2. Their results show that the size of the population and the number of generations of the GA largely influence the output of the genetic algorithm and that mutation and recombination have only minimum influence on the output.

5.2 Multiobjective Evolutionary Algorithm

5.2.1 Genetic Algorithm for the MSPP

Genetic Representation. A chromosome or an individual consists of integer-ID vertices that form a path from the source vertex to a sink vertex. The length of

the chromosome is variable and may not be greater than the number of vetices, n.

Initial Population. A path or a chromosome is generated randomly in an ordered sequence from the source vertex to the sink vertex. The ID of the source vertex s is assigned to the first locus (array index) of the chromosome. The ID of a randomly generated vertex v_i is assigned to the second locus such that v_i belongs to the set of vertices connected to the source vertex s. This procedure continues iteratively for the succeeding vertices until a simple path to the sink vertex t is created. It is known that the population size of solutions increases exponentially with the number of objectives in the MSPP. There are two common options to respond to this problem, use a large population, or integrate a dynamic population sizing procedure in the GA. Dynamic population sizing has been implemented for single-objective EAs and has been known to show promising results. However, dynamic sizing remains a challenge to MOEAs with regard to MOPs. An alternative is to estimate the size of the initial population using Deb's (2001) approximation chart for finding the minimum population size in relation to the number of objectives.

Genetic Operators. The crossover scheme is an adaptation of the one-point crossover as shown in Figure 5.2. For each pair of paths a locus is randomly selected from one of the chromosomes (the shorter path in terms of number of vertices) and the vertex ID of the locus is matched with the genes in the other chromosome. If there is a match then crossover is performed, otherwise two new paths are selected for crossover until the mating pool is empty. It should be easy to see that the loci of both individuals need not be the same.

In the mutation operator, a locus is randomly selected from the chromosome. The algorithm proceeds by employing the method in the *initial population* process to create a new path, but the start vertex is replaced by the locus.

Fitness and Selection. Two objective configurations are considered for finding efficient paths, (3-S) and (2-S|1-M). S-type objectives are sum problems that are to be minimized whereas M-type objectives are max-min problems that are to be minimized. The selection of parents and offspring is done by two algorithms, SPEA2 (Zitzler et al. 2002) and NSGA-II (Deb et al. 2002). The program runs through the interface of PISA (Bleuler, 2003).

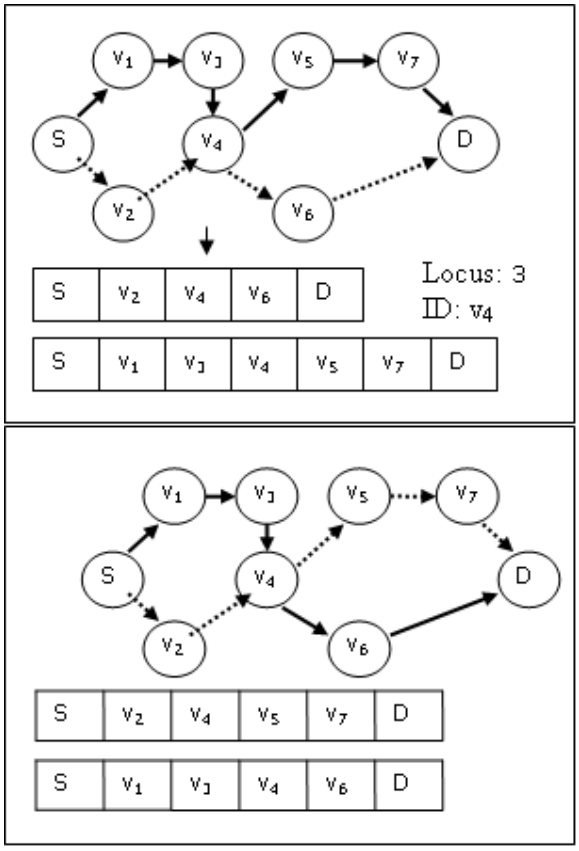

Figure 5.2 Crossover starts at the locus at position 3

5.2.2 Experiments and Results

The experiments intend to show the behavior of a multiobjective evolutionary algorithm as applied to the MSPP in terms of diversity and optimality of solutions, and computational complexity. The datasets are random networks that have been generated by Gandibleux et al. (2006). Nine configurations are selected for the random networks: (a) number of vertices: 50, 100, 200; (b) density of the network: 5%, 10%, 20%; (c) range of cost values $cp(i, j)$: [1, 100], p=1, 2, 3. Ten instances are generated for each network configuration and two objective configurations are considered for finding efficient paths. The population size of 36 was estimated from Deb (2001) with the assumption that 30% of nondominated paths are present in the initial population. The probability

for mutation and recombination is 0.5 and 0.5 respectively. Efficient paths are generated from a single source vertex (Vertex 1) to a single sink vertex (Vertex 50, 100, 200) for each objective configuration. For each network configuration, the number of efficient paths is computed from three different generation runs: 50, 100, and 200, which makes 540 combinations in total. Figure 5.3 shows a comparison of the average number of nondominated solutions generated by SPEA2 and NSGA-II with a 3-S configuration. There are more solutions generated by NSGA-II than SPEA2 in all graph configurations.

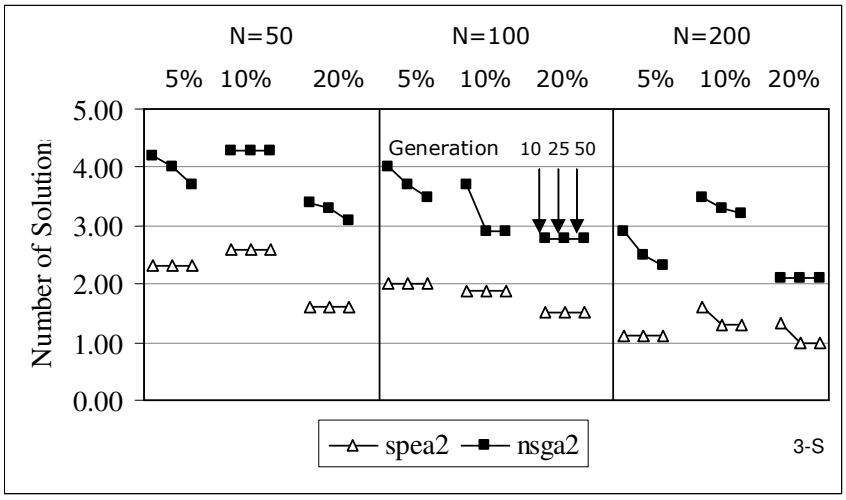

Figure 5.3 Average number of paths for the 3-S configuration

In SPEA2, solutions are not affected by the number of generations for the 50-vertex and 100-vertex graphs whereas in NSGA-II, the number of nondominated solutions in most graphs seems to change as the number of generations increases.

Figure 5.4 shows a comparison of the average number of nondominated solutions generated by SPEA2 and NSGA-II with a 2-S|1-M configuration. There are more solutions generated by NSGA-II than SPEA2 in most of the graph configurations. In both SPEA2 and NSGA-II, the number of nondominated solutions in most graphs changes as the number of generations increases.

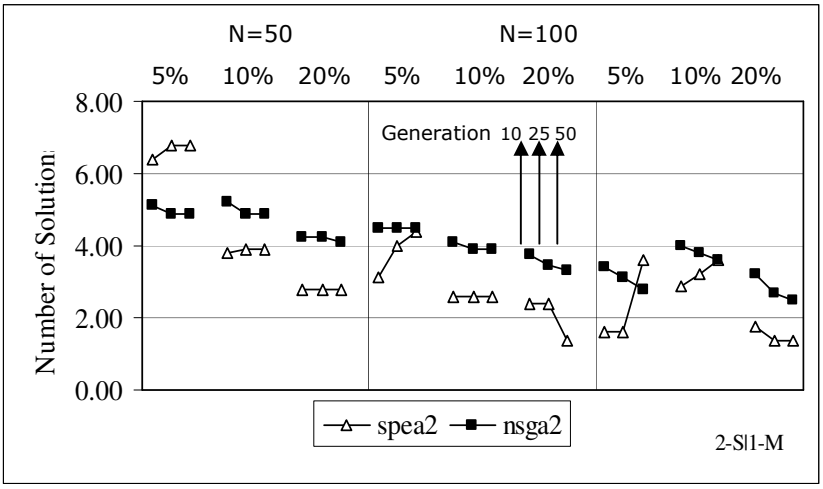

Figure 5.4 Average number of paths for the 2-S|1-M configuration

In terms of visualizing the diversity of solutions, Figures 5.5 to 5.8 show the value path plots of a 50-vertex, 10%-density network. The value path plot provides information on how good an algorithm is in finding diverse solutions and good trade-off solutions for problems having more than two objectives. A good spread of solutions over a range implies that an algorithm is good in finding diverse solutions. A large change in slope between objectives implies good trade-off solutions. The figures illustrate that both SPEA2 and NSGA-II find good trade-off paths for the 3-S and 2-S| 1-M objective configurations.

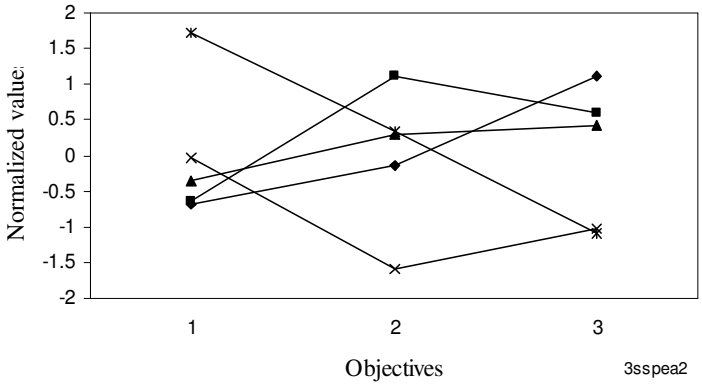

Figure 5.5 A value path plot for 3-S configuration by SPEA2.

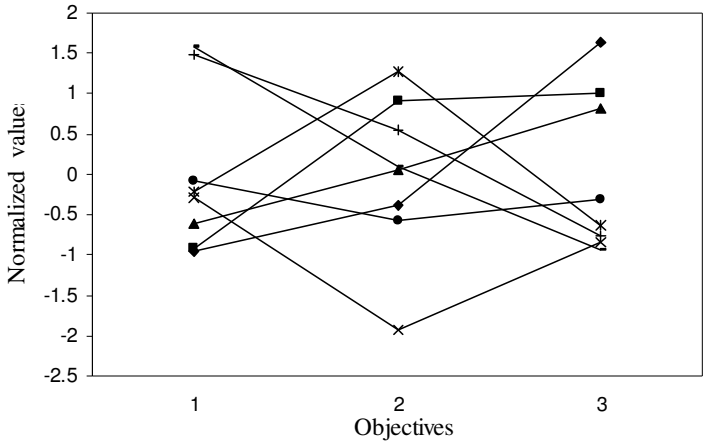

Figure 5.6 A value path plot for the 3-S configuration in NSGA-II

Proximity of solutions to the Pareto-optimal set is another issue for the evaluation of the MOEA. In the absence of a Pareto-optimal set in the MSPP, there is difficulty in the evaluation of the MOEA solutions in terms of proximity to the Pareto-optimal front. Similarly, there is no assurance that the nondominated solutions of the MOEAs will converge to the Pareto-optimal front or the maximal set. A performance measure is necessary to compare nondominated sets in order to determine which set is best.

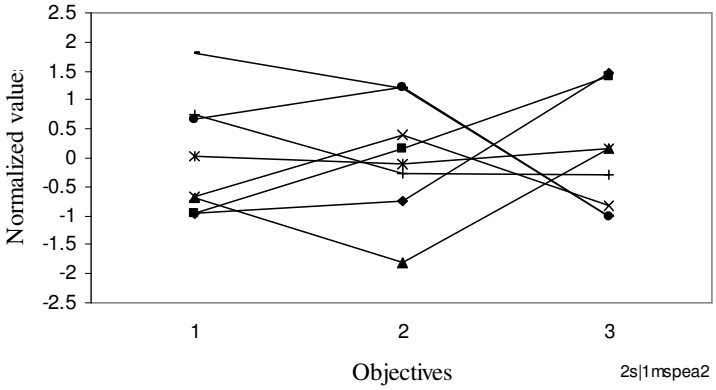

Figure 5.7 A value path plot for the 2-S|1-M configuration in SPEA2

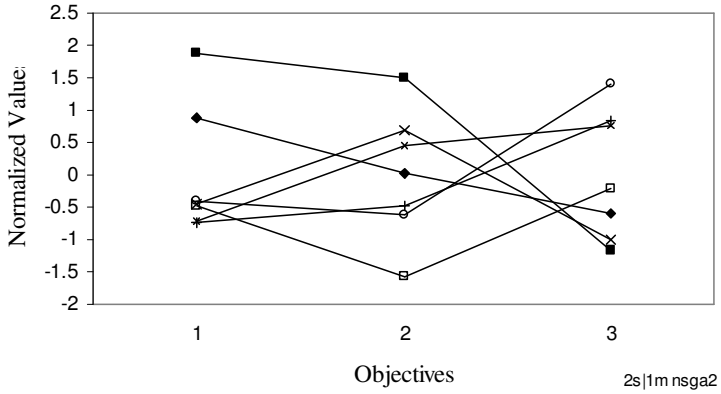

Figure 5.8 A value path plot for the 2-S|1-M configuration in NSGA-II.

5.3 Sensitivity Analysis

The results in Section 5.2.2 show that the increase in the number of generations in the MOEA does not significantly change the number of efficient paths. This means that the mutation and recombination operators do not find new solutions as the generation increases. The following section investigates why this is the case and shows which of the input parameters mainly affects the output of the MSPP.

5.3.1 Experiments and Results

A network of 50-vertices with 10% density, which represents the other networks of this type used in the study, is utilized to illustrate sensitivity analyses of SPEA2 and NSGA-II as applied the multiobjective shortest path problem. The initial population is 36 as estimated from Deb (2001). The output variable is the cardinality of the nondominated set. The input parameters are *mrate*, *xover*, and *maxgen*, the mutation probability, recombination probability, and number of generations respectively. The configurations of the input parameters are generated using the Sobol' method and their distributions are listed in Table 5.1.

Input factor	Description	Probability Distribution
mrate	Mutation probability	Uniform(0,1)
xover	Recombination probability	Uniform(0,1)
maxgen	Number of generations	Uniform(1,10) Uniform(10,20)

Table 5.1 Input factors and probability distributions

Table 5.2 shows that all of the variation in the output is due to the number of generations of the MOEA when the maximum number of generations is 10. The highest first-order indices among the input parameters are in **bold**. The mutation and recombination rates have no direct influence on the variation in the output. Similarly, their interaction indices are small. This means that the *selection operator* mainly generates the efficient paths and that the MOEAs converge to efficient paths after a few generations, in this case, in just 10 generations. Results of the sensitivity analysis for the 2-S|1-M configuration show similar results. It should be noted that during the sensitivity analysis, the population in each configuration was constant at 36 and that no significant improvements were recorded after the 10th generation in most of the graphs as shown in Figures 5.3 and 5.4.

Sobol first-order indices	SPEA2	NSGA-II
mrate	-0.010	**0.010**
xover	-0.022	-0.005
maxgen	1.227	1.138
Sobol total-order indexes		
mrate	0.004	-0.002
xover	-0.011	-0.000
maxgen	1.233	1.1429

Table 5.2 Sobol' indices for a 3-S configuration, 10 generations

Table 5.3 shows the results of the sensitivity analysis from the 11th generation until the 20th generation of a 3-S configuration of the same graph in Table 5.2. The first-order indices in SPEA2 differ from the first-order indices in NSGA-II but their total-order indices are very similar. The first-order indices for mutation rate and crossover rate contribute to more than 90% of the variation in the output, whereas the indices for mutation and crossover in SPEA2 do not change much. This explains the results in Figure 5.3, i.e. the average cardinality of

nondominated sets in SPEA does not change after 10 generations, whereas the cardinality of the solutions in NSGA-II varies. The first-order and total-order indices for *maxgen* have changed considerably. In the first 10 generations, the number of generations contributes mostly to the variation in the output, but from the 11^{th} to the 20^{th} generation, its effect on the output has diminished. Instead, the interaction of the mutation and recombination operators has the greater effect on the output, particularly in SPEA2. In NSGA-II on the other hand, the direct effects and interaction effects of the genetic operators have also increased considerably after 10 generations.

Sobol First-order indices	SPEA2	NSGA-II
mrate	-0.381	**0.567**
xover	0.121	0.487
maxgen	-0.132	0.272
Sobol total-order indexes		
mrate	0.714	0.657
xover	0.931	0.740
maxgen	0.392	0.272

Table 5.3 Sobol' indices for a 3-S configuration, 11-20 generations

Table 5.4 shows the results of the sensitivity analysis from generations 1 to 10 and generations 11 to 20 of a 2-S|1-M configuration of the same graph in Table 5.2. It presents similar results to Table 5.3 except that the Sobol' indices in NSGA-II are all zeroes. The indices are zeroes because there is no variation in the output. This means that whatever the combinations of *mrate*, *xover*, and *maxgen* from generations 11-20, NSGA-II generates the same number of nondominated solutions.

	1-10 generations		11-20 generations	
Sobol First-order indices	SPEA2	NSGA-II	SPEA2	NSGA-II
mrate	0.006	-0.006	0.554	0.000
xover	0.013	0.011	0.242	0.000
maxgen	1.152	1.214	-0.069	0.000
Sobol total-order Indexes				
mrate	0.005	0.000	0.970	0.000
xover	0.009	0.008	0.901	0.000
maxgen	1.150	1.207	0.196	0.000

Table 5.4 Sobol' indices for a 2-S|1-M configuration

There can be two reasons, either the Pareto-optimal set has been achieved or the algorithm is trapped in a local optimum. Further investigation validates that NSGA-II has achieved the Pareto-optimal set in 10 generations.

5.4 Summary

The chapter presents the feasibility of a multiobjective evolutionary algorithm as applied to the multiobjective shortest path problem on several graph configurations. Results show that an MOEA is a feasible technique in finding approximations of the Pareto-optimal set to the MSPP.

Section 5.2 shows that in most graphs, the increase in the number of generations does not necessarily change or improve the cardinality of nondominated sets. In fact, it shows that most nondominated solutions are generated in the early generations of the MOEA. This seems to exhibit the *genetic drift* phenomenon. However, the sensitivity analysis shows otherwise.

Section 5.3 explores the sensitivity of an MOEA to changes in its input factors. The sensitivity analysis shows that the mutation and recombination do not cause much of the variation in the output in the early stages of the MOEA run. The number of generations has the most influence on the output. This means that the *selection operator* mostly influences the cardinality of the nondominated sets in the early stages of the run. However, after the 10^{th} generation, the indices of the interaction effects of mutation and crossover have increased. This means that exploitation and exploration of the search space continue after the early stages of the MOEA run. Although the mutation and recombination operators have almost no direct influence on the output, their total sensitivity indices show that their interaction effects are important. Therefore, they cannot be removed as input parameters. Furthermore, the low first-order sensitivity indices of mutation and crossover imply that the MOEA is robust in terms of the input parameter values of the mutation and crossover rates.

Chapter 6

Oblique Decision Trees: An optimization problem with a tree structure of continuous variables

6.1 Introduction

The data mining task of classification using decision trees (DT) has been studied for many years. Researchers in the fields of statistics, decision theory, and machine learning have reported a huge amount of work on this topic. Significant improvements on decision trees are difficult to produce but promising researches still remains open and have to be explored. One of these areas is the application of evolutionary algorithms in decision tree construction and optimization. For example, optimal tree construction belongs to the set of NP-hard problems (Murthy et al. 1994). The selection of a linear split in multivariate decision trees is also NP-hard (Murthy et al. 1994). Both optimization problems make standard tree-based methods infeasible for finding a good or a sub-optimal solution. However, the application of evolutionary algorithms is a promising option in finding solutions to NP-hard problems and is known to exploit and explore large search spaces and solve optimization problems that many classic optimization algorithms cannot.

The purpose of this chapter is to apply evolutionary algorithms in the optimization aspects of decision tree construction, to describe the EA performance, and to investigate if they succeed or fail in relation to decision tree optimization.

6.1.1 Problem Definition

A clear definition of the class and type of variables or attributes in decision tree construction is useful in order to understand how classification works. The "target variable" is the variable to be modeled and predicted by other variables. It is equivalent to the dependent variable in linear regression. There must be one and only one target variable in a decision tree analysis. A "predictor variable" or simply predictor is a variable whose values are used to predict the value of the target variable. It is equivalent to the independent variable in linear regression. There must be at least one predictor variable specified for decision tree analysis. A continuous variable has numeric values and are called *ordered* or *monotonic* variables. A categorical variable has values that function as labels rather than as numbers. Some programs call categorical variables as *nominal* variables.

In machine learning, a decision tree is a predictive model that is a mapping of observations about an item to conclusions about the item's target value. Each inner node corresponds to a predictor. An arc to a child represents a possible value of that variable. A leaf represents the predicted value of a target variable given the values of the variables represented by the path from the root node. Figure 6.1 shows a diagram of a decision tree. The machine learning technique for inducing a decision tree from data is called decision tree learning. Decision tree learning is also a common method used in data mining. Here, a decision tree describes a tree structure wherein leaves represent classifications and branches represent conjunctions of features that lead to those classifications (Menzies and Hu 2003). Splitting the source set into subsets based on an attribute value test can learn a decision tree. This process is repeated on each derived subset in a recursive manner. The recursion is completed when splitting is either non-feasible, or a singular classification can be applied to each element of the derived subset.

Several issues have to be considered simultaneously in this exploration, such as computational complexity, accuracy, depth of tree, tree size, balance, and

stability. The study, for purposes of clarity and visualization uses two criteria only henceforth reducing the decision tree construction problem to a bi-objective optimization problem, i.e. minimization of tree size and maximization of classification accuracy. Classification accuracy is the ratio of correctly classified instances over the total number of instances in a dataset. Computational complexity is not an objective criterion in the study.

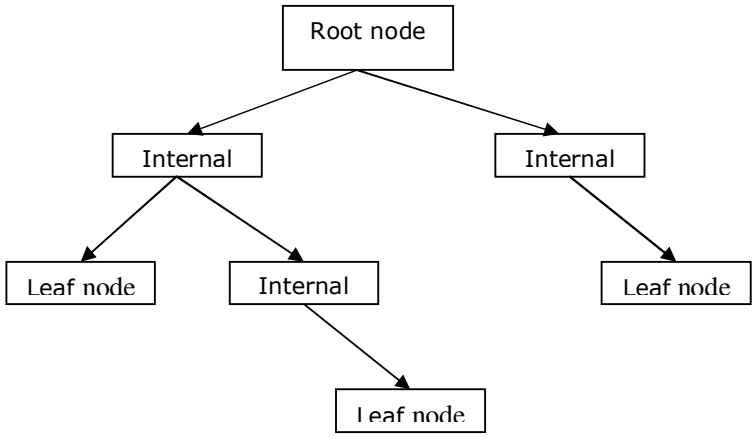

Figure 6.1 Decision tree

The DT optimization problem can be formulated as

Minimize	$(\boldsymbol{T}, x)$	$x = 1,2\ldots, n$
Maximize	$(\boldsymbol{T}, \alpha)$	$\alpha \in \Re$: $0 \leq \alpha \leq 100\%$

where: $\boldsymbol{T}$ is the decision tree
x is the size of tree $\boldsymbol{T}$
α is the classification accuracy of tree $\boldsymbol{T}$
n is the maximum number of leaves of tree $\boldsymbol{T}$

6.1.2 Literature Review

SPLITTING CRITERIA

An early technique by Quinlan (1986) that influenced a large part of the research on decision trees is useful to look at in order to understand basic decision tree construction. A fundamental part of any algorithm that constructs a decision tree from a dataset is the method in which it selects attributes at each node of the tree. Some attributes split the data up more purely than others. That means that their values correspond more consistently with instances that have particular values of the target attribute than those of another attribute. Therefore, such attributes have some underlying relationship with the target attribute. Essentially, splitting is finding a measure that compares attributes with each other and then deciding the variables that split the data more purely higher up the tree.

A measure used from information theory in decision tree construction is entropy. Informally, the entropy of a dataset can be considered as a measure of how disordered it is. It has been shown that entropy is related to information, in the sense that the higher the entropy, or uncertainty, of some data, then the more information is required to describe that data. In building a decision tree, the aim is to decrease the entropy of the dataset until leaf nodes are reached. The entropy of a dataset, $\boldsymbol{S}$ with respect to one variable, in this case the target variable, is defined as:

$$Entropy\ (\boldsymbol{S}) = \sum p_i \log_2 p_i\ ; \qquad i = 1,\ldots,c \tag{6.1}$$

where p_i is the proportion of instances in the dataset that take the i-th value of the target variable. This probability measures give an indication of how much uncertainty is present about the data. The log_2 measure represents how many bits would be needed in order to specify the class (value of the target attribute) of a random instance.

A measure called Information Gain, calculates the reduction in entropy that would result in splitting the data on a predictor variable A.

$$Gain(\boldsymbol{S}, A) = Entropy(\boldsymbol{S}) - \sum_{v \in A} \frac{|\boldsymbol{S}_v|}{|\boldsymbol{S}|} Entropy(\boldsymbol{S}_v) \qquad (6.2)$$

where v is a value of A, $|\boldsymbol{S}_v|$ is the subset of instances of $\boldsymbol{S}$ where A takes the value v, and $|\boldsymbol{S}|$ is the number of instances.

The observation that Information Gain is unfairly biased has led to new proposals of splitting criteria. An obvious way to negate the bias or "greediness" of Information Gain is to take into account the number of values of a variable. This is exactly the approach that can be used. A new, improved calculation for variable A over data $\boldsymbol{S}$ is:

$$GainRatio(\boldsymbol{S}, A) = \frac{Gain(\boldsymbol{S}, A)}{I(A)} \qquad (6.3)$$

$$\text{where} \quad I(A) = \sum \frac{-log_2 P_i}{N} \qquad (6.4)$$

The second equation for $I(A)$ measures the information content for the variable A by looking at each proportion p, of instances that take value i for the variable.

Classification and Regression Trees (CART) which was developed in the 1980s (Breiman et al. 1984) is an analytic procedure for predicting the values of continuous or categorical dependent variables from continuous or categorical predictors. When the target variable of interest is categorical, the technique is referred to as Classification Trees. If the target variable of interest is continuous in nature, the method is referred to as Regression Trees. For classification problems, the goal is generally to find a tree where the terminal tree nodes are relatively "pure" using an impurity measure called the Gini measure. For regression tree problems node purity is usually defined in terms of the sums-of-squares deviation within each node.

The Gini measure is the measure of impurity of a node and is commonly used when the target variable is a categorical variable, and defined as:

$$g(t) = \sum p(j|\, t)\, p(i|t)\ ;\ i \neq j \qquad (6.5)$$

where the sum extends over all classes and $p(j|t)$ is the probability of class j at the node t.

PERFORMANCE MEASURES

There are several criteria to evaluate a decision tree algorithm for a given dataset over another algorithm on the same dataset. Conciseness of the decision tree that an algorithm produces may be one measure and clarity of the rules from the tree is another. It is also possible to look at how fast an algorithm is, but the interest of most researchers is in an algorithm's learning ability and classification accuracy.

Moret (1982) summarizes work on measures such as tree size, expected testing cost, and worst-case testing cost. He shows that an algorithm that minimizes one measure does not guarantee minimization of other measures. On the other hand, Fayad and Irani (1990) argue that by concentrating optimization on one measure, improvement on the performance of other measures is possible. Kononenko and Bratko(1991) points out that comparisons based on classification accuracy are unreliable, because different classifiers produce different types of estimates and accuracy values can vary with prior probabilities of the classes. They suggested an information-based metric to evaluate a classifier as a remedy to accuracy problems.

Tree quality depends more on stopping rules than on splitting rules. Pruning is argued to be a better method than stop-splitting rules (Murthy et al. 1994). No single pruning method has been adjudged superior to others. Obtaining the 'right' sized trees is important for several reasons, which depend on the size of the classification problem (Gelfand and Ravishankar 1991). For moderate sized problems, the critical issues are classification accuracy, error-estimation, and gaining insight into the predictive and generalization structure of the data. For large tree classifiers, the critical issue is optimizing structural properties such as height and balance (Wang and Suen 1984).

Murthy (1998) notes that the time complexity of induction and post-processing is exponential in tree height in the worst case (Martin and Hirschberg 1995). This puts a premium on designs that produce shallower trees, multi-way rather than binary splits, and selection criteria that prefer balanced splits. Several authors have designed methods to improve upon greedy algorithms by

constructing near-optimal decision trees using a look-ahead algorithm (Murthy and Salzberg 1995) genetic programming (Koza 1991) and simulated annealing (Heath et al. 1993b, Lutsko and Kuijpers 1994, Folino et al. 2000).

Many authors suggest that using a collection of decision trees, instead of just one reduces the variance in classification performance (Kwok and Carter 1990; Shlien 1990; Shlien 1992; Buntine 1992; Breiman 1990). The idea is to build a set of trees (ensembles) for the same training sample and then combine their results. Multiple trees have been built using randomness or using different subsets of attributes for each tree (Heath et al. 1993).

OBLIQUE DECISION TREES

There have been many studies in optimizing split selection. A split can be either univariate (axis-parallel) or multivariate (oblique). An axis-parallel split divides a node with only one predictor and an oblique split divides a node with a linear or non-linear combination or predictors. Trees that are induced using oblique splits are called oblique trees. Finding optimal linear splits is known to be intractable and is an optimization problem, so heuristic methods are required. Methods for finding good linear splits include linear discriminant analysis, hill climbing search, linear programming, and perceptron training in neural networks. This section examines existing literature that addresses the problem of finding oblique splits for Classification Trees.

The more popular approach in finding univariate splits examines all possible binary splits along each predictor variable to select the split that most reduces some measure of impurity. Such method is used in CART (Breiman et al. 1984), Quinlan's C5.0 (1986), and Morgan's THAID (1973). This type of exhaustive search has two major drawbacks. First, the computational complexity of finding splits for nominal variables increases exponentially with the number of its categorical values and obtaining linear combinations of ordered predictor variables becomes computationally difficult. Second, it tends to select variables that have more instances.

Loh and Vanichsetakul (1988) free the bias in variable selection for ordered predictor variables by separating the selection of the variable and the split point. They calculate the F-statistic for each predictor variable, select the variable with the highest F-statistic, and apply linear discriminant analysis to find the split

point. Unordered variables are transformed into ordered variables but this transformation is not free from bias. Loh and Shih (1997) further developed QUEST (Quick, Unbiased, Efficient, Statistical Tree) to address the disadvantages of their first algorithm, e.g. a split rapidly lessens the learning sample when a node is split into as many sub-nodes as there are classes. Loh and Shih conclude that in terms of classification accuracy, variability of splits, and tree size, there is no clear winner when univariate splits are used. However, QUEST trees, based on a linear combination of splits are usually shorter and more accurate than the same trees based on univariate splits.

Preliminary work in oblique decision trees was done by Breiman et al. (1984). They introduced CART with linear combinations (CART-LC) as a method to create oblique splits. CART-LC iteratively finds local optimal values of each of the coefficients. CART-LC generates and tests hyperplanes until the marginal benefits become smaller than a specified constant. It uses a backward-deletion procedure to simplify the structure of the split by deleting variables that have little contribution to the effectiveness of the split.

Heath et al. (1993b) developed an algorithm (SADT) to induce oblique decision trees based on simulated annealing. The algorithm first creates an initial random hyperplane to divide the dataset and creates hyperplanes on the two new partitions. This process of creating hyperplanes is recursive. Their experiments show that simulated annealing works well for generating oblique decision trees since it generates smaller trees without reducing classification accuracy.

Murthy et al. (1994) developed a randomized algorithm Oblique Classifier 1 (OC1) and has been thought to be an extension of CART-LC. The primary contribution of OC1 in oblique decision tree research is its perturbation algorithm in finding linear splits. The perturbation is a randomization of hyperplanes to escape local minima. Murthy notes that there are data distributions wherein univariate splits perform better than oblique splits. They accounted such distributions in OC1 by computing the best axis-parallel split at each node before applying hyperplane perturbation. The experiments of Murthy et al. show that OC1 outperforms CART-LC in several datasets.

Vadera (2005) presents a new algorithm that utilizes linear discriminant analysis to identify oblique relationships between continuous attributes and then carries out an appropriate modification to ensure that the resulting tree errs on

the side of safety. The algorithm is evaluated with respect to a cost-sensitive algorithm (ICET), an oblique decision tree algorithm (OC1) and to linear programming.

A study of multi-class binary decision trees with oblique planes using nonlinear programming can be found in Street (2005). Street finds that a nonlinear programming approach (OC-SEP) appears to offer an advantage over some axis-parallel methods on oblique separating planes but is more computationally demanding than greedy algorithms. The OC-SEP implementation is still not feasible for large scale problems. He suggests that incorporating feature-selection to the objective function can achieve enhanced classification accuracy and interpretability.

EVOLUTIONARY ALGORITHMS AND DECISION TREES

An evolutionary algorithm is a promising technique to build oblique decision trees. Cantu-Paz and Kamath (2003) summarizes the advantages of an EA, which are the following:

1. EAs can consider more than one coefficient at a time and may not be trapped in local optima as easily as the simple greedy hill-climbing algorithms.
2. EAs have been shown to have good scalability to the dimensionality of the problem (Harik et al. 1999).
3. EAs are tolerant to noisy fitness evaluations (Miller and Goldberg 1996).
4. EAs are stochastic algorithms; they produce different trees on the same data set that can be easily combined into ensembles.

Cantu-Paz and Kamath (2003) extended the OC1 algorithm by Murthy et al. (1994) into three new algorithms by using evolution strategies, genetic algorithms, and simulated annealing to find oblique partitions on a variety of datasets including seven datasets from the UCI (University of California at Irvine) machine learning repository. They compared the performance of six algorithms according to accuracy, number of nodes, and execution time. They conclude (1) that the EAs scale up better than traditional methods (OC1-AP, OC1, CART-LC) to the dimensionality of the data and (2) that creating ensembles with the EAs results in higher accuracy than single trees produced by

existing methods. The study suggests future work may be done on scalability using larger datasets, on design of specialized operators, and the integration of local hill-climbing algorithms to EAs.

Sörensen and Janssens (2003) developed a genetic algorithm for binary decision trees to overcome the drawbacks of the automatic interaction (AID)-technique. The technique uses specialized genetic operators that preserve the structure of the trees to find a diverse set of DTs with high explanatory power. The algorithm's advantage as compared to the AID-technique is that it gives the decision maker a set of high-fitness decision trees to choose from instead of only one DT.

Papagelis and Kalles (2001) proposed a genetic algorithm (GAtree) to evolve binary decision trees and introduced a single-objective fitness function to balance accuracy and size of decision trees. The GAtree algorithm randomly picks a variable to create a node and a binary decision tree is completed down to the leaves with this same line of thought. There is no discussion of a splitting criterion. The selection of highest fit chromosome is taken by cross-validation. Their experiments used selected datasets from the UCI repository and the GAtree performance was compared to C4.5 and the One-R algorithm. They find that the time burden of the GAtree is substantially bigger than greedy heuristics but on the other hand, their experiments show advantages over greedy heuristics when there are irrelevant or strongly dependent variables.

Bot and Langdon (2000) applied genetic programming (GP) to linear classification trees using limited error fitness (LEF), cross validation, and Pareto scoring on continuous data attributes from four UCI repository data sets. The study measured the GP's performance, accuracy and tree size in comparison to OC1, C5.0, and M5'. They used the GPsys program by Qureshi (1997), which is a strongly-typed, steady state GP system. They conclude that in some datasets, the GP works comparatively or better than the reported accuracy from other decision tree algorithms but the GP performs worse on other datasets. Furthermore, fitness-sharing Pareto creates a promising approach to reducing tree size but creates larger trees than Pareto domination while LEF does not improve or worsen classification accuracy, but saves much on execution time.

Kim (2004) proposed an evolutionary multiobjective optimization (EMO) approach that searches for the best accuracy rate of classification for different sized trees using genetic programming. He introduces structural risk

minimization that finds a desirable number of rules for a given error bound. His EMO reduces the size of trees dramatically and his best rule set is better than that of C4.5, but computing time takes longer

A multicriteria approach in evaluating decision trees was presented by Bryson (2004). He provided a method to assist decision makers to select an efficient decision tree by ranking decision trees using a weighing model. The model evaluates a decision tree on several performance measures such as accuracy rate, tree simplicity and size, stability, and the discrimination power of its predictors.

Zhao (2007) presented a multiobjective genetic programming (MOGP) system for developing Pareto optimal decision trees. The system allows the decision-maker to make tradeoffs in several ways based on his estimates of classification errors, and recommends a set of alternative solutions accordingly. As an evolutionary approach, the system visualizes the progress of the evolution of solutions such that the decision maker can decide to stop the procedure when satisfactory solutions have been found or when the solutions on the front appear to have stabilized.

FEATURE SELECTION

Irrelevant attributes pose a significant problem for most machine learning methods Decision Tree algorithms, even axis-parallel ones, can be confused by too many irrelevant attributes. Because oblique decision trees learn the coefficients of each attribute at a DT node, searching for good coefficient values is much more efficient when there are fewer attributes. (Murthy et al. 1994). Feature selection (also known as subset selection) is a process used in machine learning, wherein a subset of the features available from the data is selected for a learning algorithm. The best subset contains the least number of dimensions that most contribute to accuracy, hence making the search space smaller. With this, oblique DT construction methods can benefit substantially by using a feature selection method.

Dash and Liu (1997) present a survey of several feature selection methods according to generation procedure and evaluation function. The generation procedure defines how a method generates a subset of features for evaluation. The procedure starts either with no feature, or with all features, or with a

random subset of features. The evaluation function measures the goodness of a subset produced by some generation procedure. They group evaluation functions into five categories: distance, information, dependence, consistency, and classifier error rate measures and are briefly described as follows:

1. *Distance Measures* also known as separability, divergence, or discrimination measures. For a two-class problem, a feature *x* is preferred to another feature *y* if *x* induces a greater difference between the two-class conditional probabilities than *y*; if the difference is zero, then *x* and *y* are indistinguishable. An example is the Euclidean distance measure.
2. *Information Measures* determine the information gain from a feature. The information gain from a feature *x* is defined as the difference between the prior uncertainty and expected posterior uncertainty using *x*. Feature *x* is preferred to feature *y* if the information gain from feature *x* is greater than that from feature *y* (e.g., entropy measure).
3. *Correlation Measures* qualify the ability to predict the value of one variable from the value of another. The coefficient is a classical dependence measure and can be used to find the correlation between a feature and a class. If the correlation of feature *x* with class *c* is higher than the correlation of feature *y* with *c*, then feature *x* is preferred to *y*. A slight variation of this is to determine the dependence of a feature on other features; this value indicates the degree of redundancy of the feature. All evaluation functions based on dependence measures can be divided between distance and information measures.
4. *Consistency Measures* find out the minimally sized subset that satisfies the acceptable inconsistency rate that is usually set by the user. These are characteristically different from other measures, because of their heavy reliance on the training dataset. In the simplest implementation, it does a breadth-first search and checks for any inconsistency considering only the candidate subset of features.
5. *Classifier Error Rate Measures* are called "wrapper methods", i.e. the classifier is the evaluation function. Features are selected using the classifier that later on uses these selected features in predicting the class labels of unseen instances. The accuracy level is very high

but computationally costly. A typical wrapper method can use different kinds of classifiers for evaluation.

Dash and Liu (1997) state that there is no single feature selection method that can handle all applications. The choice of a feature selection method depends on various data set characteristics: data types, data size, and noise. A survey and comparison of feature selection methods are also found in Guyon and Elisseeff (2003), Blum and Langley (1997), and Jain and Zongker (1997).

6.2 Multiobjective Evolutionary Algorithm

The EA in the decision tree optimization problem is a search function the finds a linear combination of the predictor variables that best splits the dataset. The algorithm to find the best linear split on a dataset $\boldsymbol{S}$ is described below.

1. $t = 0$. Find $\boldsymbol{H}_0$, the best axis-parallel split of $\boldsymbol{S}$
2. Place 1 copy of $\boldsymbol{H}_0$ to the initial population of linear splits $\boldsymbol{P}_0$
3. Evaluate fitness using an impurity measure
4. Perform selection based on SPEA2 or NSGA-II
5. If ($t \geq$ *max generation*) or (another stopping criterion is satisfied) then End.
6. Perform recombination and mutation.
7. $t = t + 1$ and go to Step 3.

6.2.1 Genetic Algorithm for DT

Genetic Representation. A chromosome or an individual consists of real-parameter genes that represent the coefficients of a hyperplane that splits the dataset to form a node of the decision tree. The length of the chromosome is fixed and is equal to the number of dimensions, d of the dataset. Each gene in a chromosome represents the coefficient of a predictor variable x_i where i=1, 2,…, d in a d-dimensional dataset. Each gene is a real random number in [-1, 1]. The decision tree is represented as a binary tree of chromosomes.

Initial Population. The initial population is composed of chromosomes of linear splits. An axis-parallel chromosome is added to the initial population to ensure

that the initial population can capture univariate splits in the early generations of the EA.

Genetic Operators. Recombination is implemented as the arithmetic average of both parents as defined in (3.3). The experiment adopts a non-uniform mutation operator and as described in (3.1) and (3.2).

Fitness and Selection. The fitness of a chromosome is evaluated by an impurity measure called the *towing rule* (Breiman et al. 1984). For the selection of parents and offspring, two MOEAs are compared namely SPEA2 (Zitzler et al. 2002) and NSGA-II (Deb et al. 2002). The program runs through the interface of PISA (Bleuler, 2003).

The proposed multiobjective evolutionary algorithm is an adaptation of OC1 (Murthy 1998). The extension comes from changing the original hill climbing and perturbation algorithms in OC1 to an EA process.

6.2.2 Experiments and Results

The following experiments intend to show the performance of two multiobjective evolutionary algorithms (SPEA2 and NSGA-II) in terms of selecting hyperplanes in the induction of oblique decision trees. The results are compared to two other algorithms: an axis-parallel algorithm (AP) and the oblique classifier (OC1 by Murthy). The experiment is similar to the work of Cantu-Paz and Kamath (2003) in terms of describing classification accuracy and tree size but is different in the sense that the current experiment describes the effects of the EA operators on building decision trees. Feature selection is not used in the UCI and synthetic datasets but is applied to the application datasets (see Section 6.4).

UCI DATASETS

In order to evaluate the EA effectively for purpose classification, the classification datasets from the UCI machine-learning repository are excellent benchmarks. Nine UCI classification datasets are chosen for the experiment. Seven of them are small datasets wherein the number of instances in dataset $n < 1000$, and two are large datasets, $n > 5000$. An additional artificial dataset,

the RCB dataset is used to test an algorithm's performance in generating oblique splits. The RCB dataset is an artificial dataset with linear partitions only. The datasets are briefly described in Table 6.1.

Name of dataset	Number of samples *n*	Number of dimensions *d*	Number of Classes *c*
Cancer	699	10	2
Diabetes	768	8	2
Housing	506	12	2
Iris	150	4	3
Vehicle	946	18	4
Glass	214	9	7
Vowel	528	10	4
RCB	2000	2	2
Optical-digit	3823 (training) 1797 (testing)	64	10
Pen-digit	7494 (training) 3498 (testing)	16	10

Table 6.1 Description of datasets

In order to estimate the classification accuracy of the classifiers, ten five-fold *cross-validation* experiments are implemented on the small datasets and the RCB dataset. Cross-validation is a technique to estimate the accuracy of a predictive model by partitioning the sample dataset into corresponding subsets and analyzing one subset (*training set*) and validating the analysis on the other subset (*validation set*). A *k*-fold cross-validation partitions the dataset in *k* subsets. Of the *k* subsets, a single subset is used as the validation set and the remaining *k*-1 subsets are combined to form the training set. The cross-validation repeats for *k* times with each of the *k* subsets used only once as the validation set.

For the large datasets (Optical-digit and Pen-digit), there is no need for cross-validation since they have separate training and testing sets. Fifty trees are generated from each dataset and the averages of their classification accuracy and tree size are used for comparison purposes.

The accuracy is measured as the proportion of correctly classified instances of each dataset and the size of the tree is measured by the number of leaves.

Each value also shows the standard error of their means. For SPEA2 and NSGA-II, a population of 28 is used, a mutation and a recombination rate of 0.5, a bit-turn probability of 0.5 and a run of 100 generations. For OC1, twenty random hyperplanes are used at each node and the best hyperplane is selected through hill-climbing and five perturbations are allowed for each hyperplane.

The results shown in Table 6.2 are the averages of fifty trees per data set. The averages in accuracy do not show significant differences for all four classifiers. The highest classification accuracy and the smallest tree size in each dataset are printed in bold. The averages of the tree size for the oblique classifiers have no significant differences but are obviously smaller than the trees induced by the axis-parallel algorithm particularly in the Vehicle and Vowel datasets. Similarly, the standard errors for accuracy and size of the oblique classifier are smaller than the results of the AP classifier in all the datasets. This means that the variation in the decision trees generated by the oblique classifiers is also small.

Dataset		AP	OC1	SPEA2	NSGA-II
Cancer	Accuracy	93.78±0.32	95.39±0.26	95.32±0.24	**95.57±0.23**
	Size	8.46±1.06	3.26±0.33	3.24±0.27	**3.20±0.35**
Diabetes	Accuracy	**73.53±0.46**	72.19±0.45	72.41±0.49	73.22±0.32
	Size	13.86±2.58	**6.54±0.98**	13.30±2.28	9.18±1.92
Glass	Accuracy	**64.63±1.08**	61.57±1.16	64.24±1.28	64.53±1.15
	Size	13.00±1.46	9.12±0.94	**8.72±0.88**	9.12±1.06
Housing	Accuracy	80.36±0.75	74.49±0.75	**80.47±0.62**	79.29±0.71
	Size	35.94±1.81	31.28±2.07	33.08±1.86	**28.96±1.78**
Iris	Accuracy	94.13±0.45	**95.53±0.57**	95.20±0.51	95.07±0.56
	Size	3.44±0.15	**3.04±0.03**	3.04±0.03	3.08±0.05
Vehicle	Accuracy	**69.42±0.55**	67.33±0.52	67.84±0.56	68.71±0.45
	Size	44.54±4.47	31.44±3.86	**29.48±3.70**	36.58±4.61
Vowel	Accuracy	74.72±0.76	78.75±0.72	**81.15±0.63**	80.64±0.70
	Size	60.76±2.04	**28.46±1.18**	31.66±1.15	31.68±1.22

Table 6.2 Comparison of accuracy and tree size from small datasets

In the 1400 trees generated from the small datasets, the AP classifier produces the worst solutions in terms of size whereas the best solutions in terms of accuracy are generated not from only one classifier but from all classifiers including the AP classifier. This observation supports the findings of Moret

(1982) that an algorithm that minimizes one performance measure does not necessarily minimize the other measures. It also corroborates the findings of Bot and Langdon (2000) that evolutionary algorithms do not necessarily perform well on all datasets. However, this observation contradicts the previous findings of Loh and Shih (1997) that a linear combination of spits is usually shorter and more accurate than axis-parallel splits. In terms of computing time, the AP classifier is the fastest followed by OC1. The EA classifiers took a much longer time to induce decision trees. This observation confirms the findings of Kim (2004).

The results of the averages of accuracy and size from the large datasets are shown in Table 6.3. The average results in accuracy do not show significant differences for all four classifiers. The averages of the tree size for the oblique classifiers have no significant differences but are much smaller than the trees induced by the axis-parallel algorithm. Of the 600 trees generated from the large datasets, the worst solutions in terms of accuracy and size were produced by the AP classifier while the best solutions were generated from the oblique classifiers. In these datasets, the observation that larger trees are less accurate holds. However, the finding the smaller trees are more accurate does not hold. The computing time of the AP classifier remains fast but the computing time of OC1 has increased tenfold. Still, the EA classifiers took a much longer time to induce decision trees from the large datasets.

Dataset		AP	OC1	SPEA2	NSGA-II
RCB	Accuracy	92.85±0.20	**98.37±0.10**	97.54±0.13	97.62±0.14
	Size	83.72±2.51	**12.78±0.63**	18.50±1.12	17.28±1.09
Optical-digits	Accuracy on test set	84.78±0.22	86.51±0.18	**88.69±0.16**	88.62±0.14
	Size	148.14±7.53	**54.44±4.59**	59.08±4.79	57.26±4.19
Pen-digits	Accuracy on test set	90.91±0.11	92.88±0.09	94.14±0.11	**94.20±0.08**
	Size	211.10±5.15	74.98±3.53	**71.72±3.87**	76.64±3.97

Table 6.3 Comparison of accuracy and tree size from large datasets

Comparing the results of Table 6.2 and Table 6.3, the performance of the AP classifier in terms of accuracy seem to decline on the larger datasets. It may be obvious that the AP classifier induces larger and less accurate trees from large-

dimensional datasets because of its technique of splitting the dataset with only one predictor at a time.

This is not conclusive. For example, the AP classifier performs poorly in terms of accuracy and tree size on the RCB dataset which has only two dimensions but it performs well on the Diabetes, Glass, and Vehicle datasets which have 8, 9, and 18 dimensions respectively. The oblique classifiers' estimates on accuracy are better than the estimates of the AP classifier for most datasets except for the Diabetes, Glass, and Vehicle datasets. It is worthwhile to examine the reasons why the performance of the AP classifier is better on these datasets. Likewise, it is useful to examine the effect of changing the parameter settings of the oblique classifiers on their accuracy and tree size.

In the case of the OC1 classifier, increasing the number of hyperplanes generated per node slows down the program but may produce better trees. Increasing the number of random jumps may generate better coefficients but may cause over-fitting. In the case of the EAs, decreasing the number of generations gives worse trees in both accuracy and tree size. Increasing the number of generations does not ensure improvement in tree quality but surely increases computing time. Increasing the population size does not necessarily result in better trees but will definitely increase its computing time. This leads the experiment to one alternative - examine the effects of changing the mutation and recombination rates on tree quality (see Section 6.3).

6.3 Sensitivity Analysis

The purpose of the sensitivity analysis is to determine the effect of the input parameters specifically the mutation and recombination rates on the induction of oblique decision trees. The sensitivity analysis on one hand determines the input parameter or parameters that cause the variation in the output, if they cause any at all. If an input parameter is known to cause much of the variation in the output, this variable may be fixed to a certain value in order to reduce the variation in the output and as a consequence makes the algorithm more stable. On the other hand, if no single parameter causes the variation, the sensitivity analysis provides a value to describe the interaction among the input parameters and determines which among them does not interact at all with other variables.

In this experiment, only two input parameters are tested for sensitivity namely, the mutation rate and the crossover rate. The input parameter *population size* remains at 28, and *maxgen* at 100, and the *bit-turn probability* at 0.5. The method of Sobol' in SIMLAB (Sec. 3.3) is used as the sensitivity analysis method. In the Sobol' setting with two input parameters, a minimum of 96 combinations must be generated for the input sample file. The values of the mutation and crossover rates are real random numbers from a Uniform distribution in the interval (0,1). It is worthy to note that when the mutation rate is almost zero and the crossover rate is high, the EA behaves as a real-parameter genetic algorithm. When the mutation rate is high and the crossover rate is zero, the EA behaves as an evolutionary program or as an evolutionary strategy. Ninety-six samples in the input file means that the experiment should generate 96 trees for each algorithm in each dataset. This totals to 2112 trees.

Table 6.4 shows the sensitivity indices for both the SPEA2 and NSGA-II algorithms in the small datasets. The highest first-order sensitivity indices in each dataset are in **bold**. The negative values are equivalent to zeroes, which means "no effect". In the *Cancer* dataset, the mutation and crossover rates share almost the same values for SPEA2 for both accuracy and size whereas the mutation rate has a large effect on the accuracy of trees of NSGA-II. In the *Diabetes* dataset, both mutation and recombination do not have any main effect on the outputs of NSGA-II while the crossover rate has a large main effect on accuracy and size in SPEA2. In the *Glass* dataset, the input parameters do not have main effects for both algorithms. In the *Housing* dataset, the crossover rate has main effects on accuracy and size in NSGA-II. In the *Iris* dataset, there are no main effects from the input parameters in both SPEA2 and NSGA-II.

In the *Vehicle* dataset, the mutation and crossover rates have main effects on accuracy and size for SPEA2 but have none for NSGA-II. In the *Vowel* dataset, only the mutation rate has a main effect on the tree size in SPEA2 and no effect in NSGA-II. For all the small datasets, the total-order effects are significant for both algorithms in all datasets. This means that the mutation and crossover rates may not be removed as input parameters even though they may not have main effects.

Table 6.5 shows the comparison of Sobol' sensitivity indices of the large datasets. The mutation rate significantly affects the size in both SPEA2 and NSGA-II trees. In the Optical-digit dataset, only the mutation rate affects the

variation in accuracy in NSGA-II. In the pen-digits dataset, similar values are observed for both SPEA2 and NSGA-II, i.e. the mutation has main effects on accuracy and tree size.

Dataset	Sensitivity Indices		SPEA2		NSGA-II	
			Accuracy	Size	Accuracy	Size
Cancer	First-order	mutation	0.326	**0.404**	**0.782**	-0.291
		crossover	0.352	0.330	0.138	-0.468
	Total-order	mutation	0.638	0.868	1.077	1.286
		crossover	0.664	0.794	0.433	1.109
Diabetes	First-order	mutation	-0.285	-0.263	-0.798	-0.961
		crossover	**0.794**	**0.721**	-0.682	-0.823
	Total-order	mutation	0.561	0.498	1.051	1.269
		crossover	1.640	1.482	1.167	1.408
Glass	First-order	mutation	-0.008	0.173	-0.424	0.065
		crossover	-0.031	**0.194**	-0.202	**0.088**
	Total-order	mutation	0.766	0.390	1.056	1.099
		crossover	0.743	0.411	1.278	1.121
Housing	First-order	mutation	0.153	0.340	-0.162	-0.014
		crossover	-0.289	-0.070	**0.476**	**0.533**
	Total-order	mutation	1.009	1.098	0.604	0.483
		crossover	0.628	0.775	1.242	1.030
Iris	First-order	mutation	-0.040	0.000	0.008	0.000
		crossover	-0.017	0.000	**0.176**	0.000
	Total-order	mutation	0.769	0.000	0.699	0.000
		crossover	0.792	0.000	0.867	0.000
Vehicle	First-order	mutation	0.326	0.377	-0.361	-0.476
		crossover	**0.548**	**0.575**	-0.341	-0.432
	Total-order	mutation	0.646	0.792	0.801	0.875
		crossover	0.869	0.989	0.821	0.918
Vowel	First-order	mutation	-0.020	**0.411**	-1.190	-0.200
		crossover	**0.100**	-0.023	-1.400	-0.386
	Total-order	mutation	0.760	1.091	2.118	1.107
		crossover	0.880	0.652	1.901	0.922

Table 6.4 Sobol' sensitivity indices of small datasets

Table 6.6 lists the input parameters that may be assigned a fixed value in order to reduce the variation in the corresponding output variable or variables. The input parameters listed have at least a main effect of 50% on the variation in the output and the values shown are their suggested values in order to generate

trees of higher accuracy or trees of smaller size. Table 6.6 also shows that the mutation rate has a significant effect on the variation of the tree size in the large datasets.

Dataset	Sensitivity Indices		SPEA2		NSGA-II	
			Accuracy	Size	Accuracy	Size
RCB	First-order	mutation	0.170	0.604	**0.301**	**1.040**
		crossover	-0.226	-0.229	0.002	0.023
	Total-order	mutation	1.127	1.215	0.943	1.192
		crossover	0.907	0.435	0.644	0.174
Optical-digits	First-order	mutation	-0.552	-0.223	**0.775**	**0.261**
		crossover	-0.575	-0.244	0.354	-0.030
	Total-order	mutation	1.373	1.138	0.927	0.778
		crossover	1.350	1.117	0.507	0.487
Pen-digits	First-order	mutation	0.496	0.670	**0.534**	**0.673**
		crossover	0.493	-0.072	0.286	0.025
	Total-order	mutation	0.801	1.226	0.931	1.053
		crossover	0.799	0.484	0.683	0.405

Table 6.5 Sobol' sensitivity indices of large datasets

Dataset	SPEA2		NSGA-II	
	Accuracy	Size	Accuracy	Size
Cancer			Mutation (≥0.70)	
Diabetes	Crossover (≥0.50)	Crossover (≥0.50)		
Housing				Crossover (≥0.65)
Vehicle	Crossover (≥0.70)	Crossover (≥0.70)		
RCB		Mutation (≥0.25)		Mutation (≥0.45)
Optical-digits			Mutation (≥0.60)	
Pen-digits		Mutation (≥0.40)	Mutation (≥0.50)	Mutation (≥0.70)

Table 6.6 Input parameters that have main effects of at least 50%

6.4 Sample Applications

Table 6.7 presents the details of two application datasets. The Bankruptcy dataset contains financial measurements that determine an establishment's ability to pay. There are 220 "can pay" samples and 72 "cannot pay" samples. A 5-fold cross-validation is sufficient to show the average training accuracy of the decision trees induced from this dataset. The four types of classifiers, as described previously, are compared in terms of training accuracy and size, and accuracy on each of class. The Funfair dataset measures the general performance of an amusement facility based on patron perception over a variety of predictors. The decision tree however, does not classify the amusement facility according to general performance but classifies the types of patrons according to their perception. There are three classes of patrons, *not satisfied* (214 samples), *satisfied* (1,873 samples), and *very satisfied* (1,254 samples). A separate training set is used to induce the decision trees and another dataset to test them. Due to the large size of the training set, a 10-fold cross-validation is ideal.

In order to compare the performance of the classifiers, each dataset is classified on four different settings: (1) the raw dataset as is, i.e. without preprocessing, (2) the dataset is cleaned using imputation, (3) the dimensions are reduced via consistency-based feature selection (CBF), and (4) the dimensions are reduced via correlation-based feature selection (CFS).

Name of dataset	Number of samples *n*	Number of predictors *d*	Number of Classes *c*
Bankruptcy	292	24	2
Funfair	3343 (training set) 690 (test set)	54	3

Table 6.7 Description of sample-application datasets.

Table 6.8 shows the results of four Bankruptcy datasets along four classifiers based on tree size, training accuracy, and accuracy of each class. *Class 1* has 220 instances and *Class 2* has 72 instances. The Bank-O refers to the raw Bankruptcy dataset with 292 instances, 24 attributes and 423 missing values. All the classifiers replace missing values with the mean. The Bank-I refers to the Bankruptcy dataset where missing values were replaced using Expectancy

Maximization imputation. The Bank-CBF refers to the Bankruptcy dataset after a consistency-based feature selection was applied. The number of attributes was reduced to eleven. The Bank-CFS refers to the Bankruptcy dataset after a correlation-based feature selection was applied. The number of attributes was reduced to one.

Bankruptcy Dataset		AP	OC1	SPEA2	NSGA-II
Bank-O 24 attributes	Training Accuracy	72.95	73.97	73.97	**74.66**
	Class 1	90.00	**94.09**	93.18	91.36
	Class 2	20.83	12.50	15.28	**23.61**
	Average Size	6.8	**2.40**	3.80	5.40
Bank-I 24 attributes	Training Accuracy	66.10	**71.23**	68.5	66.10
	Class 1	83.18	**88.73**	85.45	82.27
	Class 2	13.89	**20.83**	15.28	16.67
	Average Size	13.2	**5.60**	6.00	12.40
Bank-CBF 11 attributes	Training Accuracy	71.23	**72.60**	**72.60**	71.23
	Class 1	91.82	88.18	**95.00**	87.73
	Class 2	8.33	**25.00**	4.17	20.83
	Average Size	4.60	5.80	**2.60**	13.00
Bank-CFS 1 attribute	Training Accuracy	75.34	75.34	75.34	75.34
	Class 1	100	100	100	100
	Class 2	0	0	0	0
	Average Size	2.00	2.00	2.00	2.00

Table 6.8 Comparison of algorithms on accuracy and tree size

The results show that there are no significant differences among the classifiers in their average training accuracy and accuracy per class (best results are in **bold**). The oblique DTs perform better than the AP classifier in all datasets. In addition, among the oblique classifiers, OC1 performs best on the different Bankruptcy datasets. On the other hand, all the classifiers perform better on the raw dataset than on the "clean" datasets Bank-I and Bank-CBF. This means that the classifiers perform better when missing values are replaced by the means of their respective attributes than when missing values are imputed. Similarly, feature selection using consistency selection did not positively affect the

classification accuracy of all the classifiers. However, when the dimension of the raw dataset reduces to one attribute using correlation, the classification accuracy of all classifiers improves and in this case creates the best decision tree. The decision tree generated in this dataset has only two leaves and classifies *Class 1* without any error but misclassifies all instances of *Class 2*, which make the transformation to a decision rule simple.

Figure 6.2 shows a sample run of the axis-parallel classifier and the classification results of the Bank-CFS dataset from the rule "If (x1 <= 1.215) then *Class 1*".

```
C:\oc1dos\trial>mktree -Dbankruptcycfs.dt -Tbankruptcy.csv -v
292 testing examples loaded from bankruptcy.csv.
Decision tree read from bankruptcy.dt.
accuracy = 75.34        #leaves = 2.00  max depth = 1.00

Class   1 : accuracy = 100.00% (220/220)
Class   2 : accuracy = 0.00% (0/72)
```

Figure 6.2 Classification results from the AP classifier (Bankruptcy)

Table 6.9 shows the classification results of four Funfair datasets. For the training set, *Class 1* has 1,254 instances, *Class 2* with 1,873 instances, and *Class 3* with 214 instances. For the test set, *Class 1* has 333 instances, *Class 2* with 267 instances, and *Class 3* with 90 instances. The Fun-O refers to the raw Funfair dataset, a training set with 28,455 missing values and a test set with 4,928 missing values. All the classifiers replace missing values with their mean when preprocessing of a dataset is omitted. The Fun-I refers to the Funfair dataset where missing values were replaced using Expectancy Maximization imputation. The Fun-CBF refers to the Funfair dataset after a consistency-based feature selection was applied. The number of attributes is reduced to 15. The Fun-CFS refers to the Funfair dataset after a correlation-based feature selection was applied. The number of attributes is reduced to 19.

The results along the classifiers do not show any significant differences in terms of accuracy on the training set and test set (best results are in **bold**). In the raw Funfair dataset, the SPEA2 classifier gives the highest training set accuracy whereas the AP classifier gives the highest accuracy in the test set.

Funfair Dataset		AP	OC1	SPEA2	NSGA-II
Fun-O 54 attributes	Training Accuracy	64.35	64.29	**64.71**	63.33
	Class 1	46.89	49.76	49.84	**50.16**
	Class 2	**81.63**	79.55	81.1	78.86
	Class 3	15.12	**15.89**	8.41	9.35
	Average Size	16.6	15.6	**5.4**	6.6
	Test set Accuracy	**64.93**	64.64	62.9	62.03
	Class 1	70.27	69.97	69.67	**72.67**
	Class 2	74.91	71.91	**75.66**	61.05
	Class 3	15.56	23.33	0	**25.56**
Fun-I 54 attributes	Training Accuracy	**65.55**	64.5	60.84	64.28
	Class 1	**50.08**	46.33	49.09	43.22
	Class 2	**82.49**	82.22	82.22	86.17
	Class 3	7.94	**15.89**	**15.89**	8.41
	Average Size	13.4	6.8	3.3	**3**
	Test set Accuracy	62.17	**62.75**	61.59	61.3
	Class 1	77.18	65.15	66.67	64.56
	Class 2	64.42	74.53	76.03	**77.9**
	Class 3	0	**18.89**	0	0
Fun-CBF 15 attributes	Training Accuracy	64.98	64.05	**65.04**	63.15
	Class 1	47.85	46.97	**50.32**	44.18
	Class 2	**82.17**	81.05	80.78	81.37
	Class 3	14.95	**15.42**	13.55	14.95
	Average Size	10	7.6	**6.2**	13.6
	Test set Accuracy	**63.33**	61.74	62.61	59.71
	Class 1	65.17	66.37	**81.08**	58.26
	Class 2	**77.15**	76.78	56.18	**77.15**
	Class 3	**15.56**	0	13.33	13.33
Fun-CFS 19 attributes	Training Accuracy	**64.47**	64.44	64.17	64.02
	Class 1	47.29	**52.71**	49.52	48.96
	Class 2	**81.85**	77.31	79.33	79.6
	Class 3	13.08	**20.56**	18.22	15.89
	Average Size	19.6	27.6	19.6	**14**
	Test set Accuracy	61.3	61.59	**63.77**	62.46
	Class 1	61.26	59.76	63.36	**64.86**
	Class 2	76.78	76.4	**80.52**	75.28
	Class 3	**15.56**	24.44	**15.56**	**15.56**

Table 6.9 Comparison of algorithms on accuracy and tree size

The opposite is true for the reduced dataset using CFS, the AP classifier gives the highest training set accuracy whereas the SPEA2 classifier gives the highest accuracy in the test set. In the imputed dataset, the highest training accuracy is induced by the AP classifier and the highest test set accuracy by the OC1

classifier. The highest training accuracy for the reduced dataset using CBF is induced by SPEA2 and the highest test accuracy by the AP classifier.

All the preprocessed datasets (Fun-I, Fun-CBF, and Fun-CFS) improves the training set accuracy of most of the classifiers but worsen their test set accuracy. The best test accuracy for all classifiers is induced from the raw dataset, which means that preprocessing the Funfair dataset does improve classification accuracy.

The SPEA2 and AP classifiers induce better trees than the OC1 and NSGA-II. That AP classifier is more convenient for transformation to decision rules as it requires not more than eight nested if-then-else statements to classify the Funfair dataset. The SPEA2 classifier may also be utilized as it requires less if-then-else statements but transforming a linear combination of attributes to a decision rule remains a problem. Figure 6.3 shows the oblique decision tree induced by SPEA2 from the Funfair dataset. Although each node is composed of a linear combination of all the predictors, important variables can be identified. These variables have high-valued coefficients. The decision tree in this case is easy to interpret since only one predictor remains in each hyperplane.

```
Training set: funfair.csv, Dimensions: 54, Classes: 3


Root Hyperplane: Left = [263,811,131], Right = [657,515,29]
0.958000 x[1] -8.245000 = 0

r Hyperplane: Left = [304,381,22], Right = [353,134,7]
0.991300 x[2] - 9.500000 = 0

rl Hyperplane: Left = [220,334,16], Right = [84,47,6]
-0.403200 x[11] -0.423000 x[17] +  0.218600 x[19] +  0.310400 x[22]
- 0.288900 x[25] + 0.331900 x[29] + 0.984500 x[38] +
0.347000 x[41] -0.302600 x[54] -8.648400 = 0
```

Figure 6.3 SPEA2 decision tree induced from the Funfair dataset

Figure 6.4 shows a sample run of the SPEA2 classifier and the classification results of the raw Funfair dataset from the decision tree in Figure 6.3. The classifiers have generated varying classification accuracies and tree sizes, and none of them performed consistently in all the datasets. Oblique classifiers are best suited for generating trees for the Bankruptcy dataset when preprocessing

is omitted. Either the AP or SPEA2 classifiers may be utilized for the Funfair classifier.

```
C:\pisa\dtea>mktree -Dfunfairspea.dt -Tfunfairtest.csv -v
690 testing examples loaded from funfairtest.csv.
4928 missing values filled with respective attribute means.
Decision tree read from funfairspea.dt.
accuracy = 62.90        #leaves = 4.00  max depth = 3.00

Class   1 : accuracy = 69.67% (232/333)
Class   2 : accuracy = 75.66% (202/267)
Class   3 : accuracy = 0.00% (0/90)

C:\pisa\dtea>
```

Figure 6.4 Classification results from the SPEA2 classifier (Funfair)

.

6.5 Summary

This chapter presents the application and analysis of evolutionary algorithms in Decision Tree construction and optimization. Two evolutionary algorithms of different selection schemes are compared to an axis-parallel classifier, and a hill-climbing oblique classifier. The experiments show that the EAs generate comparable results with the latter methods in terms of classification accuracy and tree size. However, their computing time takes longer in all the datasets. The best results of the EAs become evident only in the large UCI databases. None of the classifiers dominated the UCI datasets with respect to inducing the optimal tree. This observation is also evident in the sample application datasets. The use of preprocessing techniques that clean the application datasets using imputation and that reduce their dimension using feature selection did not significantly improve the classification accuracy of all classifiers.

The sensitivity analysis shows that there are considerable dissimilarities between the main effects of mutation and recombination in all the UCI datasets, which means that none of them performs better in the induction of decision trees. However, the sensitivity analysis indicates that in the large datasets, the mutation rate significantly affects the tree size. With regard to the sensitivity indices between SPEA2 and NSGA-II, they differ in all the datasets and in most

cases, show differing sensitivity indices, which implies that they treat the nondominated solutions differently.

The evolutionary algorithms in the above experiments are imbedded functions in the induction of the decision tree. They act as search algorithms that select a predictor or a linear combination of predictors which best splits the dataset to create small trees with high classification accuracy. In this respect, the tree size is a consequence of the goodness of a split. Our experiments have produced an ensemble of decision trees of varying accuracies and sizes and an approximation of the Pareto set can be easily calculated.

In the true sense of evolutionary multiobjective optimization, the ensemble of decision trees induced in our experiments can be compared to a population of decision trees in one generation of an MOEA wherein both objective functions of maximizing accuracy and minimizing size are evaluated simultaneously.

Chapter 7

Multicriteria Performance Analysis of Nondominated Sets

7.1 Introduction

The multiobjective evolutionary algorithm experiments in the previous chapters have generated a variety of nondominated sets in each problem domain. Comparisons of the quality of the nondominated sets have not yet been presented in each of the cases except for the solution sets in Chapters 3 and 5. In the Competitive Facility Location problem, the quality of the nondominated sets generated by the MOEA were calculated and compared to the Pareto-optimal set generated by Carrizosa and Plastria (1995). The *error ratio* metric, which measures the closeness of the nondominated set to the Pareto front in terms of set membership, was used to measure the quality of solutions in the CFLP. In the Multiobjective Shortest Paths problem, the nondominated paths generated by the MOEAs were compared to the nondominated paths generated by Martins algorithm (1999). The comparisons were possible because the Pareto-optimal sets for the CFLP experiment are known and can be computed. However, in the Container Location model, the MSPP and the Decision Tree experiments, the Pareto-optimal sets are unknown and cannot be computed by deterministic methods. Hence, only approximations to their Pareto-optimal sets are available for performance analysis.

This chapter presents some performance metrics that are useful in measuring the quality of nondominated sets when the Pareto-optimal set is unknown, and utilizes a multicriteria tool to determine the best nondominated set for the CLM, MSPP, and DT problems based on the performance metrics. Several performance metrics exist in literature, and several comparative studies have been conducted that evaluate them (see Section 2.3.3).

7.2 Performance Measures

There are three main classifications of performance metrics for evaluating the quality of nondominated sets. The first classification evaluates the convergence or the proximity of the non–dominated set to the Pareto-front based on several concepts (e.g. dominance, distance, and set membership). The second classification evaluates the diversity of a non–dominated set by calculating the spread of solution along its front. The third classification evaluates both convergence and diversity.

Hypervolume

The hypervolume metric (Zitzler and Thiele 2000) calculates the volume covered by the members of the nondominated set $\boldsymbol{Q}$. For each solution $i \in \boldsymbol{Q}$, a hypercube $\boldsymbol{v}_i$ is computed from a reference point and the solution i as the diagonal corners of the hypercube, The reference point can be found by constructing a vector of worst objective function values. The hypervolume (HV) is calculated as:

$$HV = volume\left(\bigcup_{i=1}^{|Q|} v_i\right) \tag{7.1}$$

The hypervolume is a metric that measures both convergence and diversity of a nondominated set.

Spacing

Schott (1995) introduced a metric, which is a measure of the relative distances between consecutive solutions in the nondominated set $\boldsymbol{Q}$ is calculated as:

$$S = \sqrt{\frac{\sum_{i=1}^{|Q|}\left(d_i - \bar{d}\right)}{|\boldsymbol{Q}|}} \tag{7.2}$$

where $$d_i = min_{k \in Q \wedge k \neq i} \sum_{m=1}^{M} \left| f_m^i - f_m^k \right| \tag{7.3}$$

$$\bar{d} = \frac{\sum_{i-1}^{|Q|} d_i}{|\boldsymbol{Q}|} \tag{7.4}$$

Schott's metric measures the diversity of a nondominated set.

Set coverage metric

This metric is based on Zitzler (1999). The metric computes the relative spread of solutions between two nondominated sets $\boldsymbol{A}$ and $\boldsymbol{B}$. The set coverage metric $C(\boldsymbol{A}, \boldsymbol{B})$ calculates the proportion of solutions in $\boldsymbol{B}$ that are weakly dominated by solutions of $\boldsymbol{A}$:

$$C(\boldsymbol{A}, \boldsymbol{B}) = \frac{\left|\left\{\boldsymbol{b} \in \boldsymbol{B} \middle| \exists \boldsymbol{a} \in \boldsymbol{A} : \boldsymbol{a} \preceq \boldsymbol{b}\right\}\right|}{|\boldsymbol{B}|} \tag{7.5}$$

The metric value $C(\boldsymbol{A}, \boldsymbol{B}) = 1$ means all members of $\boldsymbol{B}$ are weakly dominated by $\boldsymbol{A}$. On the other hand, $C(\boldsymbol{A}, \boldsymbol{B}) = 0$ means that no member of $\boldsymbol{B}$ is weakly dominated by $\boldsymbol{A}$. This operator is not symmetric, thus it is necessary to calculate $C(\boldsymbol{B}, \boldsymbol{A})$.The set coverage metric measures convergence based on the concept of dominance relations.

Cardinality

This metric counts the number of solutions in the nondominated set. It measures neither diversity nor convergence.

7.3 Computations and Results

In order not to limit the description of the quality of a nondominated set by using only a single metric, a multicriteria evaluation is necessary. Hence, the computations in this chapter evaluate the quality of nondominated sets according to four criteria, which are mentioned in Section 7.2. The multicriteria tool employed in the computations is Decision Lab (Visual Decision, 2003). The Decision Lab software is multicriteria decision making software, which is based on the Preference Ranking Organization Method for Enrichment Evaluations (PROMETHEE) and the Graphical Analysis for Interactive Assistance (GAIA). The details of the PROMETHEE method are found in Bran and Mareschal (1986). GAIA, which makes use of principal component analysis, is a descriptive complement to the PROMETHEE methods (Bran and Mareschal 1994).

Table 7.1 shows the computed values for each performance criterion in the different multiobjective optimization problems. There are four criteria. A smaller set c*ardinality* is preferred. A *spread* that has smaller value means that the solutions on the on-dominated front are uniformly spaced therefore this criterion is minimized. *Hypervolume* and *set coverage* are maximized. Two nondominated sets are compared in each MOOP. One set is generated by NSGA-II and the other by SPEA2. Decision Lab can rank more nondominated sets but since the *set coverage* is a binary quality measure, only two nondominated sets can be evaluated each time.

The values of the LCFS and FCFS sets (for the Container Location Problem) are computed from a 50% space usage configuration. A 50-node of 10% density is the basis for the computed values in both MSPP (Multi-objective Shortest Path Problem) configurations. The Housing and the Optical Digits are used as the nondominated sets for the Decision Tree problem. The *hypervolume* values for the MSPP are blank since they cannot be computed. This reduces the number of MSPP criteria to three.

Figure 7.1 shows the nondominated fronts of SPEA2 and NSGA-II in the LCFS optimization problem. By visual observation, it is not clear which front is better because there are some overlapping solutions between the SPEA2 and NSGA-II sets. However, the range of solutions in SPEA2 is wider than NSGA-II, which consequently creates a larger hypervolume. SPEA2 has more solutions but

NSGA-II has better set coverage as some of its solutions dominate several SPEA2 solutions.

Nondominated Set	Criterion			
	Cardinality (Minimize)	Spread (Minimize)	Hyper-volume (Maximize)	Set Coverage (Maximize)
LCFS				
NSGA-II	6	0.2	4.76	0.41
SPEA2	12	0.17	6.59	0.17
FCFS				
NSGA-II	5	0.39	5.78	0.44
SPEA2	9	0.21	7.03	0.2
MSPP 2S|1M				
NSGA-II	7	0.8		0
SPEA2	8	0.51		0
MSPP 3S				
NSGA-II	8	0.57		0
SPEA2	5	1.14		0.38
Housing dataset				
NSGA-II	5	0.43	5.47	0.17
SPEA2	6	0.38	6.98	0.6
Optical-digits				
NSGA-II	5	0.36	5.96	0
SPEA2	6	0.63	7.15	0.6

Table 7.1 Computed criteria values of nondominated sets

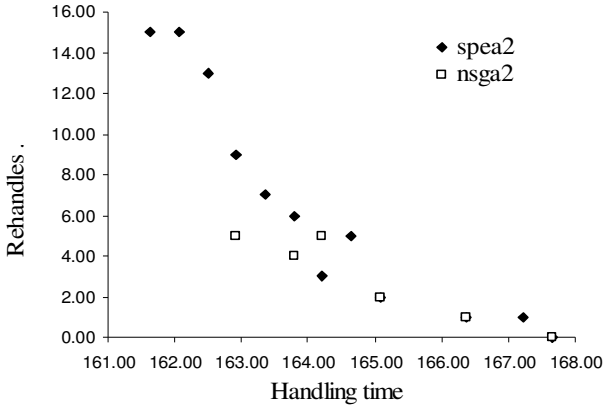

Figure 7.1 Comparison of LCFS nondominated fronts

Decision Lab has a visualization tool that shows the relation between the criteria and the nondominated set, and shows a preferred solution if it exists. Figure 7.2 shows a GAIA diagram that shows how each criterion relates to each action. The GAIA plane corresponds to the first principal components of the data, which ensures that a maximum quantity of information is available on the plane. An action in this case refers to a nondominated set. The orientation of the criteria axes indicates which criteria are in agreement with each other. In this case, *cardinality* and *set coverage* have the same orientation but opposite to *hypervolum*e and *spread*. The orientation of the position of an action indicates its strong features. The length of the axis correspond to a criterion's observed deviations between actions, the longer the axis the higher the deviation. In the example, the *set coverage* projects the longest axis, which means the difference in values between two actions in this criterion is greater, compared to the other criteria.

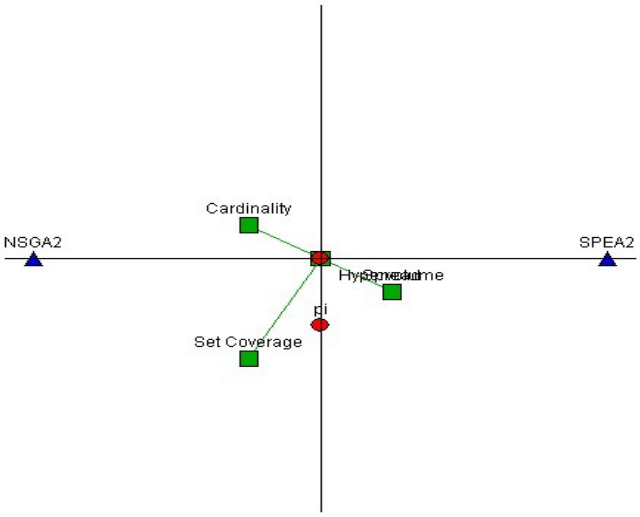

Figure 7.2 GAIA diagram for LCFS with equal weights

The orientation of the *pi*, which is the decision axis, points to the preferred action or solution considering all the criteria. In this example, *pi* does not point towards any action, which means that there is no compromise solution. This due to the condition that NSGA-II has two strong features, SPEA2 has two strong features, and all the criteria have the same weight. Adding different weights to each criterion obviously changes the orientation of the *pi* decision axis.

Figure 7.3 shows the GAIA plane wherein the weights of *hypervolume* and *set coverage* are set at 1.0, *spread* at 0.50, and *cardinality* at 0.20. Consequently, SPEA2 becomes the compromise solution.

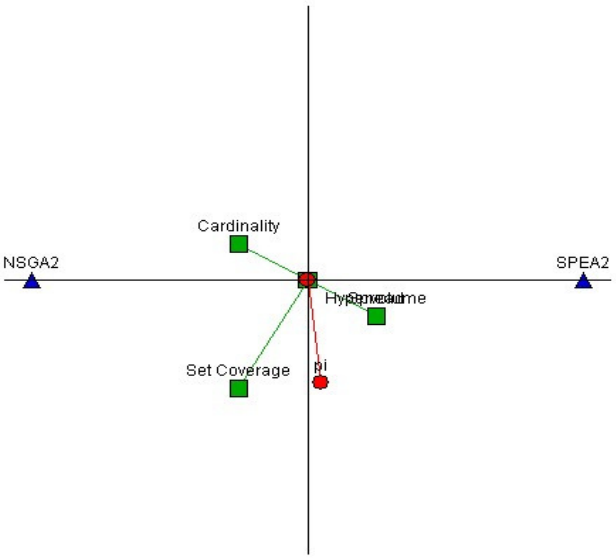

Figure 7.3 GAIA diagram for FCFS and LCFS with unequal weights

The criterion weight is independent from the scale of the criterion which means the larger the value the more important the criterion. In order to compare the different criteria independently form their measurement units, the PROMETHEE method provides six preference functions. A preference function and a preference threshold are associated with each criterion when a decision-maker compares two actions.

Table 7.2 lists the preference thresholds for the LCFS example described in Chapter 4. The threshold represents the largest deviation that is considered as decisive by the decision-maker. For example, the *cardinality* threshold means that any difference in the number of solutions between NSGA-II and SPEA2 should be important. Smaller differences correspond to lower degrees of preference. When the difference reaches the 75% threshold, then the set with the smaller cardinality is preferred in this criterion. The preference function translates the deviation between the values of two actions on a single criterion

in terms of a preference degree. The preference degree is an increasing function of the deviation. The degree of preference is expressed on a percentage scale. A linear preference function as defined by Bran and Mareschal (1986) is as follows:

$$H(d) = \begin{cases} d / p & \text{if } -p \le d \le p, \\ 1 & \text{if } d < -p \text{ or } d > p \end{cases} \qquad 7.5)$$

As long as *d* is lower than *p*, the preference of the decision maker increases linearly with *d*. If *d* becomes greater than *p*, a strict preference exists. The linear preference function is used in the succeeding computations and is associated with all the criteria listed in Table 7.2 The linear preference function was chosen since the function takes into account any difference in the values between two alternatives.

Criterion	Threshold
Cardinality	75%
Spread	50%
Hypervolume	75%
Set Coverage	50%

Table 7.2 Decision Lab preference thresholds

The criteria *spread* and *set coverage* represent diversity and convergence respectively, and they are conflicting features. Both criteria are assigned the same threshold of 50% to avoid any preference to diversity or convergence. The hypervolume, which measures diversity and convergence, is assigned a threshold of 75% to assure that there is a significant difference in the number of dominated solutions between both nondominated sets.

Figure 7.4 shows the GAIA plane when the preferences have been incorporated. The orientation of *cardinality* and *set coverage* are pointing to NSGA-II, which means that they are decisive in finding a compromise solution, whereas the *hypervolume* and *spread* criteria are not decisive, which means that their differences are small. In this case, the preferred solution to the LCFS example is the NSGA-II nondominated set.

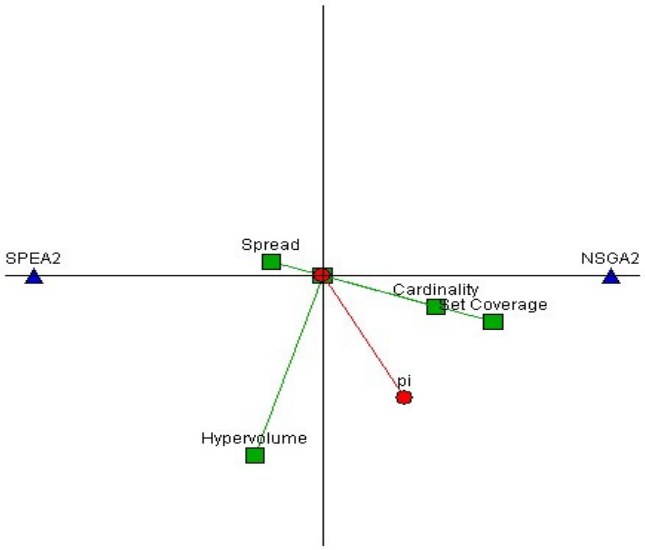

Figure 7.4 GAIA diagram for LCFS with preferences.

Figure 7.5 shows the nondominated fronts of SPEA2 and NSGA-II in the FCFS optimization problem. NSGA-II has solutions that dominate the solutions of SPEA2 when the handling time is longer than 555 minutes but has no solutions when handling time is less than 555 minutes. The range of solutions in SPEA2 is wider than NSGA-II. With this, the better set cannot be determined visually.

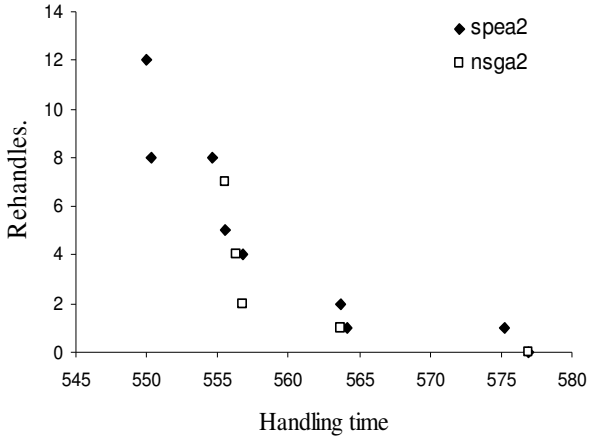

Figure 7.5 Comparison of FCFS nondominated fronts

The FCFS example has the same GAIA plane shown in Figure 7.2 when the criteria have no weights, and Figure 7.6 shows the GAIA plane when the preferences in Table 7.2 have been incorporated. The *pi* decision axis leans towards the NSGA-II option but its orientation is not as much as in the LCFS example. Still, the preferred solution is NSGA-II.

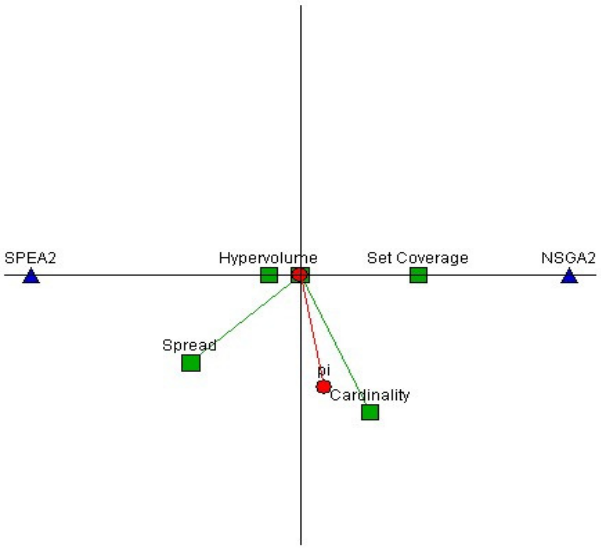

Figure 7.6 GAIA diagram for FCFS with preferences.

Decision Lab is utilized for the MSPP to show which nondominated set has better quality. Figure 7.7 shows the plane for the 2S|1M configuration. The number of criteria has reduced to three as mentioned previously. The values for *set coverage* in both sets are zero, which means that there are no weakly dominated solutions from each set or that their solutions are similar. The *pi* decision axis has no preferred solution at this point therefore preferences should be incorporated to the Decision Lab model. Figure 7.8 shows that SPEA2 is the preferred solution after incorporating the thresholds in Table 7.2, and the decisive criterion is the spread of solutions. The set coverage criterion is not a factor since both sets do not cover any weakly dominated solutions between them.

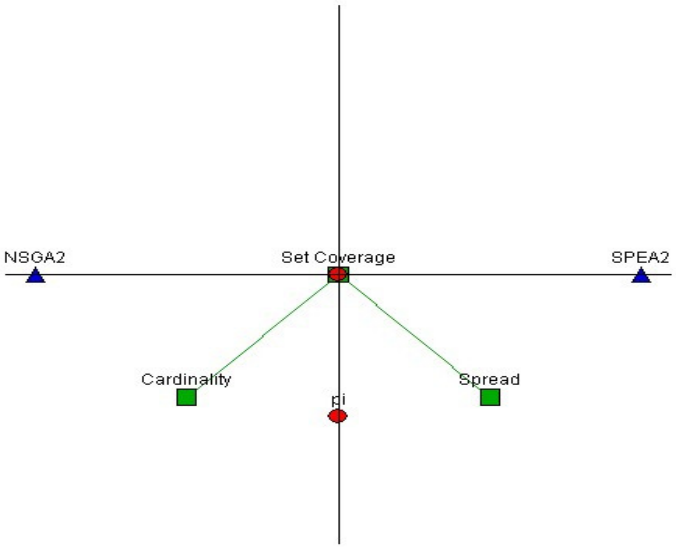

Figure 7.7 GAIA diagram for a 2S|1M configuration without preferences

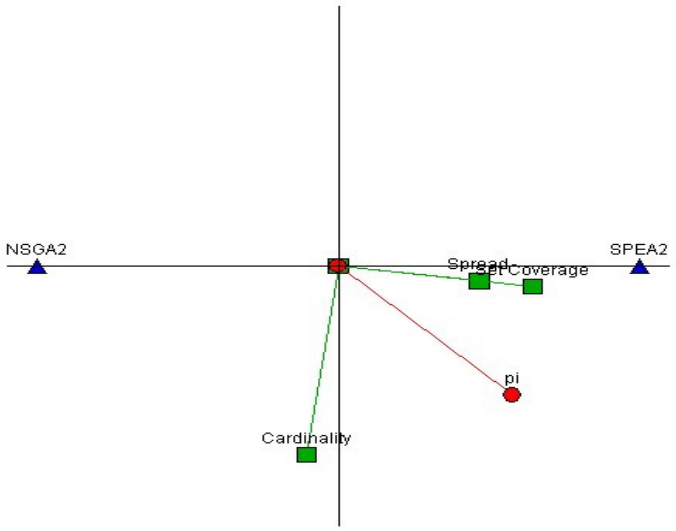

Figure 7.8 GAIA diagram for a 2S|1M configuration with preferences

In the 3S configuration as shown in Figure 7.9, the GAIA plane shows that the SPEA2 action is preferred, even without any decision-maker preference since it has two strong features whereas NSGA-II has only one. The result is the same after the preferences in Table 7.2 have been added. The decisive factors are cardinality and set coverage.

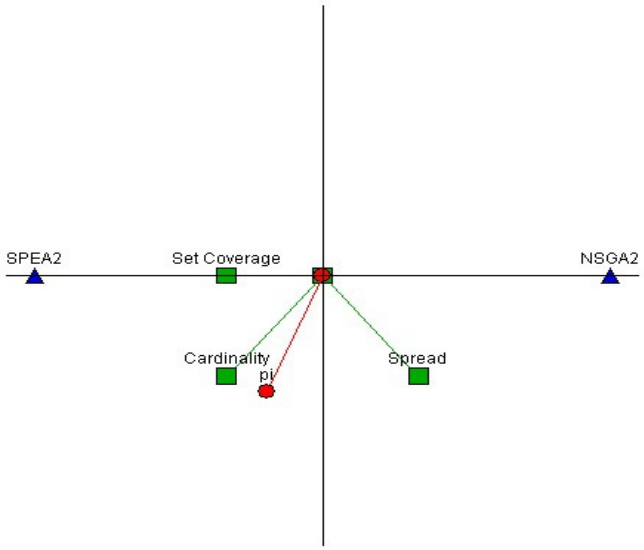

Figure 7.9 GAIA diagram for a 3S configuration without preferences

The fronts shown in Figure 7.10 are the nondominated sets from the ensembles of trees generated in Chapter 6. The OC1 solutions are dominated by the solutions of either SPEA2, or NSGA-II, or AP. The nondominated solutions of the AP classifier are dominated by either SPEA2 or NSGA-II. Most of the nondominated solutions in NSGA are dominated by solutions in SPEA2. It seems that SPEA2 is the better nondominated set through the projection of their nondominated sets but needs to be validated using the PROMETHEE method. The nondominated sets of AP and OC1 need not be tested for performance quality as their solutions are dominated by both MOEAs solutions.

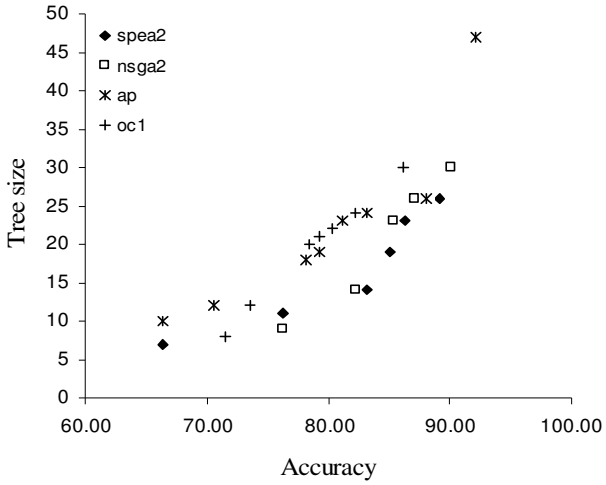

Figure 7.10 Nondominated fronts of the Housing dataset

Figure 7.11 shows the GAIA plane of the Housing dataset options, and validates that SPEA2 is the preferred solution. The factors that favor SPEA2 are the hypervolume, the spread, and the set coverage. The result does not change when preference thresholds are added. In fact, the *pi* decision axis leans more to the direction of SPEA2 when preferences are added than it does when without any preferences.

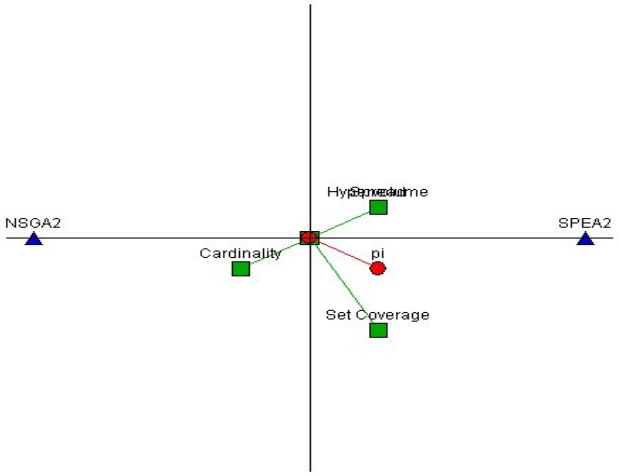

Figure 7.11 GAIA plane for the Housing dataset without preferences

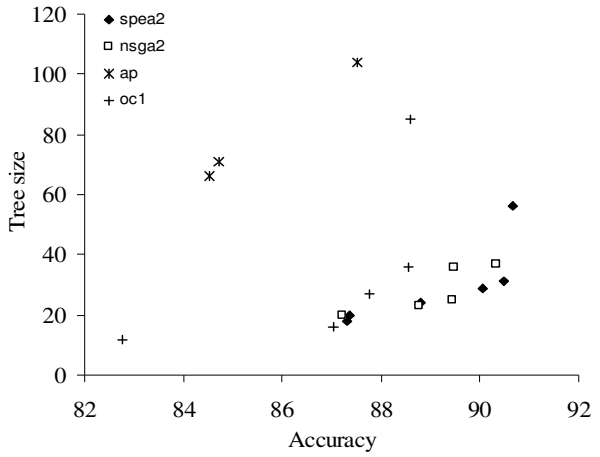

Figure 7.12 Nondominated fronts of the Optical Digits dataset

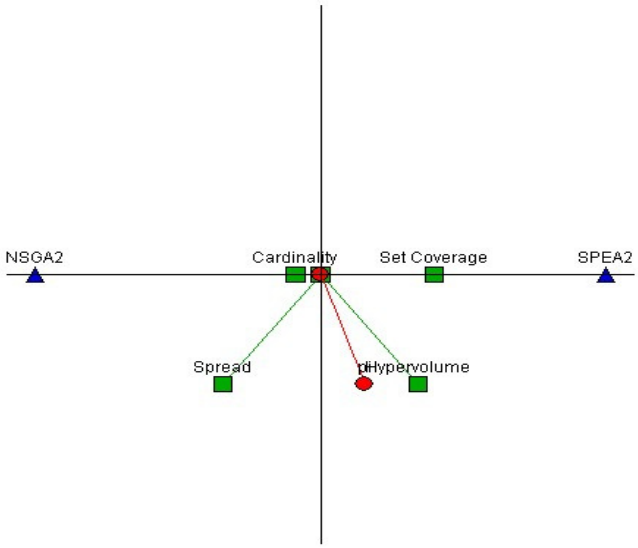

Figure 7.13 GAIA diagram for the Optical Digits dataset with preferences

Figure 7.12 shows the Optical Digits fronts, and most AP and OC1 solutions are dominated by either SPEA2 or NSGA-II, which means that AP and OC1 need not be considered as alternatives. It is difficult to determine which front is better

between SPEA2 and NSGA-II by visual observation alone, which calls for a multicriteria analysis. Similarly, a GAIA diagram (not shown) for the Optical Digits example does not show any preferred set when there are no preferences included in the computations. However, it shows that *hypervolume* and *set coverage* favor SPEA2, whereas *spread* and *cardinality* favor NSGA-II. Figure 7.13, a GAIA diagram with the preferences thresholds in Table 7.2 added, shows that the preferred alternative is SPEA2, with *hypervolume* and *set coverage* as the decisive factors.

7.4 Summary

The chapter shows the importance of a multicriteria performance analysis in evaluating the quality of nondominated sets. Six problem examples from different problem domains were analyzed on four criteria of quality. These four criteria namely *cardinality* of the nondominated set, *spread* of the solutions, *hypervolume*, and set coverage do not favor any algorithm along the six problem examples. In the CLM example, the *set coverage* and *cardinality* criteria were the decisive factors since the nondominated set of SPEA2 and NSGA-II did not differ much in terms of *hypervolume* and spread. In the MSSP examples, the *spread* of solutions is the decisive factor for the 2S|1M configuration, and the *cardinality* and *set coverage* for the 3S configuration. The difference in *set coverage* values between SPEA2 and NSGA-II in the MSPP are small since both algorithms have almost identical nondominated solutions. In the Decision Tree examples, the decisive factors are *set coverage* and *hypervolume*.

The computations show that the decisive criterion or criteria vary in all examples except for the *set coverage* criterion. This shows the importance of a binary measure in evaluating the quality of nondominated sets, as the measure itself tests for dominance.

Chapter 8

Conclusions and future work

CONCLUSIONS

The scientific objective of the dissertation is to improve the understanding of how evolutionary algorithms work in finding efficient solutions to multiobjective optimization problems through experimental research. The objective of the study is twofold: (1) to describe the performance of evolutionary algorithms in terms of stability, computational complexity, diversity and optimality of solutions in different multiobjective optimization problems, and (2) to describe their strengths and weaknesses in each of the MOOP considered in the study and identify why the MOEA succeeded or failed.

The thesis evaluated the performance of two multiobjective evolutionary algorithms on four problem sets that have different search spaces and data structure. The outputs of both MOEAs in each problem set were compared either to other algorithms or with each other, and their results with respect to each problem set were explained. The sensitivity analysis measured the effects of the input parameters on the outputs to describe stability. The multicriteria performance analysis evaluated the quality of nondominated sets in terms of diversity and optimality. The essential results are:

In terms of computational complexity, the MOEAs run in polynomial time with respect to the size of their population and are linear with respect to the

problem size. Hence, their runtime complexity does not change in the different problem sets. However, they have the worst execution time in all the problem sets. They need a large number of generations to find good nondominated solutions in both continuous and discrete search spaces. Furthermore, conducting sensitivity analyses for the MOEAs is computationally demanding in terms of execution time.

In terms of stability, the sensitivity analysis shows varying degrees of influence of the mutation and recombination rates on the output along all the problem sets and within each problem set. There are instances when the mutation rate has greater effect than the recombination rate and instances when it is the opposite. There are instances that both have small main effects and instances that their effects shift from low to high or high to low within the MOEA run. Unless the sensitivity analysis shows that the input parameters have high main effects and do not interact, fixing the values for the mutation and recombination rates that insure a stable output is difficult.

Similarly, the sensitivity indices between the MOEAs in all the test cases differ given the same input parameter values. The input parameters influence the outputs of each MOEA differently. Caution must be taken when comparing the performance of MOEAs as using the same combination of input parameters may favor one of the algorithms.

In terms of the quality of solutions, the MOEAs succeeded in finding comparable solutions to other algorithms. In the case of the decision tree problem, the MOEAs perform better in some datasets but worse on other datasets and the differences are not significant in the small datasets. The MOEAs perform better in larger datasets against a greedy algorithm and a hill climbing algorithm. In the case of the CFLP, the MOEA fails to converge to the Pareto-optimal set. For the CLM and MSPP cases, the MOEAs are compared with each other and their performance are comparable in terms of diversity and optimality in their solutions. A visual projection of two-dimensional fronts is useful in interpreting the quality of nondominated sets, but becomes worthless in higher objective spaces. A multicriteria performance evaluation proves to be beneficial in understanding the qualities of a nondominated set.

FUTURE WORK

The MOEAs did not fail to find efficient solutions to the test cases of different search spaces and structure. However, the performance of each MOEA and the influence of the genetic operators on its performance fluctuate, not only between the problem sets but also within each problem set. Due to the variation in their performance, it is difficult to characterize the problem sets on which they do well as it is difficult to find proper combinations of input parameters to have a stable output. The major difficulty lies in the fact that evolutionary algorithms combine two search strategies that behave very differently (mutation and recombination) and their behavior in searching for optimal solutions depend on the genetic representation of the optimization problem. Another drawback of an MOEA is its execution time.

The sensitivity analysis proves to be an important step in analyzing the behavior of an MOEA in terms of the main and interaction effects of its input parameters. A sensitivity analysis in each generation step better describes the behavior and the influence of each genetic operator. However, this requires much computational time. Future work on designing such experiments that reduce computing time in sensitivity analysis will be valuable.

On the issue of MOEA-execution time, future work on embedding a heuristic in the MOEA or hybrid methods will be beneficial, as a specific heuristic that suits a particular optimization may reduce the execution time and may be more efficient in finding optimal solutions for the problem sets considered in the study.

Bibliography

Abraham, A., L. C. Jain, and R. Goldberg (2005). *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. London, UK: Springer.

Benson, H.P. (1978) Existence of efficient solutions for vector maximization problems. *Journal of Optimization Theory and Applications 26*:569–580.

Beyer, H.G., and K. Deb (2000). On the desired behavior of self-adaptive evolutionary algorithms. In M. Schoenauer, K. Deb, G.Rudolph, X.Yao, E.Lutton,; J.J. Merelo, and H.P. Schwefel, eds. *Parallel Problem Solving from Nature VI*. Heidelberg:Springer. pp. 59-68.

Bleuler, S., E. Zitzler, M. Laumanns, and L. Thiele (2003). *PISA - a platform and programming language independent interface for search algorithms*. Zurich: Swiss Federal Institute of Technology TIK-Report No. 154.

Blicke, T. and L. Thiele (1995). *A comparison of selection schemes used in genetic algorithms*. Zurich: Swiss Federal Institute of Technology. Technical report nr 11.

Blum, A.L. and P. Langley (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence 97*(1):245–271.

Bosman, P. A., and D. Thierens (2003). A balance between proximity and diversity in multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation 7*(2), pp. 174-88.

Bot, M. and W. Langdon (2000). Application of genetic programming to induction of linear classification trees. In *Proceedings of the European Conference on Genetic Programming*. Edinburgh, Scotland, UK, April 2000. NewYork:Springer. pp. 247-258.

Brans, J. and B. Mareschal (1986). How to select and how to rank projects the PROMETHEE method. *European Journal of Operational Research 24*:228-238.

Brans, J. and B. Mareschal. (1994). The PROMCALC and GAIA Decision Support System for Multicriteria Decision Aid. *Decision Support Systems*(*12*):297-310.

Breiman, L. (1990). *Bagging Predictors*. Berkley (CA): Department of Statistics University of California. Technical report 421.

Breiman, L., J. Friedman, R. Olshen, and C. Stone (1984). *Classification and Regression Trees*. Belmont, Calif.: Wadsworth.

Bryson, N. (2004). Evaluation of decision trees: a multicriteria approach. *Computers and Operations Research 31*:1933-1945.

Buntine, W. (1992). Learning classification trees. *Statistics and Computing 2*:63-73.

Cantu-Paz E. and C. Kamath (2003). Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation 7*(1):54-68.

Carrizosa, E. and F. Plastria (1995). On minquantile and maxcovering optimization. *Mathematical Programming 71*:101-112.

Chaiyaratana, N., T. Piroonratana, and N. Sankawelert (2007). Effects of diversity control in single-objective and multiobjective genetic algorithms. *Journal of Heuristics 13*(1):1-34.

Chan, K., S. Tarantola, A. Saltelli (1997). Sensitivity analysis of model output: variance-based methods make a difference. In *Proceedings of the 1997 Winter Simulation Conference*. Atlanta, GA, USA. pp. 261-268.

Chen, P., Z. Fu, A. Lim, and B. Rodrigues (2004). Port yard storage optimization. *IEEE Transactions on Automated Science and Engineering 1*(1):26-37.

Chen, T. (1999). Yard operations in the container terminal-a study in the 'unproductive moves'. *Maritime Policy and Management 26*:27-38.

Coello, C. (2000). An updated survey of GA-based multiobjective optimization techniques. *ACM Computing Surveys 32*(2):109–143.

Coello, C., D. Veldhuizen, and Lamont G. (2002). *Evolutionary Algorithms for Solving Multiobjective Problems*. Boston: Kluwer.

Collette Y. and P. Siarry (2003). *Multiobjective Optimization*. Berlin: Springer.

Corne, D. W., J. D. Knowles, and M. J. Oates (2000). The Pareto envelope-based selection algorithm for multiobjective optimization. *Lecture Notes in Computer Science 1917*:839-848.

Corne, D. W., N. R. Jerram, J. D. Knowles, and M. J. Oates (2001). PESA-II: region-based selection in evolutionary multiobjective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA, July 2001. pp. 283-290.

Coutinho-Rodrigues, J., J. Climaco, and J. Current (1999). An interactive bi-objective shortest path approach: searching for unsupported non-dominated solutions. *Computers & Operations Research 26*(8):789-798.

Crichigno, J. and B. Baran (2004). A multicast routing algorithm using multiobjective optimization. In *Lecture Notes in Computer Science 3124*, pp. 1107-1113.

Czyzak P. and A. Jaszkiewicz (1998). Pareto simulated annealing- a meta-heuristic technique for multiobjective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis* 7(1):34–37.

Das, I. and, J. Dennis (1997). A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. *Structural Optimization 14*(1), 63–69.

Dash, M. and H. Liu (1997). Feature selection for classification. *Intelligent Data Analysis 1*(4), 131–156.

Deb, K. (1999). Evolutionary algorithms for multicriterion optimization in engineering design. In K. Miettinen, Marko M., Neittaanm, P., and Periaux, J. eds. *Evolutionary Algorithms in Engineering and Computer Science*, UK:Wiley. pp. 135-169.

Deb, K. (2001). *MultiObjective Optimization Using Evolutionary Algorithms*. Chichester, UK: Wiley.

Deb, K., and S. Agrawal (1999). Understanding interactions among genetic algorithm parameters. In W. Banzhaf and C. Reeves, Eds. *Foundations of Genetic Algorithms* 5. CA:Morgan Kaufmann. pp. 265-268.

Deb, K., M. Mohan, and S. Mishra (2005). Evaluating the ϵ-domination based multiobjective evolutionary algorithm. *Evolutionary Computation 13*(4):501-525.

Deb, K., S. Agrawal, A. Pratap, and T. Meyarivan (2002). A fast elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation 6*(2):182-197.

Decision Lab 2000. *Decision Lab 2000 Executive Edition Manual*. Montreal, Canada:Visual Decision Inc.

Dekker, R., P. Voogd, and E. van Asperen (2006). Advanced methods in container stacking. *OR Spectrum 28*(4):563-586.

Drezner Z. (1981). On a modified one-center problem, *Management Science 27*(7):848-851.

Drezner, T. (1994). Locating a single new facility among existing, unequally attractive facilities. *Journal of Regional Scie*nce *34*(2):237-252.

Dumitrescu, D., B. Lazzerini, L.C. Jain, and A. Dumitrescu (2000). *Evolutionary Computation*. Boca Raton (FA): CRC Press.

Ehrgott, M. (2005). *Multicriteria Optimization 2nd ed*. Berlin: Springer.

Ehrgott, M. and D. Ryan (2002). Constructing robust crew schedules with bicriteria optimization. *Journal of Multicriteria Decision Analysis 11*(3):139-150.

Ehrgott, M. and X. Gandibleux (2000). A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum 22*(4):425-460.

Eiselt H.A. and G. Laporte (1988). Location of a new facility on a linear market in the presence of weights. *Asia-Pacific Journal of Operational Research 5*:160-165.

Eiselt H.A. and G. Laporte (1989). The maximum capture problem in a weighted network. *Journal of Regional Science, 29*(3):433-439.

Eiselt, H.A., G. Laporte, and J-F. Thisse (1993). Competitive Location Models: A Framework and Bibliography. *Transportation Science 27*(1):44-54.

Engau,A. and M. Wiecek (2007). Generating ε-efficient solutions in multiobjective programming. *European Journal of Operational Research 177*(3):1566-1579.

Eshelman, L.J., R. Caruana, and J.D. Schaffer (1989). Biases in the crossover landscape. In *Proceedings of 3rd International Conference on Genetic Algorithms*,Virginia, USA. pp. 10-19.

Fayyad, U. and K. Irani (1990). What should be minimized in a decision tree?. In *Proceedings of the National Conference on Artificial Intelligence*. Boston, USA. pp. 749-754.

Fogel, D.B. (1992). *Evolving Artificial Intelligence*. Ph, D. Thesis, San Diego ,CA: University of California.

Folino, G., C. Pizzuti, and G. Spezzano (2000). Genetic Programming and Simulated Annealing: a Hybrid Method to Evolve Decision Trees. In *Proceedings of the European Conference on Genetic Programming*. Edinburgh, Scotland, UK. pp. 294-303.

Fonseca, C. and P. Fleming (1993). Genetic algorithms for multiobjective optimization: formulation, discussion, and generalization. In *Proceedings of the 5th international conference on genetic algorithms*. Urbana-Champaign, IL, USA. pp. 416 – 423.

Gandibleux, X., F. Beugnies, and S. Randriamasy (2006). Martins' algorithm revisited for multiobjective shortest path problems with a MaxMin cost function. *4OR A Quarterly Journal of Operations Research 4*:47-59.

Gandibleux, X., N. Mezdaoui, and A. Fréville. (1997). A Multiobjective tabu search procedure to solve combinatorial optimization problems. In R. Caballero, F. Ruiz, and R. Steuer eds. *Advances in Multiple Objective and Goal Programming.* Springer. pp. 291-300.

Garey, J. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. San Francisco, CA:Freeman.

Gelfand, S., C. Ravishankar, and E. Delp (1991). An iterative growing and pruning algorithm for classification tree design. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*(2):163-174.

Gen, M. and L. Lin (2004). Multiobjective genetic algorithm for solving network design problem. Presented at the 20th Fuzzy Systems Symposium, Kitakyushu, Japan.

Glover, F. (1989). Tabu search - Part I. *ORSA Journal on Computing 1*:190-206.

Goh, C. K. and K. C. Tan (2007). An investigation on noisy environments in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation 11*(3):354-381.

Goldberg, D. E. (1989). *Genetic Algorithms for Search, Optimization, and Machine Learning*. MA: Addison Wesley.

Goldberg, D.E. (1985). *Optimal initial population size for binary-coded genetic algorithms.* Tuscaloosa: University of Alabama. Technical report nr TCGA Report 85001.

Goldberg, D.E. and K. Deb (1991). A comparison of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms 1*, pp. 69-93.

Goldberg, D.E., K. Deb, and D. Thierens (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instruments and Control Engineers 32*(1), 10-16.

Goldberg, D.E., K. Deb, and J.H.Clark (1992). Genetic algorithms, noise, and sizing of populations. *Complex Systems 6*:333-362.

Granat, J. and F. Guerriero (2003). The interactive analysis of the multicriteria shortest path problem by the reference point method. *European Journal of Operational Research 151*(1):103-111.

Guddat, J., F. Guerra Vasquez, K. Tammer and K. Wendler (1985). *Multiobjective and stochastic optimization based on parametric optimization*. Berlin:Akademie-Verlag.

Guerriero, F. and R. Musmanno (2001). Label correcting methods to solve multicriteria shortest path problems. *Journal of Optimization Theory and Applications 111*(3):589-613.

Guyon, I. and A. Elisseeff (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research 3*:1157–1182.

Haimes, Y., L. Lasdon, and D. Wismer (1971). On biobjective formulation of the problem of integrated system identification and system optimization. *IEEE transactions on System, Man, and Cybernetics 1*:296-297.

Hajela, P. and C. Y. Lin (1992). Genetic search strategies in multicriterion optimal design. *Structural Optimization 4*(2):99-107.

Hansen, P. (1979). Bicriteria path problems. In *Lecture Notes in Economics and Mathematical Systems* 177, pp.109-127.

Hansen, M. (1997). Tabu search for multiobjective optimization: MOTS. *13th International Conference on Multiple Criteria Decision Making*. Cape Town, South Africa.

Harik, G., E. Cantu-Paz, D. Goldberg, and B. Miller (1999). The gamblers ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation 7*(3): 231–253.

Heath, D., S. Kasif , and S. Salzberg (1993a). *k*-DT: A multitree learning method. In *Proceedings of the 2nd International Workshop on Multistrategy Learning,* May 1993, West Virginia, USA. pp. 138-149.

Heath, D., S. Kasif , and S. Salzberg (1993b). Induction of oblique decision trees. In *International Joint Conference on Artificial Intelligence*. September 1993, Chambery, France pp 1002-1007.

Holland, J. H. (1976). Adaptation. In *Progress in Theoretical Biology 4*, New York: Academic Press.

Homma, T. and A. Saltelli (1996). Importance measures in global sensitivity analysis of nonlinear models. *Reliability Engineering and System Safety 52*:1-17.

Horn, J., N. Nafptlois and D. Goldberg (1993). *A niched Pareto genetic algorithm for multiobjective optimization.* In *Proceedings of the First IEEE Conference on Evolutionary Computation*, June 1994, Florida ,USA. pp.82-87.

Jain, A. and D. Zongker (1997). Feature selection: Evaluation, application, and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*(2):153–158.

Janssens, G.K. and J.M. Pangilinan (2008). Robustness analysis of parameter settings in a genetic algorithm for the multiobjective shortest path problem. In *Proceedings of the International Conference on Information Technologies (InfoTech-2008) 2*. Varna, Bulgaria. pp. 115-124.

Kim, D. (2004). Structural Risk Minimization on decision trees using an evolutionary multiobjective optimization. In *Lecture Notes in Computer Science 3003.* p338-348.

Kim, K.H. and K. Park (2003). A note on a dynamic space-allocation method for outbound containers. *European Journal of Operational Research 148*(1):92-101.

Kim, K.H., and J.W. Bae (1998). Re-marshalling export containers in port container terminals. *Computers and Industrial Engineering 35*(3-4):665-658.

Kim, K.H., and K.Y. Kim (1999). An optimal routing algorithm for a transfer crane in port container terminals. *Transportation Science 33*(1):17-33.

Kim, K.H., J.S. Kang, and K.R. Ryu (2004). A beam search algorithm for the load sequencing of outbound containers in port container terminals. *OR Spectrum 26*(1):93-116

Kim, K.H., Y.M. Park, and K.R. Ryu (2000). Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research 124*(1):89-101.

Knowles, J. D. and D.W. Corne (2000a). Approximating the non-dominated front using the Pareto archived evolutionary strategy. *Evolutionary Computation 8*(2):149-172.

Knowles, J. D. and D.W. Corne (2000b), M-PAES: a memetic algorithm for multiobjective optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation*, July 2000, San Diego, CA. pp.325-332

Konak, A., D. Coit, and A. Smith (2006). Multiobjective optimization using genetic algorithms: a tutorial. *Reliability Engineering & System Safety 91*(9):992-1007.

König, F. and M. Lübbecke (2008). Sorting with Complete Networks of Stacks. In *Proceedings of the 19th International Symposium on Algorithms and Computation,* December 2008, Gold Coast, Australia. pp.895-906.

Kononenko I., and I. Bratko (1991). Information based evaluation criterion for classifier's performance. *Machine Learning 6*(2):67-80.

Koza, J. (1989). Hierarchical genetic algorithms operating in populations of computer programs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence,* San Mateo, CA. pp. 768-774.

Koza, J. (1991). Concept Formation and decision tree induction using the genetic programming paradigm. In *Parallel Problem Solving from Nature.* Berlin:Springer. pp. 124-128.

Kozan, E. and P. Preston (2001). An approach to determine storage locations of containers at seaport terminals. *Computers & Operations Research 28*(10):983-995.

Kumar, R. and N. Banerjee (2003). Multicriteria network design using evolutionary algorithm. In *Lecture Notes in Computer Science 2193*. Springer. pp. 343-352.

Kwok, S., and C. Carter (1990). Multiple decision trees. In *Proceedings on the Fourth Annual Conference on Uncertainty in Artificial Intelligence*, July 1990 MIT, Cambridge, MA . pp. 327-325.

Laumanns, M., L. Thiele, K. Deb, and E. Zitzler (2002). Combining convergence and diversity in multiobjective optimization. *Evolutionary Computation 10*(3):263-282.

Loh, W. and N. Vanichsetakul (1988). Tree structured classification via generalized discriminant analysis. *Journal of the American Statistical Association 83*(403):715-725.

Loh, W. and Y. Shih (1997). Split selection methods for classification trees. *Statistica Sinica 7*(4): 815-840.

Lu, H. and G. G. Yen (2003). Rank-density-based genetic algorithm and benchmark test function study. *IEEE Transactions on Evolutionary Computation 7*(4):325-343.

Lutsko, J.F. and B. Kuijpers (1994). Simulated annealing in the construction of near-optimal decision trees. In P. Chessman and R. W. Oldford Eds., *Selecting Models from Data: AI and Statistics IV*. Springer. pp. 453-462.

Martin, J. and D. Hirschberg (1995). *The time complexity of decision tree induction*. Irvine (CA):Department of Information and Computer Science University of California. Technical Report ICS-TR-95-27.

Martins, E. and J. Santos (1999). *The labeling algorithm for the multiobjective shortest path problem*. Portugal: Departamento de Matematica, Universidade de Coimbra. Technical report nr TR-99/005.

Menzies, T. and Y. Hu (2003). Data mining for very busy people. *IEEE Computer 36*(11):22-29.

Miettinen, K. (1998). *Nonlinear Multiobjective Optimization*. New York: Springer.

Miller, B. and D. Goldberg (1996). Genetic algorithm selections schemes, and the varying effects of noise. *Evolutionary Computation 4*(2):113-131.

Mooney, P. and A. Winstanley (2006).An evolutionary algorithm for multicriteria path optimization problems. *International Journal of Geographical Information Science 20*(4):401-423.

Moret, B. (1982). Decision tree and diagrams. *Computing Surveys 14*(4):593-623.

Morgan, J. and R. Messenger (1973). *THAID: A sequential analysis program for the analysis of nominal scale dependent variables*. Ann Arbor: Institute of Social Research, University of Michigan.

Mühlenbein, H. and D. Schlierkamp-Voosen (1993). Predictive models for the breeder genetic algorithm: continuous parameter optimization. *Evolutionary Computation 1*(1): 25-49.

Müller-Hannemann M., and K. Weihe (2001). Pareto shortest paths is often feasible in practice. In *Lecture Notes in Computer Science 2141*, pp. 185-198.

Murthy S., S. Kasif, and S. Salzberg (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research 2*:1-32.

Murthy, S. (1998). Automatic construction of decision trees from data: a multidisciplinary survey. *Data Mining and Knowledge Discovery 2*(4):345-398.

Murthy, S. and S. Salzberg (1995). Look-ahead and pathology in decision tree induction. In *International Joint Conference on Artificial Intelligence*. August 1995, Montreal, Canada. pp.1025-1033.

Murty, K. G., J. Liu, Y. Wan, and R. Linn (2005). A decision support system for operations in a container terminal. *Decision Support Systems 39*(3):309-332.

Pangilinan J.M. and G.K. Janssens (2009), A genetic algorithm for the storage location of containers at a seaport terminal. In *Proceedings of ASIMMOD 2009*, Bangkok, Thailand. pp. 179-184.

Pangilinan, J.M. A. and G.K. Janssens (2007a). Evolutionary algorithms for the multiobjective shortest path problem. *International Journal of Applied Science, Engineering and Technology 4*(1):205-210.

Pangilinan J.M. and G.K. Janssens (2007b). Multiobjective Optimization and Evolutionary Algorithms. *Saint Louis University Research Journal 38*(1):87-117.

Pangilinan, J.M., G.K. Janssens and A. Caris (2005). A multiobjective evolutionary algorithm for finding efficient solutions to a competitive facility location problem. In *Proceedings of the BIVEC-GIBET Transport Research Day 2005*, Diepenbeek, Belgium. pp. 359-372.

Pangilinan, J.M., G.K. Janssens and A. Caris (2008). Sensitivity analysis of a genetic algorithm for a competitive facility location problem. In K. Elleithy ed., *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering,* Springer-Verlag, Berlin, pp. 266-271.

Papagelis A. and D. Kalles (2001). Breeding decision trees using evolutionary techniques, In *Proceedings of the 18th International Conference on Machine Learning*. Williamstown, MA, USA. pp. 393-400.

Plastria, F. (2001). Static competitive facility location: an overview of optimisation approaches. *European Journal of Operations Research 129*(3):461-470.

Plastria, F. and E. Carrizosa (2004). Optimal location and design of a competitive facility. *Mathematical Programming 100*(2):247-265.

Quinlan R. (1986). Induction of Decision Trees. *Machine Learning 1*:81-106.

Qureshi A. (1997). GPsys software, http://www.cs.ucl.ac.uk/external/A.Qureshi/gpsys.html

Rechenberg, I. (1973). *Evolutionsstrategie-Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Ph.D. Thesis, Stuttgart, Germany:Frommann-Holzboog.

Reeves, C.R. (1993). *Modern Heuristic Techniques for Combinatorial Problems*. New York: Wiley.

Reeves, C.R. and J.E. Rowe (2003). *Genetic Algorithms – Principles and Perspectives A Guide to GA theory*. London:Kluwer.

Richardson, J., M. Palmer M, G. Liepins G, and M. Hilliard (1989). Some guidelines for genetic algorithms with penalty functions. In *Proceedings of the third international conference on genetic algorithms*, June 1989, San Mateo, CA. pp. 191–197.

Saltelli A., K. Chan, and M. Scott (2000). *Sensitivity Analysis*. Probability and Statistics Series. New York, NY :Wiley.

Saltelli, A. (2002a). Sensitivity analysis for importance assessment. *Risk Analysis 22*(3): 1-12.

Saltelli, A. T. H. Andres, and T. Homma (1993). Sensitivity of model output: an investigation of new techniques. *Computational Statistics and Data Analysis 2*(15):211-239.

Saltelli, A., M. Saisana, and S. Tarantola (2005). Uncertainty and sensitivity analysis techniques as tools for the quality assessment of composite indicators. *Journal of the Royal Statistical Society –A 168(*2):1-17.

Saltelli, A., S. Tarantola, F. Campolongo, and M. Ratto (2004). *Sensitivity Analysis in Practice, a Guide to Assessing Scientific models.* Chichester, UK:Wiley.

Schaffer, J. D. (1984*). Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*. Ph. D. Thesis, Nashville, TN: Vanderbilt University.

Schott, J. (1995). *Fault tolerant design using single and multicriteria genetic algorithm optimization*. Master thesis, Cambridge, MA: Massachusetts Institute of Technology.

Serafini, P. (1994). Simulated annealing for multiple objective optimization problems. In *Proceedings of the 10th International Conference on Multiple Criteria Decision Making*. Berlin: Springer-Verlag. pp. 283–294.

Shlien S. (1992). Nonparametric classification using matched binary decision trees. *Pattern Recognition Letters 13*(2):83-88.

SIMLAB 2.2 (2004). *Reference Manual*. European Commission-ISPC.

Sörensen K. and G.K. Janssens (2003). Data mining with genetic algorithms on binary trees. *European Journal of Operations Research 151*(2):253-264.

Srinivas, N. and K. Deb (1994). Multiobjective optimization using non-dominated sorting in genetic algorithms. *Evolutionary Computation 2*(3):221-248.

Steenken, D., S. Voss, and R. Stahlbock (2004). Container terminal operation and operations research – a classification and literature review. *OR Spectrum 26*(1):3-49.

Street, N. (2005). Oblique Multicategory decision trees using nonlinear programming. *Journal of Computing 17*(1):25-31.

Surry, P.D. and N.J. Radcliffe (1996). Inoculation to initialize evolutionary search. In *Evolutionary Computing: AISB Workshop*, pp. 269-265.

Tan, K. C., T. H. Lee, and E. F. Khor (2002). Evolutionary algorithms for multiobjective optimization: performance assessments and comparisons. *Artificial Intelligence Review 17*(4):253-290.

Tsaggouris, G. and C. Zaroliagis (2006). *Multiobjective optimization: improved FPTAS for shortest paths and non-linear objectives with applications*. Greece: Computer Technology Institute, University of Patras. Technical Report nr TR 2006/03/01.

Ulungu, E., J. Teghem, P. Fortemps, and D. Tuyttens (1999). MOSA method: A tool for solving multiobjective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis 8*(4):221–236.

Vadera, S. (2005). Inducing safer oblique trees without costs. *Expert Systems* 22(4):206-221.

Visual Decision (2003). *Decision Lab and Decision Lab 2000*. Montreal, Canada: Visual Decision Inc.

Wang, Q. and C. Suen (1984). Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition*. IEEE Transactions on Pattern Analysis and Machine Intelligence 6*(4):406-417.

Warburton, A. (1987). Approximation of Pareto-optima in multiple-objective shortest path problems. *Operations Research 35*(1):70-79.

Yen, G. G. and H. Lu (2003). Rank-density-based genetic algorithm and benchmark test function study. *IEEE Transactions on Evolutionary Computation 7*(3):253-274.

Zhang, C., J. Liu, Y. Wan, K. Murty, and R. Linn (2003). Storage space allocation in container terminals. *Transportation Research Part B 37*(10):883-903

Zhao, H. (2007). A multiobjective genetic programming approach to developing Pareto-optimal decision trees. *Decision Support Systems 43*:809-806.

Zitzler, E. and L. Thiele (1999). Multiobjective evolutionary algorithms: a comparative study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation 3*(4):257-271.

Zitzler, E. and L. Thiele (2000). Comparison of multiobjective evolutionary algorithms: empirical results. *Evolutionary Computation 8*(2):125-148.

Zitzler, E., M Laumanns, and L. Thiele (2002). SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimization and Control*, pp. 19-26.

Appendix A

The Sobol' method and SIMLAB

Sensitivity analysis (SA) is the study of how the variation in an observed response can be apportioned to different possible sources or factors. Variation in SA, is the randomness in a given dataset. It tries to determine how variable the output *Y* is to changes in the inputs. Sensitivity analysis is often performed using regression techniques. The regression coefficient for a given factor plays the role of a sensitivity measure for that factor. One advantage of regression methods is that they explore the entire interval of definition of each factor. Another is that each effect for a factor is in fact an average over the possible values of the other factors. Moreover *standardized regression coefficients* (SRC) also give the sign of the effect of an input factor on the output providing a simplified model of the input-output mapping. Methods of this type are called *global* to distinguish them from *local* methods wherein only one point of the factor space is explored, and factors are changed one at a time. A disadvantage of regression-based methods is that their performance is poor for non-linear models and can be misleading for non-monotonic models (Saltelli et al 2004).

Model-free sensitivity analysis

Many techniques have been developed that can be considered as a *model-free* extension of the regression methods as they can be applied to non-linear, non-monotonic models. This subsection presents a general introduction to variance-

based techniques, with cases of non-correlated input variables and of correlated input variables.

Consider a deterministic model represented by $Y = f(\boldsymbol{X})$ where $\boldsymbol{X} = (X_1, X_2, \ldots, X_d)$ is a vector of d input variables and Y is the model output. Methods of sensitivity analysis can be defined in terms of the decomposition of the function $Y=f(\boldsymbol{X})$ into main effects and interactions. A high dimensional model representation (HDMR) of $f(\boldsymbol{X})$ can be written as,

$$f(\boldsymbol{X}) = Const + \sum_i f_i(X_i) + \sum_{i<j} f_{ij}(X_i, X_j) + \sum_{i<j<k} f_{ijk}(X_i, X_j, X_k) + \ldots \qquad \text{(A.1)}$$

This procedure is called *decomposition of* $f(\boldsymbol{X})$ into terms of increasing dimensionality, wherein each first order term is a function of a single input variable, each second order term is a function of two variables, and so on. This decomposition is not unique as the lower order terms can be selected arbitrarily and the highest order term can be written as the difference between $f(\boldsymbol{X})$ and the terms of lower order.

If each term in the HDMR is chosen with a zero average, then all pairs of terms in the HDMR are orthogonal to each other and the HDMR decomposition is unique. Being unique, each term can be defined by the conditional averages of $f(\boldsymbol{X})$.

$$f_i(X_i) = E(Y|X_i) - E(Y)$$
$$f_{ij}(X_i, X_j) = E(Y|X_i, X_j) - f_i(X_i) - f_j(X_j)$$
$$f_{ijk}(X_i, X_j, X_k) = E(Y|X_i, X_j, X_k) - f_{ij}(X_i, X_j) - f_{ik}(X_i, X_k) - f_{jk}(X_j, X_k)$$

The $f_i(X_i)$ are referred to as main effects of X_i, the $f_{ij}(X_i, X_j)$ are two-way interactions between the pairs (X_i, X_j), etc.

One measure of sensitivity of Y to an individual input variable X_i that is often used is $V[E(Y|\ X_i)]$, the expected amount of variance that would be removed from the total output variance when the true value of X_i is known (called as *main effect*). Dividing the main effect by the total unconditional variance, the *first-order sensitivity index* for variable X_i can be obtained and is defined as

$$S_i = \frac{V\left[E\left(Y|X_i\right)\right]}{V\left(Y\right)} \qquad \text{(A.2)}$$

This measure indicates the relative importance of an individual input variable X_i in driving the uncertainty, and directs the effort to reduce uncertainty in the future. This type of measure is used before conducting a calibration experiment on a given input. A high value for the main effect of a given input variable indicates that the variable is a good candidate for being calibrated via observations of the model output. The term 1- S_i can be interpreted as the minimum value of the expected quadratic loss when approximating $f(\boldsymbol{X})$ with the function $E(Y|X_i)$. If X_i is important, then the approximating function $E(Y|\ X_i)$ explains much of the variance of $f(\boldsymbol{X})$ and S_i is high.

If $f(\boldsymbol{X})$ is approximated by a two variable function $E(Y\ |X_i,\ X_j)$, then the minimum expected quadratic loss is $V(Y)$-$V[E(Y\ |X_i,\ X_j)]$. This corresponds to the maximum value of $V[E(Y\ |X_i,\ X_j)]$, which is the reduction in expected output variance when the true value of the pair $(X_i,\ X_j)$ is known. It may be interpreted as the fraction of the output variance that is removed when the true value of X_i and X_j is known, or the fraction of the output variance that is explained by the function $E(Y\ |X_i,\ X_j)$.

The total effect for the input variable X_i is linked to $E[V(Y|X_{-i})]$, which is the expected amount of output variance that would remain unexplained (residual variance) if only X_i and X_j were left free to vary over its uncertainty range and all the other variables are known. The term X_{-i} represents all the input variables except X_i. Dividing the total effect by the total unconditional variance, the *total sensitivity index* (TSI) for variable X_i can be obtained and is defined as

$$S_{Ti} = \frac{E\left[V\left(Y|X_{-i}\right)\right]}{V\left(Y\right)} \qquad \text{(A.3)}$$

The total sensitivity indices are used in model building to identify unessential variables - those that are unimportant either singularly or in combination with others. All the input variables having a low total index can be frozen to any value within their range of uncertainty. Total sensitivity indices should be employed to reduce the number of uncertain model inputs.

When the input variables are mutually orthogonal, independent, or non-correlated then it is possible to decompose (or partition) the variance of $f(\boldsymbol{X})$ into terms of increasing dimensionality

$$V(Y) = \sum_i V_i + \sum_{i<j} V_{ij} + \sum_{i<j<k} V_{ijk} + \ldots + \sum V_{1,2,\ldots d} \qquad \text{(A.4)}$$

This decomposition is called ANOVA-HDMR and it is unique. The single terms V_i, V_{ij}, V_{ijk},…, are called partial variances and they are orthogonal. No covariances are involved in the decomposition. The single terms V_i, V_{ij}, V_{ijk},…, can be computed by suitably integrating the corresponding terms in the decomposition.

$$V_i = V\left[E\left(Y \middle| X_i\right)\right]$$
$$V_{ij} = V\left[E\left(Y \middle| X_i, X_j\right)\right] - V_i - V_j$$
$$V_{ijk} = V\left[E\left(Y \middle| X_i, X_j, X_k\right)\right] - V_{ij} - V_{ik} - V_{jk}$$

Hence:

$V_i / V(Y) =$ S_i are the first order sensitivity indices (also called main effects).

$V_{ij} / V(Y) =$ S_{ij} are the second order sensitivity indices (also called two-way interactions)

$V_{ijk} / V(Y) =$ S_{ijk} are the third order sensitivity indices (also called three-way interactions)

The total sensitivity index S_{Ti} for a given X_i in the orthogonal case is the sum of all sensitivity indices that include the input variable X_i.

There are several variance-based techniques for sensitivity analysis and one such method is by Sobol'(Chan et al. 1997). The Sobol' method (Saltelli et al 2000) is a variance-based global sensitivity analysis method based upon total sensitivity indices that take into account interaction effects. The TSI of an input is defined as the sum of all the sensitivity indices involving that input. The TSI includes both the main effect as well as interaction effects (Homma and Saltelli, 1996). For example, if there are three inputs X_1, X_2 and X_3, the TSI of input X_1 is given by S(X_1) + S(X_1 X_2) + S(X_1 X_2 X_3), where S(X_i) is the sensitivity index of X_i. S(X_1) refers to the main effect of X_1. S(X_1 X_2) refers to the interaction effect

between X_1 and X_2. S(X_1 X_2 X_3) refers to the interaction effect between X_1, X_2, and X_3. Effort has been made to reduce the computational complexity associated with the calculation of Sobol' indices (Saltelli 2002a). The method of Sobol' in its improved version uses quasi-random sampling of the input factors The pair (S_i, S_{Ti}) give a fairly good description of the model sensitivities at a reasonable cost, which for the improved Sobol' method is of $2n(k+1)$ model evaluations, where n represents the sample size that is required to approximate the multidimensional integration to a plain sum. n can vary in the 100–1000 range (Saltelli et al., 2005).

The Sobol' method is used in the study because of the following features:

(1) It can cope with both nonlinear and non-monotonic models, and provide a truly quantitative ranking of inputs and not just a relative qualitative measure.
(2) The types of influence of an input that are captured by Sobol' method include additive, nonlinear or with interactions.
(3) The Sobol' method can be smoothly applied to categorical variables without re-scaling.
(4) It can explore the whole range of variation in the input factors instead of just sampling factors over a limited number of values.

SIMLAB

SIMLAB (2004) is a program designed for global uncertainty and sensitivity analysis based on Monte Carlo methods. It offers several techniques for sample generation, sensitivity analysis, and a link to external model execution. The link allows execution of complex models that can hardly be coded as simple mathematical functions such as genetic algorithms (see Figure A.1).

In general, a Monte Carlo sensitivity analysis involves five steps. In the *first* step, a range and distribution are selected for each input variable (input factor). If the analysis is primarily of an exploratory nature, then quite rough distribution assumptions may be adequate. In the *second* step, a sample of points is generated from the distribution of the inputs specified in the first step. The result is a sequence of samples (input sample). In the *third* step, the model is fed with the samples and a set of model outputs is produced. In essence, these model evaluations create a mapping from the input space to the space of the

results. This mapping is the basis for subsequent uncertainty and sensitivity analysis. In the *fourth* step, the results of model evaluations are used as the basis for uncertainty analysis. Uncertainty is characterized statistically by the mean value and the variance. In the *fifth* step, the results of model evaluations are used as the basis for sensitivity analysis.

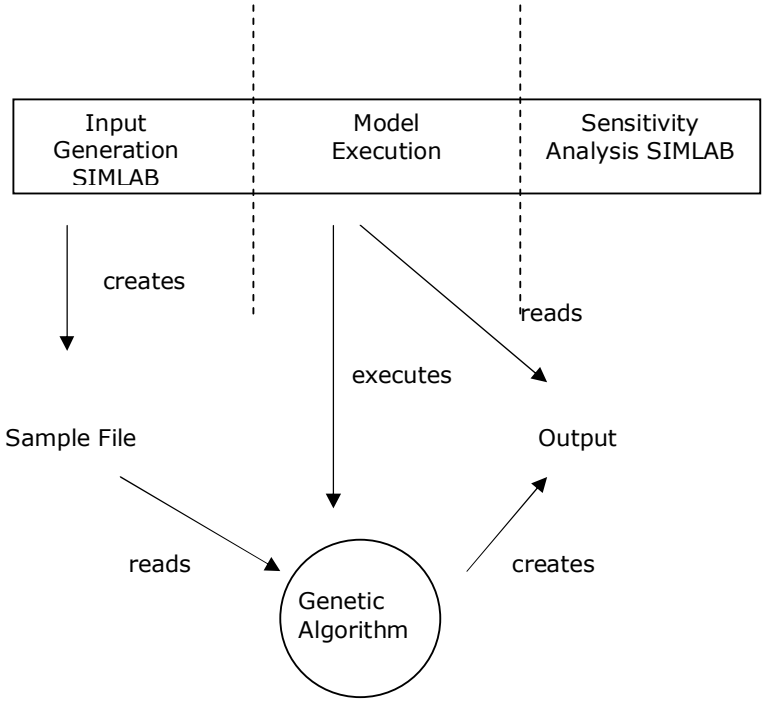

Figure A.1 External model execution.

Appendix B

SPEA2 and NSGA-II

Appendix B presents the complete pseudo code of SPEA2 and NSGA-II to show their differences in terms of finding nondominated solutions and runtime complexity (see Table A.1). The main reference of SPEA in literature is found in Zitzler et al. (2002). The main reference of NSGA-II is found in Deb et al. (2002). Among several reports that evaluate MOEAs as enumerated in Chapter 2, the technical report by Deb et al. (2001) which compares the performance of both algorithms on scalable multiobjective optimization test problems is most significant.

Strength Pareto Evolutionary Algorithm 2 (SPEA2)

Input:	N :	*population size*
	$\overline{N}$:	*archive size*
	T :	*maximum number of generations*
Output:	$\boldsymbol{A}$:	*non-dominated set*

Step 1: *Initialization*.

Initialize($\boldsymbol{P}_0$)	generate an initial population of size N
$\overline{\boldsymbol{P}}_0 = \varnothing$	create the empty archive (external set)
$t = 0$.	

Step 2: *Fitness assignment*:

$$S(\boldsymbol{i}) = \left|\left\{\boldsymbol{j} \mid \boldsymbol{j} \in \overline{\boldsymbol{P}}_t + \overline{\boldsymbol{P}}_{t+1} \wedge \boldsymbol{i} \preceq \boldsymbol{j}\right\}\right| \quad \text{calculate strength of individual } \boldsymbol{i} \qquad \text{(B.1)}$$

$$R(\boldsymbol{i}) = \sum_{\boldsymbol{j} \in \boldsymbol{P}_t + \boldsymbol{P}_t,\, \boldsymbol{j} \succ \boldsymbol{i}} S(\boldsymbol{j})$$ calculate raw fitness of individual (B.2)

$$D(\boldsymbol{i}) = \frac{1}{\sigma_i^k + 2};$$ calculate density of individual $\boldsymbol{i}$ (B.3)

$k = \sqrt{N + N}$ is a density parameter

σ_i^k is the distance between solution $\boldsymbol{i}$ and its k^{th} nearest neighbor

$F(\boldsymbol{i}) = R(\boldsymbol{i}) + D(\boldsymbol{i})$ calculate fitness of individual $\boldsymbol{i}$ (B.4)

Step 3: *Environmental Selection.*

$\overline{\boldsymbol{P}}_{t+1} = \boldsymbol{P}_t + \overline{\boldsymbol{P}}_t$ copy all non-dominated individuals

if ($|\overline{\boldsymbol{P}}_{t+1}| > \overline{N}$)then

reduce($\overline{\boldsymbol{P}}_{t+1}$) use truncation operator to reduce the size of $\overline{\boldsymbol{P}}_{t+1}$ to $\overline{N}$,

else if ($|\overline{\boldsymbol{P}}_{t+1}| < \overline{N}$)

then *fill*($\overline{\boldsymbol{P}}_{t+1}$) add dominated individuals from $\overline{\boldsymbol{P}}_t$ and $\overline{\boldsymbol{P}}_{t+1}$.

Step 4: *Termination*.

if ($t \geq T$) or

(other stopping criterion)

then return($\boldsymbol{A}$) return the nondominated set $\boldsymbol{A}$

Stop.

Step 5:*Mating selection*.

Select($\overline{\boldsymbol{P}}_{t+1}$) perform binary tournament selection with replacement

Step 6: *Variation*.

variate(*mating_pool*) apply recombination the mating pool

mutate(*mating_pool*) apply mutation to the mating pool

$t = t + 1$ increment generation counter

go to Step 2

SPEA2 first assigns a strength value $S(\boldsymbol{i})$, to each individual $\boldsymbol{i}$ from the archive ($\overline{N}$) and population (*N)* representing the number of solutions $\boldsymbol{i}$ dominates. Refer to (B.1) Then the raw fitness $R(\boldsymbol{i})$ of each individual $\boldsymbol{i}$ is calculated which measures the strength of $\boldsymbol{i}$'s dominators. The raw fitness acts as a niching mechanism but poorly performs when most individuals in $N+\overline{N}$ are non-dominated, i.e. the population forms new solutions in only a few clusters, in effect compromising exploration of the search space. Refer to (B.2). This phenomenon is called genetic drift. SPEA2 introduces a density estimator (B.3), a fitness sharing mechanism to avoid genetic drift. The density estimator is defined as the inverse of the distance of an individual in objective space to the *k*-th nearest neighbor. The density value is then added to the raw fitness value to give the final fitness function (B.4). The computing run-time of the fitness function is governed by the density estimator which is $O(N^2 \log N)$.

SPEA2 offers two selection procedures, environmental and mating selection. The environmental selection is concerned with choosing individuals that will have to move on to the next generation archive $\overline{\boldsymbol{P}}_{t+1}$ from the current archive $\overline{\boldsymbol{P}}_t$ and population $\boldsymbol{P_t}$. SPEA2 maintains an archive $\overline{\boldsymbol{P}}_t$ in each generation and is composed of the "best" individuals of a fixed size $\overline{N}$ which is equal to the population size N. Two usual situations may occur in selection. First, the number of non-dominated solutions in $\overline{\boldsymbol{P}}_{t+1}$ is less than $\overline{N}$. SPEA2 resolves this by adding the "best" dominated individuals from $\overline{\boldsymbol{P}}_t + \boldsymbol{P_t}$ to $\overline{\boldsymbol{P}}_{t+1}$. Second, the number of non-dominated solutions for the next generation is greater than $\overline{N}$. SPEA2 uses a truncation procedure whereby the individual with the minimum distance to another individual is truncated until $|\overline{\boldsymbol{P}}_{t+1}| = \overline{N}$. SPEA2 implements binary tournament selection with replacement to fill in the mating pool. This type of mating selects two solutions at a time in each tournament. Their fitness values are evaluated and the better solution is placed in the mating pool. The truncation operator dominates the runtime complexity of the selection procedure and takes $O(N^2 \log N)$on the average and $O(N^3)$ on the worst case.

Nondominated Sorting Genetic Algorithm II (NSGA-II)

Input:	*N* :	*population size*
	T :	*maximum number of generations*
Output:	***A*** :	*non-dominated set*

Step 1: *Initialization*.

Initialize($\boldsymbol{P}_0$)	Generate a population of size N
$t = 0$.	
$\boldsymbol{Q}_0$ = variate($\boldsymbol{P}_0$)	recombine and mutate $\boldsymbol{P}_0$ and create offspring population $\boldsymbol{Q}_0$
Sort($\boldsymbol{F}_k$, $\prec$)	

Step 2: Termination

if ($t \geq T$) or	
(other stopping criterion)	
Then $\boldsymbol{A} = \boldsymbol{P}_t$	
return($\boldsymbol{A}$)	return the nondominated set $\boldsymbol{A}$
Stop.	

Step 3: *Environmental Selection*

$\boldsymbol{R}_t = \boldsymbol{P}_t \cup \boldsymbol{Q}_t$	combine parent and offspring population
$\boldsymbol{F}$ = *fast-nondominated-sort*($\boldsymbol{R}_t$)	create all nondominated fronts of $\boldsymbol{R}_t$
$\boldsymbol{P}_{t+1} = \emptyset$ and $k = 1$	
while ($\lvert\boldsymbol{P}_{t+1}\rvert + \lvert\boldsymbol{F}_k\rvert$) $\leq N$	
crowding-distance($\boldsymbol{F}_k$)	calculate crowding distance
$\boldsymbol{P}_{t+1} = \boldsymbol{P}_{t+1} \cup \boldsymbol{F}_k$	add members of $\boldsymbol{F}_k$ to new parent population
$k = k + 1$	
Sort($\boldsymbol{F}_k$, $\prec$)	sort in descending order using $\prec$
$\boldsymbol{P}_{t+1} = \boldsymbol{P}_{t+1} \cup \boldsymbol{F}_k[1: (N - \lvert\boldsymbol{P}_{t+1}\rvert)]$	choose first ($N - \lvert\boldsymbol{P}_{t+1}\rvert$) of $\boldsymbol{F}_k$

Step 4: *Mating Selection and Variation*

$\boldsymbol{Q}_{t+1}$ = *create*($\boldsymbol{P}_{t+1}$)	apply selection, crossover, and mutation to create new offspring population
$t = t + 1$	increment generation counter
goto Step 2	

In NSGA-II, an initial random parent population $\boldsymbol{P}_0$ is created. The population is sorted based on nondomination. Each solution is assigned a fitness (or rank) equal to its nondomination level (1 is the best level, 2 is the next-best level, and so on). Thus, minimization of fitness is assumed. At first, the usual binary

tournament selection, recombination, and mutation operators are used to create an offspring population $\boldsymbol{Q}_0$ of size N. The step-by-step procedure shows that NSGA-II algorithm is simple and straightforward. First, a combined population $\boldsymbol{R}_t = \boldsymbol{P}_t \cup \boldsymbol{Q}_t$ of size $2N$ is formed and sorted according to nondomination. Since all previous and current population members are included in $\boldsymbol{R}_t$, elitism is ensured. The solutions belonging to the best nondominated set $\boldsymbol{F}_1$ are the best solutions in the combined population and must be emphasized more than any other solution in the combined population. If the size of $\boldsymbol{F}_1$ is smaller than N, all members of the $\boldsymbol{F}_1$ are chosen for the new population $\boldsymbol{P}_{t+1}$. The remaining slots of the next population $\boldsymbol{P}_{t+1}$ are chosen from subsequent nondominated fronts ($\boldsymbol{F}_2$, $\boldsymbol{F}_3$, ..., $\boldsymbol{F}_k$) in the order of their ranking. This procedure is continued until no more sets can be accommodated. In general, the count of solutions in all sets from $\boldsymbol{F}_1$ to $\boldsymbol{F}_k$ would be larger than the population size. The solutions of the last front are sorted using the crowded-comparison operator in descending order to fill N population members. The new population $\boldsymbol{P}_{t+1}$ is subjected to selection, crossover, and mutation to create a new offspring population $\boldsymbol{Q}_{t+1}$ of size N. The algorithm uses a binary tournament selection operator but the selection criterion is based on the crowded-comparison operator $\prec$. Since this operator requires both the rank and crowded distance of each solution in the population, these quantities are calculated while forming the new parent population $\boldsymbol{P}_{t+1}$.

NSGA-II uses a fast nondominated sorting approach which requires $O(N^2)$ computations. For each solution, NSGA-II calculates two entities: 1) domination count n_i, the number of solutions which dominate the solution $\boldsymbol{i}$, and 2) a set of solutions $\boldsymbol{S}_i$ that the solution $\boldsymbol{i}$ dominates. This requires $O(N^2)$ comparisons.

All solutions in the first nondominated front have their domination count as zero. For each solution $\boldsymbol{i}$ with $n_i = 0$, NSGA-II visits each member $\boldsymbol{j}$ of its set $\boldsymbol{S}_i$ and reduces its domination count by one. If for any member $\boldsymbol{j}$, the domination count becomes zero, $\boldsymbol{j}$ is added in a separate list $\boldsymbol{Q}$. These individuals belong to the second nondominated front. This procedure is continued with each member of $\boldsymbol{Q}$ and the third front is identified. The process continues until all fronts are identified. For each solution in the second or higher level of nondomination, the domination count n_i can be at most $N-1$. Each solution is visited at most $N-1$ times before its domination count becomes zero. At this point, the solution is assigned a nondomination level and is never be visited again. Since there are at most $N-1$ such solutions, the total complexity is $O(N^2)$. Thus, the overall complexity of the procedure is $O(N^2)$. Another way to calculate this complexity is to realize that the body of the first inner loop (for each $\boldsymbol{i} \in \boldsymbol{F}_k$) is executed

exactly N times as each individual can be the member of at most one front and the second inner loop (for each $\boldsymbol{j} \in \boldsymbol{S}_i$) can be executed at maximum N–1 times for each individual results in the overall $O(N^2)$ computations.

Fast-nondominated-sort ($\boldsymbol{P}$)
for each $\boldsymbol{i} \in \boldsymbol{P}$
    $\boldsymbol{S}_i = \varnothing$
    $n_i = 0$
    for each $\boldsymbol{j} \in \boldsymbol{P}$
        if ($\boldsymbol{i} \prec \boldsymbol{j}$) then
            $\boldsymbol{S}_i = \boldsymbol{S}_i \cup \{\boldsymbol{j}\}$ Add $\boldsymbol{j}$ to solutions dominated by $\boldsymbol{i}$
        else if ($\boldsymbol{j} \prec \boldsymbol{i}$) then
            $n_i = n_i + 1$ Increment the domination counter of $\boldsymbol{i}$
    if ($n_i = 0$) then $\boldsymbol{i}$ belong to the first front
        $i_{rank} = 1$
        $\boldsymbol{F}_1 = \boldsymbol{F}_1 \cup \{\boldsymbol{i}\}$

$k = 1$ Initialize the *front* counter
while $\boldsymbol{F}_k \neq \varnothing$
    $\boldsymbol{Q} = \varnothing$ Used to store the members of the next front
    for each $\boldsymbol{i} \in \boldsymbol{F}_k$
        for each $\boldsymbol{j} \in \boldsymbol{S}_i$
            $n_j = n_j - 1$
            if ($n_j = 0$) then $\boldsymbol{j}$ belongs to the next front
                $j_{rank} = k + 1$
                $\boldsymbol{Q} = \boldsymbol{Q} \cup \{\boldsymbol{j}\}$
    $k = k + 1$
    $\boldsymbol{F}_k = \boldsymbol{Q}$

NSGA-II uses a crowded-comparison approach. This approach does not require any user-defined parameter for maintaining diversity among population members. An estimate of the density of solutions surrounding a particular solution in the population is calculated, which is the average distance of two points on either side of a particular solution along each of the objectives. This quantity serves as an estimate of the perimeter of the cuboid formed by using the nearest neighbors as the vertices and is called the crowding distance. The crowding-distance computation requires sorting the population according to each objective function value in ascending order of magnitude. Thereafter, for each

objective function, the boundary solutions (solutions with smallest and largest function values) are assigned an infinite distance value. All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. This calculation is continued with other objective functions. The overall crowding-distance value is calculated as the sum of individual distance values corresponding to each objective. Each objective function is normalized before calculating the crowding distance. The complexity of this procedure is governed by the sorting algorithm which is $O(N\log N)$. After all population members in the nondominated set are assigned a distance metric, two solutions are compared for their proximity with other solutions. A solution with a smaller value of this distance measure means it is more crowded by other solutions.

The crowded-comparison operator ($\prec$) guides the selection process at the various stages of the algorithm toward a uniformly-spread Pareto-optimal front. Assume that every individual in the population has two attributes: nondomination rank ($\boldsymbol{i}_{\text{rank}}$) and crowding distance ($\boldsymbol{i}_{\text{distance}}$). A partial order $\prec$ can be defined as

$$\boldsymbol{i} \prec \boldsymbol{j} \text{ if } (\boldsymbol{i}_{\text{rank}} < \boldsymbol{j}_{\text{rank}}) \text{ or } ((\boldsymbol{i}_{\text{rank}} = \boldsymbol{j}_{\text{rank}}) \text{ and } (\boldsymbol{i}_{\text{distance}} > \boldsymbol{j}_{\text{distance}}))$$

That is, between two solutions with differing nondomination ranks, preference is given to the solution with the lower (better) rank. Otherwise, if both solutions belong to the same front, then preference is given to the solution that is located in a lesser crowded region.

SPEA2	NSGA-II
• Creates a single non-dominated front • Fitness is based on one attribute – a function of dominators and density • $O(N^2 logN)$	• Creates subsequent non-dominated fronts • Fitness is based on two separate attributes. – non-domination rank – crowding distance • $O(N^2)$

Figure B.1 Differences between SPEA2 and NSGA-II

Appendix C

PISA – A Platform and Programming Language Independent Interface for Search Algorithms

Bleuler et al. (2002) proposed a platform and programming language independent interface for search algorithms (PISA) that uses a text file format for data exchange. PISA allows developers to maintain collections of precompiled optimization algorithms and applications which can be arbitrarily combined. Application developers with little knowledge in optimization can easily try different optimization strategies for the problem at hand whereas algorithm developers have the opportunity to test optimization techniques on various applications without the need to program the problem-specific parts. The interface is simple to use, and most existing optimizers and applications can be adapted to the interface specification with several modifications

Control Flow

The model ensures that there is a consistent control flow state for the whole optimization process and that only one module is active at any time. Whenever a module reads a state that requires some action on its part, the operations are performed and the next state is set.

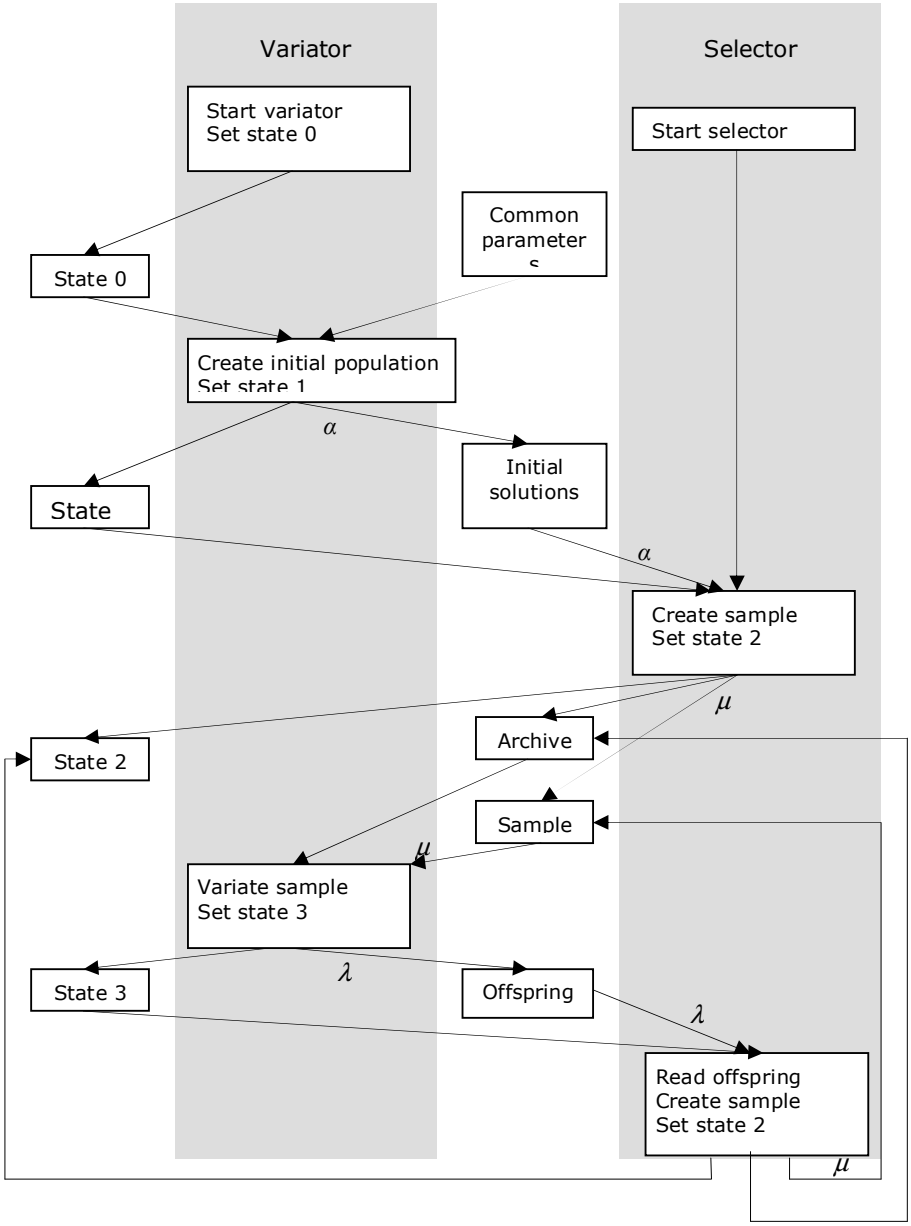

Figure C.1 The control flow and data flow specifications of PISA.

The core of the optimization process consists of state 2 and state 3: In each generation the selector chooses a set of parent individuals (μ) and passes them

to the variator. The variator generates new individuals (λ) on the basis of the parents, computes the objective function values of the new individuals, and passes them back to the selector. In addition to the core states, two more states are shown in Figure C.1. State 0 and state 1 trigger the initialization of the variator and the selector, respectively. In state 0 the variator reads the necessary parameters. Then, the variator creates an initial population (a), calculates the objective values of the individuals and passes the initial population to the selector. In state 1, the selector also reads the required parameters, then selects a sample of parent individuals and passes them to the variator. The abovementioned states provide the basic functionality of the optimization.

Data Flow

The data transfer between both modules introduces some overhead compared to a traditional monolithic implementation. Thus, the amount of data exchange for each individual must be minimized. Since all representation specific operators are located in the variator, the selector does not have to know the representation of the individuals. Therefore, it is sufficient to convey only the following data to the selector for each individual: an index, which identifies the individual in both modules, and one objective vector. In return, the selector only needs to communicate the indices of the parent individuals to the variator. The proposed scheme allows restricting the amount of data exchange between both modules to a minimum. An individual is superior to another in regard to one objective, if the corresponding element of the objective vector is smaller, i.e., objective values are to be minimized. Furthermore, the two modules need to agree on the sizes of the three collections of individuals passed between each other: the initial population, the sample of parent individuals, and the offspring individuals. These sizes are denoted as a, μ and λ in Fig. C.1. Instead of using some kind of automatic coordination, which would increase the overhead for implementing the interface, the sizes are specified as parameter values. Setting μ and λ as parameters requires that they are constant during the optimization run. Most existing algorithms comply with this requirement. A collection of parent individuals is passed from the selector to the variator and a collection of offspring individuals is returned. The actual individuals are stored on the variation side.

Synchronization

In order to reach the necessary separation and compatibility, the selector and the variator are implemented as two separate processes. These two processes can be located on different machines with possibly different operating systems. A common state variable is used for synchronization, which both modules can read and write. Both processes regularly read this state variable and perform the corresponding actions. If no action is required in a certain state, the respective process sleeps for a specified amount of time and then rereads the state variable. The common state variable is implemented as an integer written to a text file. File access is completely portable and familiar to all programmers. The only requirement is access to the same file system. All data exchange is established through text files. Using text files with human readable format allows the user to monitor data exchange easily. A separate file is used for each collection of individuals shown in Fig. C.1. Several parameters are necessary for both modules and each module specifies its own parameter set. However, parameters that are common to both modules are written in a common parameter file. This prevents users from setting different values for the same parameter on the variation and the selection side. The set of common parameters consists of the number of objectives and the sizes of the three different collections of individuals that are passed between the two modules.

The authors of PISA maintain a website at http://www.tik.ethz.ch/~sop/pisa/. The website provides documentation, downloads for variator and selector modules, and PISA beginner modules among others. The source code of the modules is written in the C programming language in a Linux environment but the modules come with binary files that run in Linux, Solaris, and Windows. For Windows users, the original C code must be ported to a C compiler that runs in DOS. Since PISA uses a text file format for data exchange, collections of precompiled optimization algorithms and applications can be arbitrarily combined independent of their platform. However, the data exchange via files increases the execution time but this overhead is small compared to the benefits of PISA.