Multiple Objective Evolutionary Algorithms for Independent, Computationally Expensive Objective Evaluations

A Thesis
Presented to
The Academic Faculty

by

Greg Rohling

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Electrical and Computer Engineering
Georgia Institute of Technology
November 2004

Multiple Objective Evolutionary Algorithms for Independent, Computationally Expensive Objective Evaluations

Approved by:

Dr. James O. Hamblen, Committee Chair	Dr. Mark A. Richards
Dr. Mark A. Clements, Advisor	Dr. Ellis Johnson
Dr. James H. McClellan	Dr. Darrell R. Lamm

Date Approved: November 2nd, 2004@

This thesis is dedicated to my wife and eight children. Without their support this would not have been possible.

ACKNOWLEDGEMENTS

I would like to acknowledge the following who have made this thesis possible:

1. *Dr. Darrell Lamm*, my mentor and friend for the past 15 years, spent countless hours over these years listening and directing my research efforts.

2. *Dr. Mark Clements*, my thesis advisor, had the patience to take on a part-time student for this long road.

3. *Georgia Tech Research Institute*, my employer, provided funding for the qualifying exam, Ph.D research, development of the PRESTO software, and tuition. Without their support I would have never begun this research.

4. *Charles Carstensen*, of GTRI, saw the potential of Genetic Algorithm application to flare pattern design against infra-red surface to are missiles. Fortunately, he saw the potential before the shooting started.

5. *Mark Salyers* of WRALC/LNSES had the foresight to see the potential of the Genetic Algorithms application to the AAR-44A. He pursued the funds within the government to make this application a reality.

6. *Dave Stephens* of WRALC/LNSES pushed the government paperwork to get public release for the AAR-44A chapter.

7. *Rob Ritchie* of USARMY/CECOM funded research into the use of Genetic Algorithms for Flare Pattern Design

8. *Nancy Rohling*, my wife and mother of our eight children, supported this effort from the start, (seven children ago).

9. *Wilfred and Pat Rohling*, my parents, helped hold down the home front. They were always there when we needed them.

10. *Dr. James O. Hamblen*, my committee qualifier and proposal chair, sent me the first article on Genetic Algorithms that started this research.

11. *Dr. James H. McClellan* and *Dr. Mark A. Richards*, members of my qualifying and proposal committees, were readers for this thesis.

12. *Mr. Dan Kline* and *Mrs. Grace Kline*, long time friends, suffered through reading the first version of all chapters and repeatedly corrected my poor use of commas, abrupt changes in tense, plural versus singular problems, and my use of "which" instead of "that".

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ADF	Automatically Defined Functions.
ADS	Automatically Defined Storage.
$\boldsymbol{x}$	Attributes. Decision variables. A set of n attributes for an individual. $x_1, x_2, \cdots, x_n$.
$\boldsymbol{L}$	Region of interest in objective space point. Point used to define hypercube $h_f(\boldsymbol{L})$ that locates the region of interest in objectives space.
BTS	Binary Tournament Selection method.
$\|C\|$	Number of individuals in the current population C.
CE	Computationally Expensive.
CPU	Central Processing Unit.
DARPA	Defense Advanced Research Project Agency.
DISAMS	Digital Infrared Seeker and Missile Simulation.
D_O	Number of individuals to be evaluated between updates of dynamic objective ordering.
D_T	Number of individuals to be evaluated between updates of dynamic objective thresholds.
DTD	Document Type Description.
$\|E\|$	Number of individuals in the elite population E.
EA	Evolutionary Algorithm.
EOC	Extended Operation Conditions.
FA	False Alarm. Also a False Postive.
F_{adf}	Function set for Automatic Defined Function.
FN	False Negative.
FP	False Positive. Also a False Alarm.
FPS	Fitness Proportionate Selection method.
F_{rpb}	Function set for the Result Producing Branch.

FS	Niched Pareto Genetic Algorithm Fitness Sharing method.
GA	Genetic Algorithm.
GP	Genetic Programming.
GTMOEA	Georgia Tech Multiple Objective Evolutionary Algorithm Software.
GTRI	Georgia Tech Research Institute.
HCD	Hypercube Distance. Distance of location in $\mathbb{R}^n$ space to to a n-dimesional hypercube in the same space.
$h_f(\boldsymbol{L})$	Hypercube defining region of interest. Located in front of point $\boldsymbol{L}$.
$h_f(\boldsymbol{S})$	Hypercube defining region of known interests. Located in front of point $\boldsymbol{S}$. Used by dynamic objective thresholding algorithm.
$\|H_O\|$	Number of individuals in historic objective ordering population H_O.
$\|H_T\|$	Number of individuals in historic objective thresholding population H_T.
ICEO	Independent Computational Expensive Objectives.
IIR	Infinite Impluse Response.
ISAMS	Imaging Missile And Seeker Simulation.
JNI	Java Native Interface.
LID	Linear Interpolated Data.
$\|LID\|$	Number of point in Linear Interpolated Data transformation list.
m	Number of constraints on the search space.
$NFAC$	Negative False Alarm Count.
TTI_{Avg}	Time To Intercept Average.
TTI_{Min}	Time To Intercept Minimum.
MOEA	Multiple Objective Evolutionary Algorithm.
MOEP	Multiple Objective Evolutionary Programming.

MoML	Modeling Markup Language. A dialect of XML.
MOP	Multi-objective Optimization Problem.
MR	Maximum Rank.
n	Number of dimensions in the search space.
NPGA	Niched Pareto Genetic Algorithm.
$\boldsymbol{y}$	Objectives. The set of q objective values for an individual solution. $y_1, y_2, \cdots, y_q$.
$\boldsymbol{Y}$	Objective space. The set of all valid objective sets, $\boldsymbol{y}$.
OFP	Operational Flight Program.
$\|P\|$	Number of individuals in the total population P, which includes the current C and elite E population.
P_d	Probability of Detection.
PR1	Pareto Rank Method 1.
PR2	Pareto Rank Method 2.
PRESTO	Pattern Recognitio Evolutionary Synthesis Through Optimization.
$P_{s,BTSNR}$	Probability of selection using binary tournament selection without replacement.
$P_{s,BTSWR}$	Probability of selection using binary tournament selection with replacement.
$P_{s,FPS}$	Probability of selection using fitness proportionate selection.
q	Number of dimensions in the objective space.
ROI	Region of interest in objective space. Defined by $h_f(\boldsymbol{L})$.
s	Number of "important" individuals from the historic ordering population to be compared between the current objective ordering and a new proposed objective ordering.
SAR	Synthetic Aperture Radar.
SD	Strength Pareto Evolutionary Algorithm II Density method.
$\boldsymbol{X}$	Search space. The set of all valid attribute sets, $\boldsymbol{x}$.
SOP	Single Objective Optimization Problem.

SPEA	Strength Pareto Evolutionary Algorithm.
SPEAII	Strength Pareto Evolutionary Algorithm II.
SRF	Storage Reading Function.
SWF	Storage Writing Function.
T_{adf}	Terminal set for Automatic Defined Function.
TN	True Negative.
TP	True Positive.
T_{rpb}	Terminal set for the Result Producing Branch.
TTI	Time To Intercept.
WS	Weighted-Sum.
WSMR	White Sands Missile Range.
XML	eXtended Markup Language.

SUMMARY

This research augments current Multiple Objective Evolutionary Algorithms with methods that dramatically reduce the time required to evolve toward a region of interest in objective space.

Multiple Objective Evolutionary Algorithms (MOEAs) are superior to other optimization techniques when the search space is of high dimension and contains many local minima and maxima. Likewise, MOEAs are most interesting when applied to non-intuitive complex systems. But, these systems are often computationally expensive to calculate. When these systems require independent computations to evaluate each objective, the computational expense grows with each additional objective. This method has developed methods that reduces the time required for evolution by reducing the number of objective evaluations, while still evolving solutions that are Pareto optimal. To date, all other Multiple Objective Evolutionary Algorithms (MOEAs) require the evaluation of all objectives before a fitness value can be assigned to an individual.

The original contributions of this thesis are:

1. Development of a hierarchical search space description that allows association of crossover and mutation settings with elements of the genotypic description.

2. Development of a method for parallel evaluation of individuals that removes the need for delays for synchronization.

3. Dynamical evolution of thresholds for objectives to allow partial evaluation of objectives for individuals.

4. Dynamic objective orderings to minimize the time required for unnecessary objective evaluations.

5. Application of MOEAs to the computationally expensive flare pattern design domain.

6. Application of MOEAs to the optimization of fielded missile warning receiver algorithms.

7. Development of a new method of using MOEAs for automatic design of pattern recognition systems.

CHAPTER I

INTRODUCTION

Automated optimization methods allow exploration of a multi-dimensional search space of possible decision variables looking for a single set of decision variables that produce the desired objective. Many of today's optimization problems do not require optimization for a single objective, but rather require optimization for many often competing objectives. For example, in any industry there is a desire to optimize a system such that it is highly productive, uses minimal power, and is inexpensive. For these multiple objective problems, a suite of system solutions that provides trade-offs in objectives is desired.

Fortunately, computer simulations are available, or are commonly developed, to understand the performance tradeoffs of a proposed solution in the many diverse environments a system may be required to operate. But, computer simulations for today's complex systems require very detailed modeling, which is often computationally expensive. This computational expense results in a substantial quantity of time expended for any optimization technique that uses these simulations. Thus, there is a desire to find the suite of optimal system solutions in a minimal amount of time, implying a minimal use of simulation. Multiple Objective Evolutionary Algorithms (MOEAs) provide an automated method for exploring large search spaces to find optimal solutions, but they require evaluation of many objectives [24]. This research augments MOEAs with methods that reduce the time to explore the search space for independent, computationally expensive objectives by reducing the number of simulations required.

The definitions for all symbols and abbreviations used in this thesis are given upon their first usage. For the convenience of the reader, the definitions for the major symbols and abbreviations are also given in the "List of Symbols and Abbreviations" section located

in the front of this document.

The rest of this chapter is organized as follows: Section 1.1 provides the lexicon for discussion of optimization of single and multiple objective optimization problems. Section 1.2 provides the basics for single and multiple objective evolutionary algorithms. Section 1.3 provides an introduction to Pareto optimal concepts. These sections provide the foundation for the overview of the research contributions made by this research in Section 1.4. The final section of the Introduction, Section 1.5, provides an overview of the remaining chapters.

1.1 Optimization Problems

For single objective optimization problems (SOP), an individual system is defined by a set of n attributes (or decision variables), $\boldsymbol{x} = (x_1, x_2, \cdots, x_n) \in \mathbb{R}^n$. All possible combinations of attributes may not be feasible. A set of m constraints for the attributes can be defined such that each constraint must be greater than 0, $\boldsymbol{e(x)} = (e_1(\boldsymbol{x}), e_2(\boldsymbol{x}), \cdots, e_m(\boldsymbol{x})) \leq \boldsymbol{0}$. The set of all feasible individuals defines the search space, $\boldsymbol{X}$. The single objective is then the performance of the system for this objective, y, where all possible objective values define the objective space, $Y \subset \mathbb{R}$. A single objective optimization problem can then be formulated as:

$$\begin{aligned} \text{Maximize} \quad & y = f(\boldsymbol{x}) \\ \text{Subject to} \quad & \boldsymbol{e(x)} = (e_1(\boldsymbol{x}), e_2(\boldsymbol{x}), \cdots, e_m(\boldsymbol{x})) \leq \boldsymbol{0} \\ \text{where} \quad & \boldsymbol{x} = (x_1, x_2, \cdots, x_n) \in \boldsymbol{X} \subset \mathbb{R}^n \\ & y \in Y \subset \mathbb{R}. \end{aligned}$$

As discussed above, many real world problems desire the optimization of multiple objectives rather than a single objective. These multi-objective optimization problems (MOP)

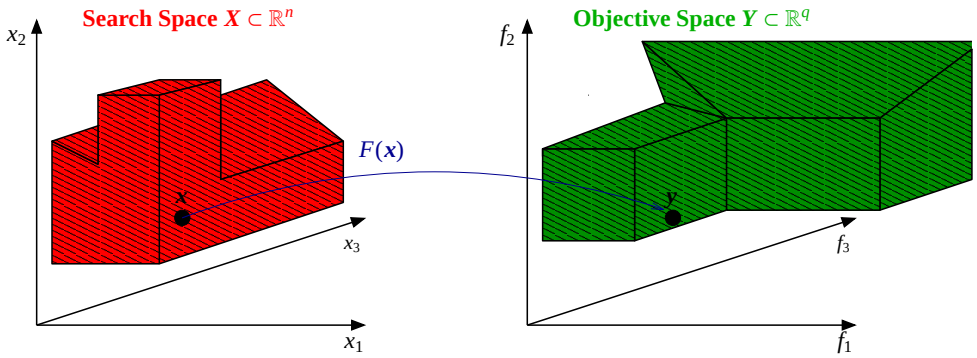

Figure 1: Search Space to Objective Space Mapping

require evaluation of q objectives for each individual system,

$$\boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \cdots, f_q(\boldsymbol{x})) \in \mathbb{R}^q. \tag{1}$$

The set of all possible solutions $\boldsymbol{y}$, defines the multiple objective space $\boldsymbol{Y}$. Thus as illustrated in Figure 1, the fitness function, $\boldsymbol{F}$, provides a mapping for each individual $\boldsymbol{x}$ in the search space $\boldsymbol{X} \subset \mathbb{R}^n$ to a location $\boldsymbol{y}$ in the objective space $\boldsymbol{Y} \subset \mathbb{R}^q$. A multi-objective optimization problem can then be formulated as:

$$\begin{aligned} \text{Maximize} \quad & \boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \cdots, f_q(\boldsymbol{x})) \\ \text{Subject to} \quad & \boldsymbol{e}(\boldsymbol{x}) = (e_1(\boldsymbol{x}), e_2(\boldsymbol{x}), \cdots, e_m(\boldsymbol{x})) \leq \boldsymbol{0} \\ \text{where} \quad & \boldsymbol{x} = (x_1, x_2, \cdots, x_n) \in \boldsymbol{X} \subset \mathbb{R}^n \\ & \boldsymbol{y} \in \boldsymbol{Y} \subset \mathbb{R}^q. \end{aligned}$$

For both multi-objective and single objective optimization problems, minimization problems are formulated similarly and are often recast as maximization problems by changing the sign of the desired objective, y_k.

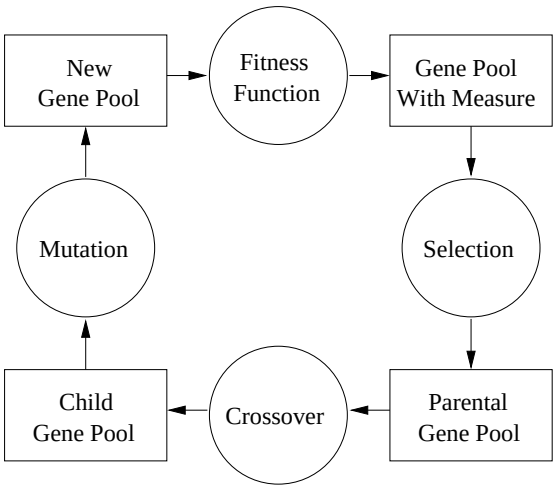

Figure 2: Evolutionary Algorithm Process

1.2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) provide optimization techniques that mimic the three major components of natural evolution and selective breeding: selection, exchange of genetic material during reproduction (or crossover), and random mutations. Figure 2 illustrates this process. Although as shown in the *No Free Lunch Theorems* of Wolpert and Macready [59] there is a danger of comparing algorithms, EAs have shown themselves to be of increasing interest for those problems with large search spaces and complex high frequency functions with many local minima and maxima [57].

The basic concepts for EAs originated in 1859 with Darwin [16]. Over a century later in 1975, Holland [29] is credited as pioneering the application of these evolutionary concepts to optimization problems. By their definition EAs lend themselves to discussion in terms of a biological paradigm. For example, a specific instance of a system is referred to as an individual. An individual contains a genotypic description, which is the list of attributes (or decision variables) to be optimized, $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$. Likewise, an individual's genotypic description is manifested in a phenotypic description, which is the individual's realization in the operating environment, $\boldsymbol{y} = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \cdots, f_q(\boldsymbol{x}))$. A group of individuals make up a population, P. Selection is mimicked by comparing the performances of

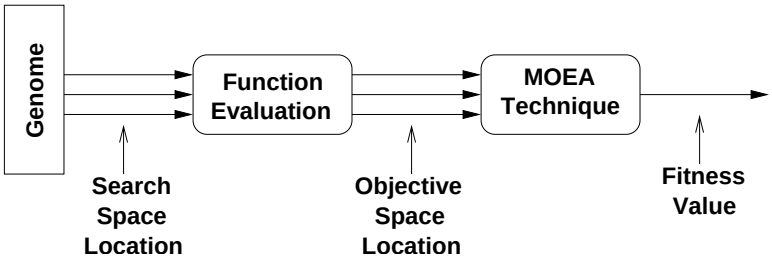

Figure 3: Multiple Objective Fitness

individuals in a population and determining which individuals get to mate. An individual with more desirable fitness is given a higher probability of mating. Mating is performed by taking some attributes from one parent and the remaining attributes from another parent. The attributes of the resulting individual can then be mutated using any method desired, including normal and uniform distributions. Mutation provides a natural resistance to the optimization process converging on a local minimum or maximum and allows introduction of new genetic material into the gene pool.

1.2.1 Multiple Objective Evolutionary Algorithms

As long as a fitness function is available that can associate the performance of any possible individual's genotypic description with its phenotypic description, then the processes of selection, crossover, and mutation can be applied to optimize the performance. But, the process of selection requires the ability to rank all individuals' performance in order to determine which individuals are selected for mating, i.e., selection requires a fitness value. For SOPs, this rank can be made with the single objective value itself, y. But for MOPs, the phenotypic description contains multiple objectives. As illustrated in Figure 3, MOEAs provide methods for mapping individuals from the multiple objective domain to a single fitness value that can be used by the selection process. The details of state of the art MOEA techniques as well as the MOEA contributions of this research are presented in Chapter 2.

1.2.2 Weighted-Sum

The simplest method for mapping from multiple objective space to a single fitness value is the weighted-sum. The weighted-sum of objectives method combines the objective values $f_k(i)$, of an individual i, by multiplying by a weight w_k,

$$f(i) = \sum_{k=1}^{q} w_k f_k(i). \tag{2}$$

The weighted-sum coefficients, w_k, define a vector in objective space. Thus, individuals that have the best fitness are those that are farthest from the hyperplane that is perpendicular to the defined vector. A simple weighted-sum technique is adequate for many problems but is less appealing for others. Three limitations for optimization using weighted-sum techniques are:

1. A simple weighted-sum technique is difficult for problems where the relationships between the objectives are not well understood; i.e., problems where it is hard to determine a valid set of weights w_k.

2. Since the fitness measure finds those solutions furthest from the hyperplane perpendicular to the vector defined by the weights, a weighted-sum technique can result in problems finding solutions in concave portions of the optimal surface, as illustrated in Figure 4.

3. A simple weighted-sum technique only finds a single solution of the many possible optimal solutions in the objective space. Thus, the single solution does not provide the ability to understand the various trade-offs in objective space that are possible.

Despite these limitations, a linear combination of objectives is commonly used. In fact, as reported by Van Veldhuizen [57], weighted-sums are the most common method found in open literature for determination of a fitness value from multiple objectives.

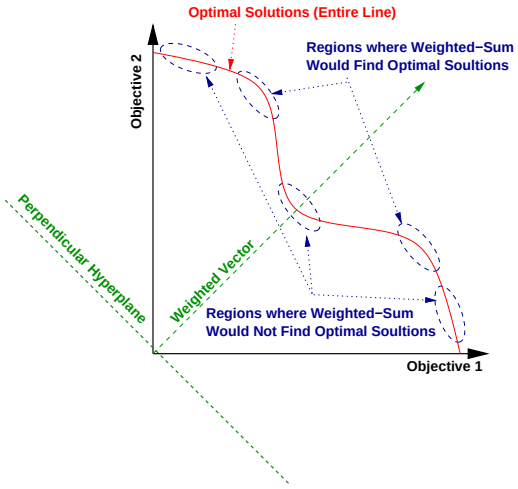

Figure 4: Weighted-Sum Optimization with Concave Surfaces

1.3 Pareto Optimality

One of the most desirable products of MOEA techniques is the evolution of a set of solutions that provide the trade-offs in objective space. With single objective optimization an individual i is better than another solution j if $f(i) > f(j)$. For multiple objectives this becomes slightly more complex and benefits from the definition of Pareto optimality. Assuming the purpose of the optimization is to maximize all objectives, an individual i *dominates* individual j, $(i \succ j)$, if and only if for all $k = 1$ to q, $f_k(i) \geq f_k(j)$ and $f_k(i) > f_k(j)$ for at least one k. An individual i *weakly dominates* individual j, $(i \succeq j)$, if and only if for all $k = 1$ to q, $f_k(i) \geq f_k(j)$. Weak domination allows individuals to have identical objectives values. Defining P as a set of individuals, an individual $i \in P$ is Pareto optimal if and only if there exists no $j \in P$ such that $j \succ i$. Simply put, an individual is Pareto optimal if it is not outperformed in every dimension by any single individual. The set of Pareto optimal solutions, E, is a subset of the original population, $E \in P$. Figure 5 gives an example of Pareto optimal solutions in two dimensions. An individual that is not Pareto optimal is said to be *dominated* or *inferior*. Pareto optimal solutions are also referred to as *non-dominated*. The resulting Pareto optimal solutions create a Pareto front. Any new solution residing behind the Pareto front will not be Pareto optimal. Any new solution residing in front of the Pareto

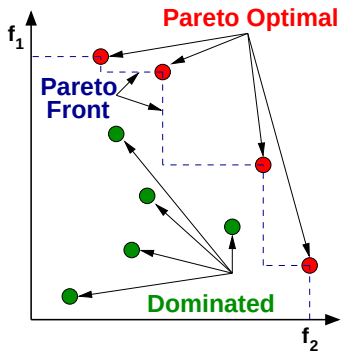

Figure 5: Pareto Optimal Example

front will be Pareto optimal. The results of a MOP do not result in a single individual as in SOP, but rather a population, E, of Pareto optimal solutions.

1.3.1 Hypercubes

Each individual in objective space $\mathbb{R}^q$ defines q planes each of dimension $q-1$. These planes define two hypercubes in the $\mathbb{R}^q$ space. First, there is a hypercube in front of the point a such that every point b within the hypercube contains values such that for all $k = 1$ to q, $a_k > b_k$. Likewise, there is a hypercube located behind each point a such that every point b within the hypercube contains values such that for all $k = 1$ to q, $a_k < b_k$. Given an individual i and the location in objective space $f(i)$, define the hypercube behind the individual i in objective space $f(i)$ as $h_b(f(i))$. Likewise, define the hypercube in front of the individual i in objective space as $h_f(f(i))$. Figure 6 illustrates these two hypercubes for a three-dimensional case.

The definition of these hypercubes allows us to discuss Pareto optimality in terms of these hypercubes. An individual i is Pareto optimal only if there are no individuals that reside in the hypercube in front of it, $h_f(f(i))$. Likewise, an individual is dominated by all individuals whose hypercubes behind them contain the individual.

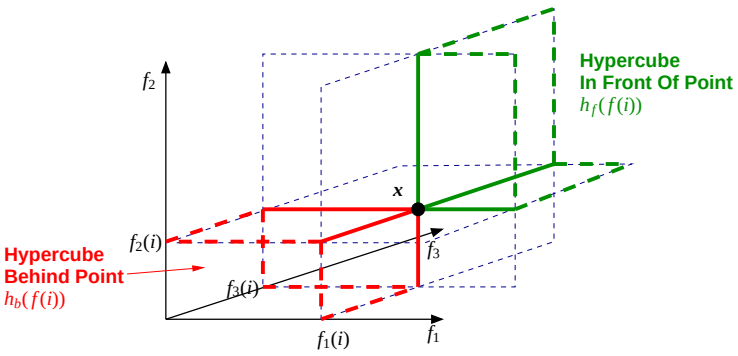

Figure 6: Hypercube Example

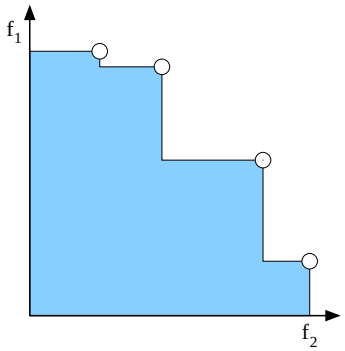

Figure 7: S Hypervolume of Pareto Optimal Solutions

1.3.2 Hypervolume

Hypercubes also lead naturally into a discussion of measure for MOEA performance developed by Zitzler [61] called the hypervolume. The Pareto front encloses a volume of dimension q in objective space that is the union of the hypercubes behind each Pareto optimal individual. Given a population of Pareto optimal individuals, E, then the volume enclosed by the population $S(E)$ becomes a measure for the Pareto optimal front,

$$S(E) = v\left(\bigcup_{i \in E} h_b(f(i))\right) \tag{3}$$

as illustrated in Figure 7. This implies that the origin of the objective spaces is at $(0, \cdots, 0)$. If this is not desired, the origin can easily be translated.

Throughout the evolutionary process, there are sets of solutions that for an instance in

the optimization process are Pareto optimal. Measuring the hypervolume over the optimization process provides a measure of the improvement being made. If no new Pareto optimal solutions are found, then the hypervolume will remain constant. If the hypervolume is increasing then new areas of the objective space in front of the previous Pareto front are being explored.

Hypervolume also allows comparison of populations that are the results of multiple optimizations. In general, resulting populations with large hypervolumes are exploring more of the objective space. But, the hypervolume can not tell the entire story. For example, as illustrated in Figure 8, two populations can contain identical hypervolumes and still be exploring different regions of the objective space.

1.3.3 Coverage Measure of Two Sets

An additional measure for comparison of two populations of solutions, also developed by Zitzler [61], is the coverage measure. The coverage measure can be used with the hypervolume measure for additional information. Consider two sets of solutions $A, B \in Y$. The coverage function C then determines the percentage of solutions in B that are weakly dominate the solutions in A Let $|B|$ indicate the number of individuals in population B. Then,

$$C(A,B) = \frac{|\{b \in B | \exists a \in A : a \succeq b\}|}{|B|}. \tag{4}$$

The value $C(A,B) = 1$ implies that all decision vectors in B are weakly dominated by A. Conversely, a value of $C(A,B) = 0$ implies that none of the objective vectors in B are weakly dominated by A. Note that because sets A and B may cover different regions of objective space, $C(B,A)$ does not necessarily equal $1 - C(A,B)$. Therefore as illustrated in figure 8, both $C(A,B)$ and $C(B,A)$ must be considered.

The cases for relationships between $S(A)$, $S(B)$, $C(A,B)$, and $C(B,A)$ are as follows:

1. *A dominates B.* If $C(A,B) = 1$ and $C(B,A) = 0$ then set A dominates set B, and thus $S(A) > S(B)$, as illustrated on the left side of Figure 8.

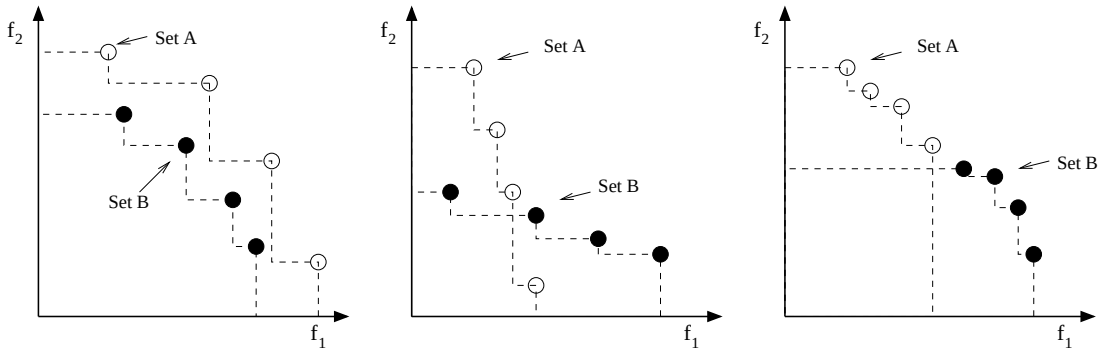

Figure 8: Hypervolume and Coverage Measure Example, (Left) $S(A) > S(B), C(A,B) = 1$ and $C(B,A) = 0$, (Middle) $S(A) = S(B)$ and $C(A,B) = C(B,A) = 0.25$, (Right) $S(A) > 0, S(B) > 0$ and $C(A,B) = C(B,A) = 0$.

2. *A and B cover two different regions.* If $S(A) > 0$, $S(B) > 0$, $C(A,B) = 0$ and $C(B,A) = 0$ then A and B must be two different regions of the space, as illustrated on the right side of Figure 8.

3. *A and B overlap, but neither dominates the other.* If $S(A) > 0$, $S(B) > 0$, $0 < C(A,B) < 1$ and $0 < C(B,A) < 1$ then A and B overlap, but neither dominates the other, as illustrated in the middle of Figure 8.

1.4 Research Contributions

Many MOEA techniques exist as will be detailed in Chapter 2. Most of these methods utilize the concept of Pareto optimality to give better fitness values, and thus better probability of mating, to those individuals residing near the Pareto front. MOEAs are most interesting when applied to non-intuitive complex systems. But, these systems are often computationally expensive. When these systems require independent computations to evaluate each objective, the computational expense grows with each additional objective. This class of problems will be identified as Independent Computational Expensive Objectives (ICEO) problems.

Current MOEA techniques have two characteristics that are *not* required for ICEO problems, and are exploited by this research:

1. The entire Pareto front in the multi-dimensional objective space is found.

2. The fitness methods require evaluation of all objectives in order to determine a valid fitness value.

For many problems the entire Pareto front is not required. Optimization problems are often specified by a set of requirements or goals for the desired system, and only the Pareto optimal solutions that meet these requirements are desired. These requirements define minimum values of performance for the measured objectives, which in turn specify "regions of interest" in the objective space, i.e., those regions where the minimum values of performance are exceeded.

The region of interest in objective space can be defined most abstractly much as the constraints are in the search space. The region of interest, $\boldsymbol{G}$, is the subset of $\mathbb{R}^q$ such that a set of t constraints is greater than 0, $\boldsymbol{r}(\boldsymbol{y}) = r_1(\boldsymbol{y}), r_2(\boldsymbol{y}), \cdots, r_t(\boldsymbol{y}) \geq \boldsymbol{0}$.

For many optimization problems, the problem can easily be recast into a maximization problem, and the constraints simply become a set of lower bounds on each of the objectives specified by the requirements. Note that some objectives may not have a requirement for a minimum performance. For these objectives, a lower bound can be set to the lower bound of the objective to include the entire span of this objective. In this case, the abstract description can be reduced to a set of q threshold tests, one for each of the objectives. Letting L_k be the threshold for the k^{th} objective, the region of interest, $\boldsymbol{G} \subset \boldsymbol{Y}$, is defined as those locations in $\boldsymbol{y}$ such that for all $k = 1$ to q, $y_k \geq L_k$. As illustrated in two dimensions in Figure 9, the q thresholds, $\boldsymbol{L}$, define a hypercube of dimension q in objective space, $h_f(\boldsymbol{L})$. This simplified definition for the region of interest, $G = h_f(\boldsymbol{L})$, will be utilized by this research. Once the Pareto front has been found in the region of interest, the designer can examine the trade-offs of the systems that exceed all specifications to select the system that is most desirable.

Another consideration for MOEA methods that exploit the region of interest, *ROI*, is that even though requirements for a system can be designed that are within the space R^q, this does not mean it is within the objective space $\boldsymbol{Y} \subset R^q$. In other words, just because there

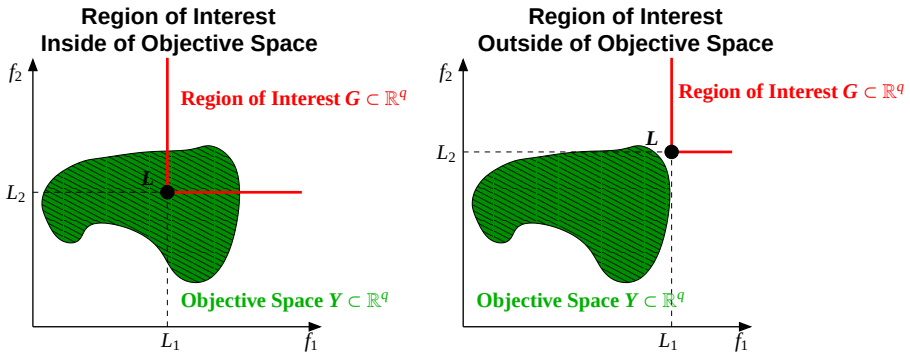

Figure 9: Region of Interest

is a set of requirements on a system does not mean that a system exists that can meet these requirements. For complex functions, $\boldsymbol{Y}$ can not be found without evaluation of all decision variables in the search space $\boldsymbol{X}$. If the mapping of all variables from the search space to the objective space was known *a priori*, then one would not need a complex method such as MOEAs to explore the search space. Therefore, the methods must evolve sets of Pareto fronts that move toward $\boldsymbol{G}$ without knowledge that it can reach $\boldsymbol{G}$. Figure 9 illustrates regions of interest that are inside and outside of the objective space.

In the course of research, (see Section 3.4.1), a method has been developed that reduces the time required for evolution by reducing the number of objective evaluations, while still evolving solutions that are Pareto optimal. To date, all other MOEAs require the evaluation of *all* objectives before a fitness value can be assigned to an individual. By removing this restriction, individuals that are determined to be less likely to produce children that reside closer to the region of interest after the evaluation of only a few objectives can terminate the evaluation of additional objectives. This allows the evolution of Pareto fronts that *push* solutions toward the desired region of interest, but degrade gracefully if the region of interest cannot be reached.

The original contributions of this thesis are:

1. Development of a hierarchical search space description that allows association of crossover and mutation settings with elements of the genotypic description.

2. Development of a method for parallel evaluation of individuals that removes the need for delays for synchronization.

3. Dynamical evolution of thresholds for objectives to allow partial evaluation of objectives for individuals.

4. Dynamic objective orderings to minimize the time required for unnecessary objective evaluations.

5. Application of MOEAs to the computationally expensive flare pattern design domain.

6. Application of MOEAs to the optimization of fielded missile warning receiver algorithms.

7. Development of a new method of using MOEAs for automatic design of pattern recognition systems.

In summary, *this research augments current Multiple Objective Evolutionary Algorithms with methods that dramatically reduce the time required to evolve toward a region of interest in objective space*. The research then demonstrates these methods with optimization in two complex objective domains as presented in Chapters 5 and 6.

1.5 Organization

Chapter 2 details the common selection mating and mutation methods of most EAs techniques. Also included are discussions of other common MOEA issues including parallel evaluation, initial population, and elitism. The details of the Strength Pareto Evolutionary Algorithm (SPEA) [61], and SPEA2 [62] techniques, of Zitzler, as well as the Niched Pareto Genetic Algorithm (NPGA) of Horn, Nafpliotis and Goldberg [30] are then analyzed, allowing the reuse and quantitative analysis of these components in the following Chapters.

Chapter 3 details the algorithms developed by this research. Original contributions for all MOEA domains include: a hierarchical genome description, ability to link mutation and crossover to specific genome attributes, and enhanced parallel evaluation methods. Innovations for ICEO specific problems are also presented including: the new concepts of dynamic thresholding, hypercube distance scaling, hypercube distance ordering, and auto ordering.

Chapter 4 presents quantitative analysis of these algorithms against the fast-running 0/1 Knapsack problem extended to multiple dimensions of Martello and Toth [42]. This analysis compares the efficiency for different combinations of the components of the SPEA, SPEAII, NPGA, and ICEO methods to reach regions of interest in objective space. Because the 0/1 Knapsack problem does not contain independent objectives, nor is it computationally expensive, the number of objectives required is minimized instead of minimizing the time required to evolve. The six MOPs developed by Deb [17] provide a suite of fast-running problems that are designed to contain many of the most difficult scenarios of MOPs. The best performing algorithms from the 0/1 Knapsack analysis are also investigated to insure that the resulting algorithms are robust.

Chapter 5 presents the results of the ICEO methods against a truly ICEO problem of designing flare pattern to defeat threat missiles. The search space for a flare pattern contains the selection of the timing, type of flare, and flare dispenser location for a set of flares. The performance of the patterns is evaluated using an energy-weighted centroid tracking algorithm. Evolution of patterns that cause large errors in the track point is desirable. The objectives contain the performance of the flare pattern against multiple angles and ranges. Because this is a true ICEO problem, results can be examined in terms of run-time.

Chapter 6 presents the application of the ICEO MOEA methods in the optimization of the AAR-44A missile warning receiver performance. The search space for this optimization includes 156 variables selected from the Operational Flight Program (OFP). The objectives are to maximize the probability of detection P_d of a threat, the negative of the

number of false alarms, $NFAC$, the average time to intercept before impact TTI_{Avg}, and the minimum time to intercept TTI_{Min}. The fitness evaluations were performed using a simulation that contained a rehosted version of the actual OFP. Large quantities of False Alarm (FA) data, available recorded threat missile shots, and a plethora of simulated missile shots were used for training and evaluation. Optimization resulted in dramatic improvements, especially in the P_d and FA performance.

Chapter 7 presents the application of the ICEO MOEA methods to a new form of Evolutionary Programming developed to work directly on block diagrams as opposed to traditional methods using inverted tree structures [36]. This method allows ICEO MOEA methods to not only modify the attributes of a system, but to also modify the topology of the system. Layered on top of the Ptolemy II simulation, high level components can be tuned and combined in novel ways to design a set of Pareto optimal systems. The application of this problem to the design of a filter to predict the income level of an individual based on attributes such as race, sex, and education level is presented and compared to other pattern recognition methods. Training and evaluation data originate from the 1994 census.

In conclusion, Chapter 8 offers a summary of the contributions and advancement in knowledge created in this body of research. This chapter also outlines areas of incomplete knowledge and offers suggestions for future research.

CHAPTER II

MULTIPLE OBJECTIVE EVOLUTIONARY ALGORITHMS

The previous chapter introduced the basics of single objective and multiple objective optimization problems along with the definitions for a search space and objective space. Chapter 1 then introduced the basics of Pareto optimality in a multi-dimensional objective space, the concept of the region of interest in the objective space.

This chapter expands upon these basics with detailed analysis of current Evolutionary Algorithm (EA) and Multiple Objective Evolutionary Algorithm (MOEA) techniques. The chapter begins with a detailed look at the standard EA concepts in Section 2.1 as a method for single objective optimization. These basic evolutionary algorithms concepts are then expanded to multiple objective problems in Section 2.2 with a detailed analysis of a sampling of current multiple objective evolutionary algorithm (MOEA) methods. The detailed understanding of the current MOEAs is required to understand the motivation for the research contributions presented in Chapter 3. Section 2.3 completes the chapter with a summary of the conclusions reached during this chapter.

2.1 Standard Evolutionary Algorithm Concepts

The three standard processes of EAs are selection, mating, and mutation. Additional considerations include the encoding of the genotypic description of an individual, the ability of select individuals to be available for selection for longer than a single generation, and methods for parallelizing evaluation of individuals. The following subsections detail the basics of each of these processes.

2.1.1 Genome Encoding

In EAs, an individual's genotypic description, $\boldsymbol{x}$, is encoded as a sequence of attributes. Attributes can be one of three basic data types that must be considered; sampled, enumerated or real. Other more complex data types can be created by aggregation of these base data types.

Many problems such as the 0/1 knapsack and Deb's T functions only require a single bit for each attribute (see Chapter 4). In this case, the search space contains 2^n possibilities. Other problems, such as the flare pattern design problem, (see Chapter 5), contain attributes that are sampled. For example, the time of ejection time for a flare may occur any time between 0.0 and 4.0 seconds, but only on 50 millisecond boundaries. In this case, there are $4.0/0.050 + 1 = 81$ samples, which requires 7 bits to fully encode, (i.e., $\lceil(log_2(\text{range}/\text{delta})\rceil + 1$, where $\lceil\rceil$ is the ceiling function, which rounds values to the next highest integer). Note that 2^7 can encode 128 values. Thus, to remain in the feasibility region, many possible bit values need to map to a single valid attribute value (see Section 1.1 for a definition of the feasibility region).

Similarly, the flare pattern problem also requires encoding of the type of flare ejected, which is an enumeration of possible names. These too must be encoded in a small sequence of bits, where each possible value maps to one of the possible flare types.

The AAR-44A OFP optimization (see Chapter 6), requires attributes that are valid over the entire $\mathbb{R}^1$ or at least a range of the $\mathbb{R}^1$ values. For these problems, the same approximation utilized by simulations and software implementations is used; i.e., the encoding of values as 64-bit double precision numbers.

2.1.2 Selection

The selection operator supports the choice of the individuals from a population that will be allowed to mate in order to create a new generation of individuals. EA methods attempt to develop fitness methods and elitism rules that find a set of Pareto optimal values quickly

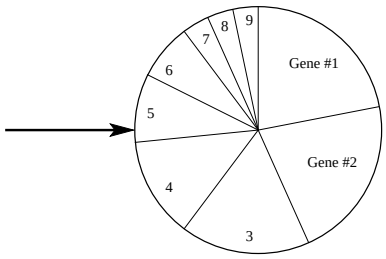

Figure 10: Area Weighted Roulette Wheel Selection

and reliably. But, ultimately it is the selection method that is responsible for the choice of a new generation of parents. Thus, the selection process is responsible for making the mapping from the fitness values of the input population to a probability of selection, and thus the probability of an individual's genetic material being passed to the next generation. Therefore, to fully understand the probability of selection for fitness values, the following sub-sections examine two common methods for selection: fitness proportionate and tournament. These sub-sections provide detailed analysis of the probability of selection for each of the methods, which has not been found in the literature. The details of the probability of selection will also be helpful in understanding the results of new algorithms, as will be required in Chapter 4.

2.1.2.1 Fitness Proportionate Selection

Fitness-proportionate selection, also known as area weighted roulette selection, gives a probability of selection to an individual in proportion to its fitness value, $P_{s,FPS}(i)$ and was proposed by Holland in 1975 [29]. The title, area weighted roulette selection, comes from the analogy that each individual is given an area of a roulette wheel that is proportionate to its fitness. The wheel is spun to determine which individual is chosen as a parent. An area weighted roulette wheel is illustrated in Figure 10.

Some methods desire maximization of the fitness value such as the weighted-sum method, and others such as the SPEA methods require a minimization of the fitness value. To transform a minimization problem to a maximization problem, fitness values are set to

the negative of the current value, $F(i) \leftarrow -F(i)$. Second, there must be a mapping of a desired fitness value to zero area on the roulette wheel [26]. Considering the population, P, of individuals, each with fitness $F(i)$, the minimum and maximum of the population can be found as:

$$F_{Min} = \min_{i \in P} F(i). \tag{5}$$

$$F_{Max} = \max_{i \in P} F(i) \tag{6}$$

The area of the roulette wheel for each individual $A(i)$ is

$$A(i) = \frac{F(i) - F_{Min}}{F_{Max} - F_{Min}}.$$

The total area of the roulette wheel is

$$A_{tot} = \sum_{j \in P} A(j).$$

Thus, the probability of selection for an individual is

$$\begin{aligned} P_{s,FPS}(i) &= \frac{A(i)}{A_{tot}} \\ P_{s,FPS}(i) &= \frac{F(i) - F_{Min}}{\sum_{j \in P} F(j) - F_{Min}}. \end{aligned} \tag{7}$$

Note that for individuals with $F(i) = F_{Min}$ the probability of selection is zero, $P_{s,FPS}(i) = 0$.

2.1.2.2 Tournament Selection

The tournament method allows selection using tournament rules. With this method a number of individuals, m, are chosen as competitors at random from the input pool [6]. The competitor with the *best* fitness becomes the parent. For maximization problems, the individual with the *best* fitness is the individual with the highest fitness value. Likewise, for minimization problems the individual with the smallest fitness value has the *best* fitness. In either minimization or maximization problems, if the fitness values of the randomly chosen individuals are equal, then one is chosen at random as the winner.

Tournament selection can occur with and without replacement. When using replacement, all individuals are reintroduced into the population of possible parents after the selection of each parent. Thus, the same parent can be chosen multiple times. Without replacement, those individuals selected as parents are removed from the population of possible parents as they are chosen. Thus, for each generation a possible parent can only have one child.

A common instance of tournament selection is binary tournament selection *with replacement*, BTSWR. With binary tournament selection, $m = 2$, two individuals are chosen at random from the input pool and the individual with the best fitness is chosen as a parent, as illustrated in Figure 11. For the binary tournament selection method with an input population of n individuals, the probability of selection for the best individual is $2/n$, and the probability of selection for the poorest individual is 0. The probability of selection for the remaining individuals, ($P_{s,BTSWR}$), can be found by ordering the individuals according to their fitness values. Let t_j be the index of the selected individual on the j^{th} trial. The probability of selection for the k^{th} individual between the lowest ranked individual 1 and the highest ranked individual is as follows:

$$\begin{aligned} P_{s,BTSWR}(k) &= P_s(k \text{ drawn on } 1^{st})P_s(t_2 < k | k \text{ drawn on } 1^{st}) \\ &\quad + P_s(t_1 < k)P_s(k \text{ drawn on } 2^{nd} | t_1 < k) \\ &= P_s(t_1 = k)P_s(t_2 < k | t_1 = k) + P_s(t_1 < k)P_s(t_2 = k | t_1 < k) \\ &= \left(\frac{1}{n}\right)\left(\frac{k-1}{n-1}\right) + \left(\frac{k-1}{n}\right)\left(\frac{1}{n-1}\right) \\ P_{s,BTSWR}(k) &= \frac{2(k-1)}{n(n-1)} \end{aligned} \tag{8}$$

The same methods can be used when more than two competitors are used. For example, when three competitors are chosen at random from the ranked pool, then the probability of the selection for the k^{th} individual becomes:

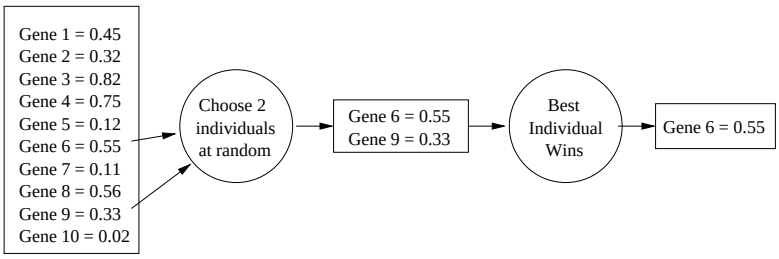

Figure 11: Binary Tournament Selection with Maximization

$$
\begin{aligned}
P_{s,TTSWR}(k) &= P_s(t_1 = k)P_s(t_2 < k | t_1 = k)P_s(t_3 < k | t_1 = k, t_2 < k) \\
&\quad + P_s(t_1 < k)P_s(t_2 = k | t_1 < k)P_s(t_3 < k | t_1 < k, t_2 = k) \\
&\quad + P_s(t_1 < k)P_s(t_2 < k | t_1 < k)P_s(t_3 = k | t_1 < k, t_2 < k) \\
&= \left(\frac{1}{n}\right)\left(\frac{k-1}{n-1}\right)\left(\frac{k-2}{n-2}\right) + \left(\frac{k-1}{n}\right)\left(\frac{1}{n-1}\right)\left(\frac{k-2}{n-2}\right) + \left(\frac{k-1}{n}\right)\left(\frac{k-2}{n-1}\right)\left(\frac{1}{n-2}\right) \\
&= 3\left(\frac{(k-1)(k-2)}{n(n-1)(n-2)}\right) \\
P_{s,TTSWR}(k) &= 3\left(\frac{(k-1)!(n-3)!}{(k-3)!n!}\right) \qquad (9)
\end{aligned}
$$

Allowing j individuals to be chosen instead of only 2 or 3 results in a mapping from a maximum probability of j/n for the best fitness individual to 0 for the $j-1$ individuals with the worst fitness. Thus the probability selecting of the k^{th} individual, using the tournament method choosing j competitors for $j \geq 2$, from a population of n ordered individuals is

$$P_{s,TSWR}(k, j) = \frac{j(n-j)!(k-1)!}{n!(k-j)!} \qquad (10)$$

Figure 12 gives an example plot of the probability of choosing each of the 10 individuals using 2, 3 and 4 competitors. As illustrated, the more competitors the higher the probability of selection for individuals with the best fitness.

Without replacement, tournament selection removes all individuals chosen by the tournament from the pool of possible subsequent parents. Therefore, the selection of two parents to create two children results in 2 fewer individuals in the pool of possible parents.

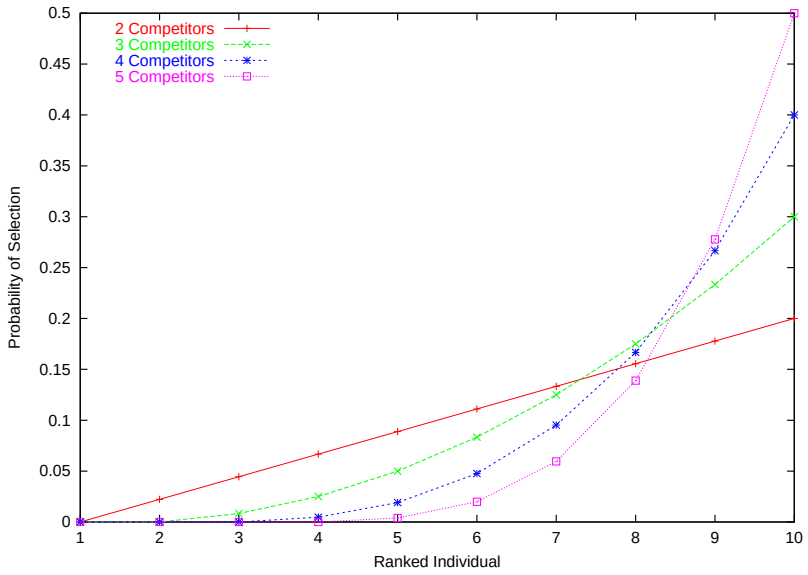

Figure 12: Probability of Selection for 2, 3, 4 and 5 Competitors Using Binary Tournament Selection on 10 Ranked Individuals

Thus, if it is desired to use tournament selection without replacement then the size of the input population for selection must be greater than the number of children to be created. If only one parent is selected, then probability of selection using binary tournament selection without replacement, ($P_{s,BTSNR}$), is the same as in Equation 8.

$$P_{s,BTSNR}(k,1) = \frac{2(k-1)}{n(n-1)} \tag{11}$$

The selection of the second parent becomes slightly more complicated. In this case we must take into consideration the probability that the rank selected on the first trial is greater or less than the value of k, ($P_{s,BTSNR}(t_1 > k)$ or $P_{s,BTSNR}(t_1 < k)$), and the conditional probability that the k^{th} value is selected on the second trail given the first is greater than or less than the k, ($P_{s,BTSNR}(t_2 = k|t_1 > k)$ or $P_{s,BTSNR}(t_2 = k|t_1 < k)$).

$$\begin{aligned} P_{s,BTSNR}(k,2) &= P_{BTSNR}(t_1 > k)P_{s,BTSNR}(t_2 = k|t_1 > k) + P_{BTSNR}(t_1 < k)P_{s,BTSNR}(t_2 = k|t_1 < k) \\ P_{s,BTSNR}(k,2) &= \sum_{j=k-1}^{n} \left(\frac{2(j-1)}{(n)(n-1)}\right)\frac{2(k-1)}{(n-1)(n-2)} + \sum_{j=1}^{k-1} \left(\frac{2(j-1)}{(n)(n-1)}\right)\frac{2(k-2)}{(n-1)(n-2)} \end{aligned} \tag{12}$$

The equation for three parent selection requires looking at four possible combinations of

the previous 2 selections:

$$\begin{aligned}
P_{s,BTSNR}(k,3) &= P_{s,BTSNR}(t_1 > k)P_{s,BTSNR}(t_2 > k)P_{s,BTSNR}(t_3 = k|t_1 > k, t_2 > k) \\
&\quad + P_{s,BTSNR}(t_1 > k)P_{s,BTSNR}(t_2 < k)P_{s,BTSNR}(t_3 = k|t_1 > k, t_2 < k) \\
&\quad + P_{s,BTSNR}(t_1 < k)P_{s,BTSNR}(t_2 > k)P_{s,BTSNR}(t_3 = k|t_1 < k, t_2 > k) \\
&\quad + P_{s,BTSNR}(t_1 < k)P_{s,BTSNR}(t_2 < k)P_{s,BTSNR}(t_3 = k|t_1 < k, t_2 < k) \\
P_{s,BTSNR}(k,3) &= \left(\sum_{j=k-1}^{n} \frac{2(j-1)}{(n)(n-1)}\right)\left(\sum_{j=k+1}^{n-1} \frac{2(j-1)}{(n-1)(n-2)}\right)\frac{2(k-1)}{(n-2)(n-3)} \\
&\quad + \left(\sum_{j=k-1}^{n} \frac{2(j-1)}{(n)(n-1)}\right)\left(\sum_{j=1}^{k-1} \frac{2(j-1)}{(n-1)(n-2)}\right)\frac{2(k-2)}{(n-2)(n-3)} \\
&\quad + \left(\sum_{j=1}^{k-1} \frac{2(j-1)}{(n)(n-1)}\right)\left(\sum_{j=k+1-1}^{n-1} \frac{2(j-1)}{(n-1)(n-2)}\right)\frac{2(k-2)}{(n-2)(n-3)} \\
&\quad + \left(\sum_{j=1}^{k-1} \frac{2(j-1)}{(n)(n-1)}\right)\left(\sum_{j=1}^{k-1-1} \frac{2(j-1)}{(n-1)(n-2)}\right)\frac{2(k-3)}{(n-2)(n-3)} \qquad (13)
\end{aligned}$$

Let $|C|$ be the number of individuals in the population C. The creation of $|C|$ children for a population C requires looking at the $2^{|C|-1}$ combinations of selections of less than and greater than for the previous $|C|-1$ selections, $(x \in [l,g]^{|C|-1})$. Define the function $LT(x, j)$ equal to the number of values of l in the first j dimensions of x.

$$LT(x,j) = \sum_{i=1}^{j} \begin{cases} 1 & x_i = l \\ 0 & x_i = g \end{cases} \qquad (14)$$

Then the probability of selection for the k^{th} individual on the $|C|$ selection of a parent using binary tournament selection without replacement is

$$P_{s,BTSNR}(k,|C|) = \sum_{x\in[l,g]^{|C|-1}} \left(\prod_{j=1}^{|C|-1} \begin{cases} \sum_{m=k+1-LT(x,j)}^{n-j+1} \frac{2(m-1)}{(n-j+1)(n-j)} & x_j = l \\ \sum_{m=1}^{k-1-LT(x,j)} \frac{2(m-1)}{(n-j+1)(n-j)} & x_j = g \end{cases} \right) \frac{2(k-1-LT(x,|C|-1)}{(n-|C|+1)(n-|C|)}. \qquad (15)$$

Thus the probability of selection for the k^{th} individual in any of the $|C|$ selections of a parent using binary tournament selection without replacement is

$$P_{s,BTSNR}(k, any|C|) = \sum_{i=1}^{|C|} P_{s,BTSNR}(k,i).$$

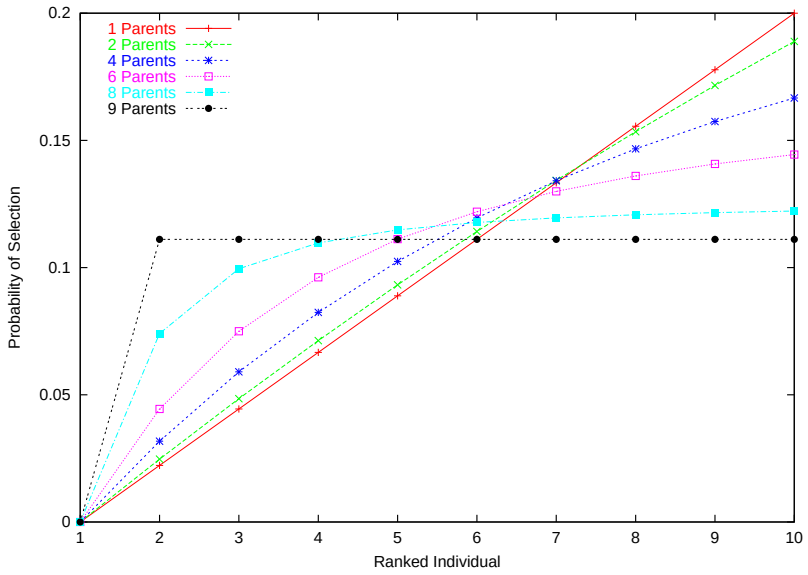

Figure 13: Cumulative Probability of Selection for 1, 2, 4, 6, 8 and 9 Parents Using Binary Tournament Selection Without Replacement on 10 Ranked Individuals

$$P_{s,BTSNR}(k, any|C|) =$$

$$\frac{\sum_{i=1}^{|C|}\sum_{x\in[l,g]^{i-1}}\left(\prod_{j=1}^{i-1}\begin{cases}\sum_{m=k+1-LT(x,j)}^{n-j+1}\frac{2(m-1)}{(n-j+1)(n-j)} & x_j = l\\ \sum_{m=1}^{k-1-LT(x,j)}\frac{2(m-1)}{(n-j+1)(n-j)} & x_j = g\end{cases}\right)\frac{2(k-1-LT(x,i-1)}{(n-i+1)(n-i)}}{|C|}. \quad (16)$$

Figure 13 illustrates the results of Equation 16 with 10 ranked individuals for values of $|C|$ in 1, 2, 4, 6, 8, and 9. Note that for $|C| = 1$ then the probability of selection is the same linear function as the binary tournament selection without replacement. At the other extreme when $|C| = 9$ or $|C| = n-1$ then the probability for the 2^{nd} to n^{th} ranked individuals is $\frac{1}{n-1}$. This implies that all but the poorest individual are given the same probability of mating, and thus the ranking of these individuals is not important. Without replacement, the number of children created can never equal the number of individuals in the current pool, ($|C| < n$), as the last in selection still requires two individuals for binary tournament selection, but only one individual remains.

The response of the probability for the 2^{nd} to n^{th} individual approaches $\frac{1}{n-1}$ as $|C|$ approaches $n-1$. This results in the loss of the influence on the fitness values on the probability of selection. Since the distribution of fitness values are one of two major influences in MOEA techniques, *binary tournament selection without replacement is not recommended*. If a tournament selection method is desired, one should use replacement.

2.1.2.3 Selection Comparisons

The fitness proportionate selection method utilizes the fitness values in order to calculate the area of the roulette wheel, and is therefore sensitive to the distribution of the fitness values. Because the tournament selection method makes a comparison, but does not depend on the relative differences, the tournament method is impervious to any non-linearities that may exist in the fitness values. For example, as illustrated in Figure 14, take a population of 3 individuals with fitness values of 0, 100 or $10,000$. Fitness proportionate selection would give the 0 fitness individual a zero probability of selection. The individual with a fitness of $10,000$ would have a 100 times larger probability of selection than the individual with a fitness of 100. The binary tournament method would also give the individual with the lowest fitness value of 0 a 0 probability of selection. But, the individual with a fitness of $10,000$ would have a probability of selection of $2/3$, and the individual with a fitness of 100 would have a probability of selection of $1/3$.

Therefore, the choice of the selection method must take into account the origination of the fitness values. *Weighted-sum techniques are most likely to contain a high variation in fitness values and should therefore use tournament selection if the range of fitness values is not well distributed*. As will be shown in Section 2.2, algorithms such as SPEA and NPGA often base fitness on the Pareto rank, which can be more easily mapped into a desired probability of selection. Therefore, *Pareto rank techniques can use either fitness proportionate or tournament selection.*

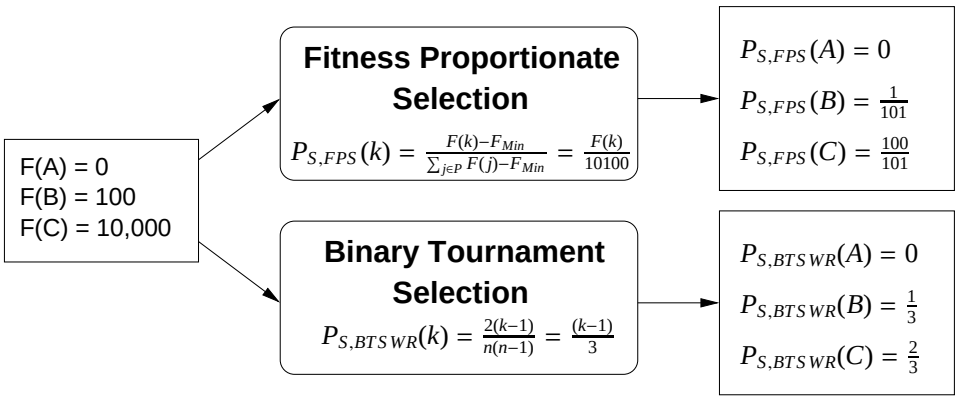

Figure 14: Example Comparison of Binary Tournament and Fitness Proportionate Selection with Large Variations in the Fitness Values

2.1.3 Crossover

Crossover is the feature of EAs that distinguishes it from other optimization techniques. As with other optimization techniques EAs must calculate a fitness value, select individuals for permutation for following iterations, and use mutation to prevent convergence on a local maxima. But, only EAs use crossover to take some attributes from one parent and other attributes from a second parent.

Unlike sexual reproduction in plants and animals, which is limited to two parents, EAs allow for input genetic information from two *or more* parents. As with selection, multiple crossover methods may be used in a single EA. For example, the SPEA method specifies ninety percent of individuals to originate from single point crossover and ten percent of individuals to originate from the clone crossover method.

The specifics of the two common crossover methods, (Clone and N-Point), are discussed in the following subsections. See Section 3.1.2 for a discussion of the location crossover method, developed during the course of this research. Other crossover methods such as a weighted average method [46, 47] also exist.

2.1.3.1 Clone Crossover

The clone crossover method mimics the creation of an identical copy of the genotypic description for of one of the parents. In this case, an exact copy of one of the parents is given to the child. The child will still be subject to the mutation operators described in the next section and therefore by the time of evaluation may not be the same as the parent. Use of clone crossover becomes equivalent to asexual reproduction. Selection still allows the best individuals to have a higher probability of propagating their genetic material into future generations and the worst individuals to have their genetic material selected out from future generations. Mutation allows insertion of new genetic possibilities. But, without the advantages of the exchange of genetic material between two individuals, only mutation can improve performance.

2.1.3.2 N-Point Crossover

The N-Point crossover method allows for crossover at an arbitrary number of points [8]. The literature often uses single point crossover. Using single point crossover as an example, the method first calculates a random position in the genome (i.e., sequence of bits encoding the individual). With the most common implementation, there are only two parents and a single crossover point, resulting in the creation of two children. The first child genome contains the first part of the first parent genome and the second part of the second parent genome. The second child genome contains the first part of the second parent genome and the second part of the first parent genome. The N-Point crossover method is illustrated in Figure 15 for single and dual point crossover.

Unfortunately, as given in literature, choosing an arbitrary location in the bit sequence can result in choosing a location that is in the middle of an attribute for all cases except a list of boolean values. This can ultimately result in mutating attributes at the point of crossover. The mutating effect can be seen in Figure 15 by thinking of each 4 bit hex value as an attribute and observing the appearance of attributes that are not inherited from either

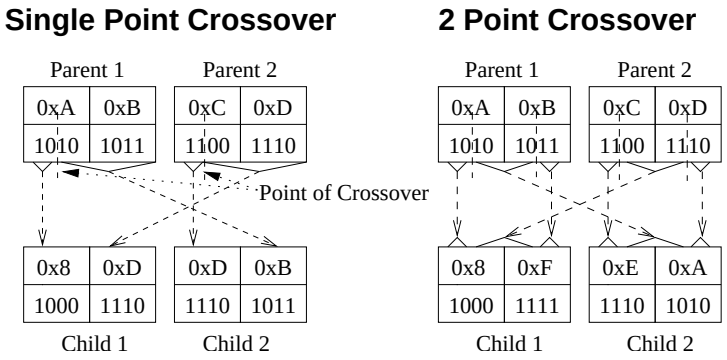

Figure 15: N-Point Crossover

parent. For sampled data types, the magnitude of the attribute is obtained from the first parent, (i.e., the upper bits of the attribute). The mutation effects are even worse for double precision encodings of attributes. If crossover occurs in the exponent of a double precision attribute then dramatic changes can occur in the magnitude of the attribute, not just the value of the component.

Let b be the number of bits required to encode a search space, N be the number of crossover locations, and n be the dimensionality of the search space, (i.e., the number of attributes). Then the probability of mutation due to N-Point crossover is

$$P_{m,c} = N\frac{b-n}{b}. \tag{17}$$

To avoid this mutation behavior, the implementation in this research forces crossover to only occur on the boundaries of attributes. This is implemented by calculating a random position in the bit sequence and then moving the position to the nearest attribute boundary. Nature also performs crossover on attribute values as well. For example, crossover in humans most often occurs on chromosome values.

2.1.4 Mutation

Unlike the crossover and selection operators, mutation is not necessarily applied to all individuals. Multiple mutation specifications are available: bit, uniform and normal. Each

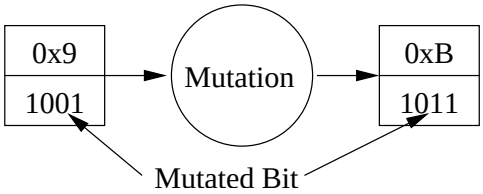

Figure 16: Bit Mutation

mutation type has an independent probability of being applied to each individual. Therefore, some individuals may be mutated by several mutation methods, some individuals may be mutated by a single mutation method, and some individuals may not be mutated at all.

Others [60] have developed and implemented methods that attempt to change the mutation rate over the course of the run. Typically, high mutation rates, e.g., 2 to 5 percent, are desired early in the run. As the run progresses these mutations rates are decreased, e.g., 1 to 0.5 percent.

2.1.4.1 Bit Mutation

The bit mutation method allows for the independent flipping of bits within the genome [5] The mutation method will only be applied to individuals selected with a desired probability, $P_{m,b}$. Once an individual has been selected for flipping, each bit is flipped with an independent probability, P_f. Figure 16 illustrates the bit mutation process.

The default behavior is to simply flip the bits, i.e., if a bit is 0 it becomes 1, and if it is 1 it becomes 0. It is also possible to specify sticky bit mutation, (i.e., that any bit chosen for mutation will be set to 1 regardless of its previous value, or that any bit chosen for mutation will be set to 0 regardless of its previous value). Sticky bit mutation is not typically used and will not be discussed in the following distribution discussions.

Bitwise mutation with flipping provides a uniform distribution for sampled attribute specifications that are encoded in a power of 2 number of bits. Unfortunately, when attributes are not divisible by a power of 2, then the values distributed outside of the feasibility region must be mapped back into the feasibility region, (see Section 1.1 for a definition

of the feasibility region). If clipping is used to map values back to the feasibility region, a disproportionate number of values will be set to is set to the minimum or maximum values depending on the clipping implementation utilized.

The same principles apply to enumerated specifications as well, (i.e., it provides a uniform distribution when there is a power of 2 number of enumerations). Otherwise, the method will result in a disproportionate number of values mutated to the last string in the list of possible values.

Bit mutation is deleterious for type double specifications. Because real attributes are typically valid over a region of $\mathbb{R}$, (e.g. $x \in [0.0, 1.0]$), bit mutation in the exponent will likely translate the value to a location outside of the feasibility region. Therefore, clipping the values to place them back in the feasibility region typically results in clustering near the limits of the variable. The format for IEEE-754 double precision numbers is 1 sign bit, S, 11 exponent bits, E, and 52 mantissa bits M. Therefore the probability of a large deviation (greater than one order of magnitude) is $(E + S)/64 = 12/64 = 0.1875$.

The length of attributes should also be considered for bit mutation. Single bit attributes will have a probability of $P_{m,b}P_f$ of being mutated. But, the longer the attribute the larger the probability of mutation. Attributes requiring b bits for encoding have a probability of $P_{m,b}\left(1 - (1 - P_f)^b\right)$ of being mutated. For example, letting $P_{m,b} = 1$, and $P_f = 0.01$, then the probability of mutation for a single bit attribute is 0.01. Using the same settings, the probability of mutation for a type double attribute, which requires 64 bits for encoding, would be $1 - (0.99)^{64} = 0.47$.

2.1.4.2 Uniform Mutation Specification

Uniform mutation methods allow for uniform distribution of attributes [22]. Unlike the bit mutation method, the uniform mutation method results in a true binomial distribution for all data types. Also, unlike the normal mutation method discussed below, the uniform mutation method is not dependent on the order of variables for enumerated data types. Therefore, the

uniform mutation method has the capabilities of the bitwise mutation operator for attributes that have an exact power of two encoding, but does not have the deleterious effects for other data type encodings. Thus, *uniform mutation is recommended as a replacement for bitwise mutation*.

This mutation method will only be applied to individuals selected with the desired probability, $P_{m,u}$. Once an individual has been selected for uniform mutation, each attribute of the genome has an independent probability, P_u, of being mutated. Note that the uniform mutation has the advantage that, unlike the bit mutation method, the mutation rate for an attribute is insensitive to the bit length.

2.1.4.3 Normal Mutation

Normal mutation methods allow for normal distribution of attributes. To specify a normal distribution, also known as a Gaussian distribution, a standard deviation and mean are required [22]. The mean can be set to the current value of the attribute. The standard deviation then specifies the width of the Gaussian distribution or "bell curve" around the mean. Enumerated data types are typically cast into integer values. Thus, normal mutation is more likely to move values to those names close in the specified list of values. This mutation method will only be applied to individuals selected with the desired probability, $P_{m,n}$. Once an individual has been selected for normal mutation, each attribute has an independent probability, P_n, of being mutated.

2.1.5 Elitism

Elitism allows individuals to produce children for more than a single generation. By definition elite individuals are those that are most likely to produce desirable offspring. When a new generation is required, the current generation of individuals, C, is added to the previous elite pool, E, to create the basis for the elite pool ($P = E \cup C$).

But, due to the time required to calculate the fitness for individuals that will be discussed in detail in Section 2.2, there is a limit on the number of elite individuals that should

be kept. Therefore, a set of rules for removing individuals from the new elite pool is applied. A simple rule for removal of individuals is to only take the individuals with the best fitness. This simplistic rule may result in undesired effects when the fitness values utilize the density of solutions in the objective space to deweight the fitness values. In this circumstance, the resulting population may not contain the genotypic and phenotypic diversity desired. Additional methods for elitism are presented in the Section 2.2 discussion of current MOEA algorithms.

Another method of keeping a geneticly diverse population for selection and thus mating is the incremental gene pool. With this method, a constant population size is maintained in the gene pool. After an individual is evaluated it is determined if the individual will replace another individual in the gene pool. Incremental gene pool methods become a special case of elitism where elitism rules are applied after the evaluation of each individual. Most often though, elitism rules are applied between each generation.

2.1.6 Parallel Objective Evaluation

For complex problems, the most time consuming task in EAs and MOEAs is the evaluation of the objective values $f(i)$. Fortunately, EAs can support the parallel evaluation of $f(i)$ using two general methods. The first is to parallelize the evaluation of lives within a generation and is referred to as a *Single Gene Pool* method. The second method supports evolution of *multiple gene pools* (typically 1 for each processor) in parallel, and allows migration of genome between the gene pools. The following sub-sections provide a detailed look at these two methods. Van Veldhuizen provides a more detailed breakdown [58].

2.1.6.1 Single Gene Pool Parallelism

Parallel evaluation of $f(i)$ is often performed because it requires little modification of the evolutionary process. As illustrated in Figure 17, this method can be thought of as a master-slave relation. The master process executes the evolutionary processes and communicates to slave processes/processors for calculation of $f(i)$.

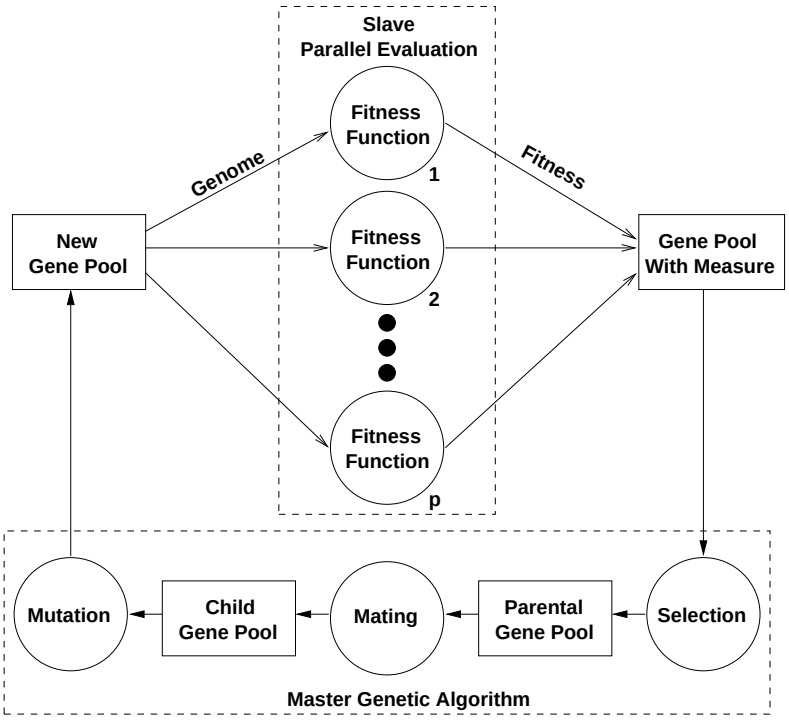

Figure 17: Single Gene Pool Parallel Life Evaluation

In 1999, Cantu-Paz and Goldberg [10] investigated the time required for single gene pool evolution

$$T_G = \frac{|C|T_f}{P} + (P-1)T_c \tag{18}$$

where P is number of processors, $|C|$ is the number of lives in the current generation, T_f is the time to calculate the fitness value, and T_c is the time for communication between the master and slave nodes. Taking the $\frac{\partial T_G}{\partial P} = 0$ results in number of processors that minimize the processing time P^* as:

$$P^* = \sqrt{\frac{|C|T_f}{T_c}} \tag{19}$$

The condition $\frac{\partial T_G}{\partial P} = 0$ is necessary but not sufficient for a minimum. Over the valid range of P it must be verified that $\frac{\partial T_G}{\partial P}$ is negative for $P < P^*$ and positive for $P > P^*$.

In 2000, Gwo, Hoffman and Hargrove [27] reported experimentally investigating this relationship. They found values of P^* much less that predicted in Equation (19). The results

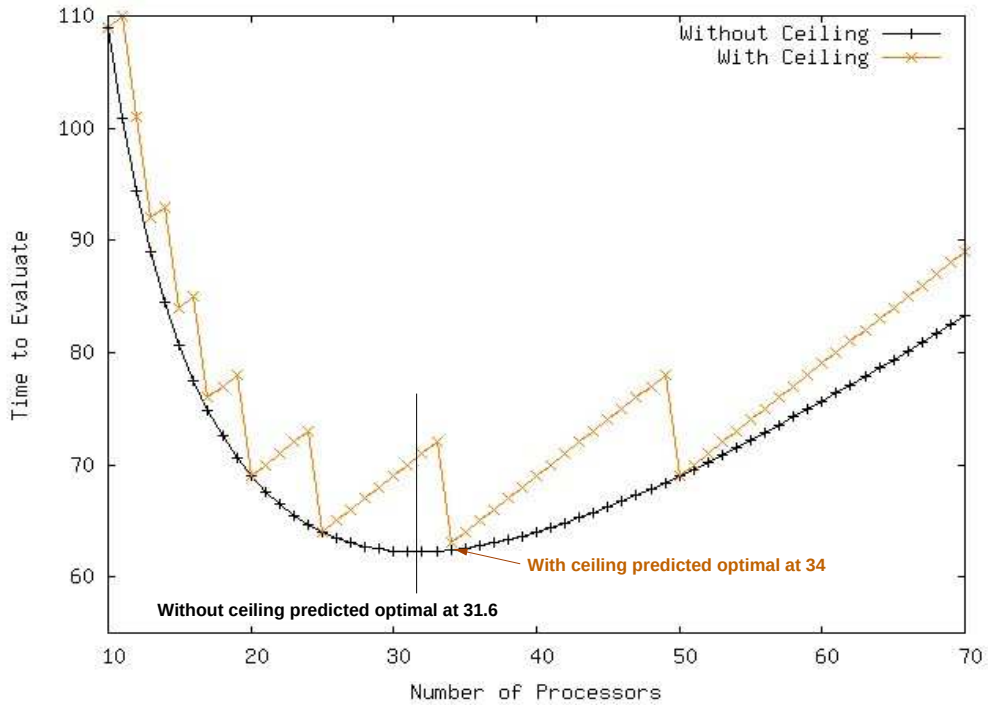

Figure 18: Results of Equations 18 and 20 with $T_F = 10$, $T_C = 1$, and $|C| = 100$

below prove that Equation 18 is too simplified for accurate prediction. For the equation to be accurate for a set of P machines that can all evaluate each individual in T_f, $|C|$ must be evenly divisible by P and P^* must be an integer. To remove this assumption, the ceiling function, $\lceil\rceil$, must be utilized as follows:

$$T_G = \lceil |C|/P \rceil T_f + (P-1)T_c. \tag{20}$$

Figure 18 illustrates the differences between the predictions of Equation 18 and Equation 20 for an example case with $T_F = 10$, $T_C = 1$, and $|C| = 100$. From this Figure note the shift in the number of processors for the minimum total processing time from 31.6 to 34. Also note that if the optimal value of P of 31 or 32 predicted from Equation 18 is chosen, the resulting T_G from Equation 20 would be highly unoptimal.

The "saw tooth" of Figure 18 is a result of the relationship between T_F and T_C. But, for the computationally expensive objective problems that are of concern for this thesis T_F is much larger that T_C. Removing this relationship by setting T_C to zero results in Figure

19, where the effects of the ceiling function are evident when then number of processors is between 50 and 99. The effects are a result of the parallel life evaluation requirement for completion of all runs within a generation before beginning the creation of the next generation. Setting T_C to zero in Equation 20 results in the best possible processing time of $T_G = T_f$ occurring when the number of processors P is equal to the size of the population C, i.e., each processors evaluates a single individual in parallel. The next best processing time requiring twice as long $T_G = 2T_f$ occurs for the number of processors P in the range $[|C|/2, |C| - 1]$.

From Equation 20, the worst case synchronization time is T_f. As EA problems' complexity increase, several factors create a diversity of T_f values, removing the predictability of T_G by either Equation 18 or Equation 20 and resulting in a worst case synchronization time as the worst case value of T_f. First, a disparity of processor speeds and loading on the processors can dramatically increase T_f on the slower and more loaded processors. Second, even with equivalent processors speeds and loading, with complex objective evaluations different genotypic descriptions may result in different values of T_f. Third, the implementation of MOEA ICEO techniques, which will be presented in Section 3.4, does not require the evaluation of all objectives. In fact, it is desirable to dramatically decrease T_f by identifying the fast evaluating objectives that can identify poor performing individuals quickly and thus remove the need to evaluate any other objectives for that individual. This results in a maximum T_f as the time required to evaluate all objectives and a minimum T_f as the time required to evaluate the fastest running objective.

In conclusion, if all processors do not finish evaluation simultaneously, which is highly likely for complex processors configurations and complex fitness functions, then many processors will be waiting at the end of the generation for other processors to complete evaluation. To remove these limitations, Section 3.2.2 will present new techniques for removing the need for synchronization.

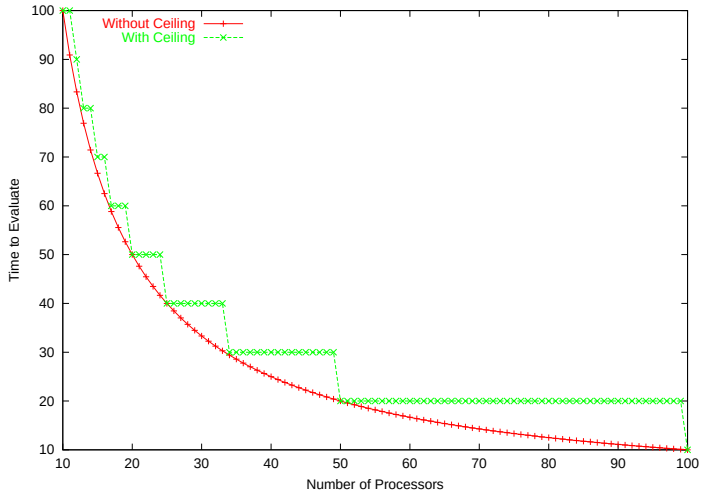

Figure 19: Results of Equations 18 and 20 with $T_F = 10$, $T_C = 0$, and $|C| = 100$

2.1.6.2 Multiple Gene Pool Parallelism

The second method of parallel EA distributes entire gene pools to each of the multiple computers [43]. This method is often referred to as the *island* method because each processor evolves while maintaining its own gene pool. Interaction with other computer nodes or islands is allowed via migration, as illustrated in Figure 20. The island method has been utilized by Andre and Koza (1998) [4] on a cluster of 64 transputers, which achieved super-linear performance over a single gene pool implementation. This super-linear performance is likely due to the additional genotypic diversity of the much larger overall gene pool. The transputer implementation required a system-dependent implementation for quick node-to-node communication.

One major advantage of the multiple gene pool implementation is its scalability and fault tolerance. The evolution rate can be increased (at least to a limit described below) by simply adding more computers. The implementation is fault tolerant as the loss of a single computing node is equivalent to, say, volcanic annihilation of a single island. The loss of an island does not stop the evolution of the overall world population.

Another advantage of the multiple gene pool implementation is the reduced number of calculations required for the fitness evaluation. As will be seen in Section 2.2 the time

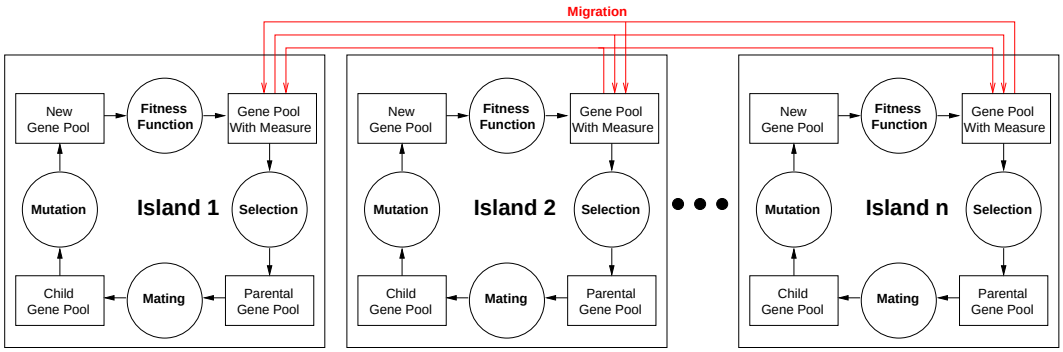

Figure 20: Distributed Gene Pool GA

required to calculate the fitness for a population of |C| individuals is often on the order $|C|^2$. By dividing the population among P processors, then the total time spent calculating the fitness is on the order of $(|C|/P)^2$ which is much less that $|C|^2$.

The island method introduces two additional parameters for adjustment: connection topology and migration rate. These parameters, as well as limits on scalability, have also been investigated by Cantu-Paz and Goldberg [10]. The limit on scalability defines the optimal number of *demes* or processors P^* as:

$$P^* = \sqrt{g\frac{\sqrt{\delta}-1}{\delta}}\sqrt{\frac{|C|T_f}{T_c}} \tag{21}$$

where g is the number of generations required for evolution and δ is the *degree*, which is the number of connections to neighboring islands.

From Equations 19 and 21, note that both the parallel gene pool and single gene pool methods have a theoretical optimal number of processors $O\left(\sqrt{\frac{|C|T_f}{T_c}}\right)$. Both $|C|$ and T_f typically increase as an optimization problem becomes more difficult. Therefore, both methods allow a large number of parallel processors to reduce execution time of difficult problems.

2.2 Multiple Objective Evolutionary Algorithms

This section discusses the concepts of current MOEA methods. In addition to the typical process of EAs detailed in the previous section, MOEAs must make a mapping from the

multi-dimensional objective space, $f(i)$, to the single dimension fitness value $F(i)$.

To fully examine each algorithm, pseudo-code is given as listings. As a measure of complexity, the listings are examined for the number of comparisons required. Comparisons are used because they are non-trivial operations that require at least two memory accesses, a comparison, and a conditional branch to implement in software. The conditional branch also prevents parallelization of software in the implementation. Even though the number of comparisons is not a complete indication of the time required to execute an algorithm, they are a good indicator of the complexity. Calculation of this complexity measure allows comparison among the various algorithms presented.

2.2.1 Weighted-Sum Algorithm

As briefly described in Section 1.2.2, the weighted-sum is the simplest method for mapping from multiple objective space to a single fitness value. The weighted-sum of objectives method combines the fitness values of each objective $f_k(i)$ by multiplying by a weight w_k.

$$F(i) = \sum_{k=1}^{q} w_k f_k(i) \tag{22}$$

The following listing details this algorithm and two comparison locations A and B.

```
// Loop through each individual in the population
for (i in P) {                                          // Comparison A
    // Initialize Sum to zero
    i.F = 0.0;
    // Loop through each objective
    for (k=1; k<=q; k++) {                              // Comparison B
        i.F += ( dWeight[k] * i.f[k] );
    } // End of loop through objectives
} // End of loop through individuals, i
```

Letting N_A and N_B represent the number of comparisons for each of these locations, q be the number of objectives, and $|P|$ be the number of individuals in the population, then

the number of comparisons total, N_{WS} is derived as:

$$\begin{aligned} \text{(from line 2) } N_A &= |P| \\ \text{(from line 6) } N_B &= q|P| \\ N_{WS} &= N_A + N_B \\ N_{WS} &= (1+q)|P|. \end{aligned} \tag{23}$$

2.2.2 Vector Evaluated Genetic Algorithms

In 1984, Schaffer [52] published a thesis in which he proposed the Vector Evaluated Genetic Algorithm (VEGA). This approach calculates fitness by selecting sub-populations based on each of the objectives. For example, if a population of $|C|$ is needed and there are q objectives, then q sub-populations with $|C|/q$ individuals for each objective are selected. The entire population of $|C|$ individuals is then subjected to normal crossover and mutation.

The following listing details this algorithm and two comparison locations A and B.

```
1  // Initialize counter
   iCount = 0;
   // Loop through each individual in the population
   for (i in P) {                                    // Comparison A
5     iCount++;
      // Find Objective to use
      iObjective=ceil(iCount/|P| * q);               // Comparison B
      i.F = i.f[iObjective];
   } // End of loop through individuals, i
```

From this listing, the total number of comparisons is:

$$\begin{aligned} \text{(from line 4) } N_A &= |P| \\ \text{(from line 8) } N_B &= |P| \\ N_{VEGA} &= N_A + N_B \\ N_{VEGA} &= 2|P|. \end{aligned} \tag{24}$$

Experiments by Murata and Ishibuchi published in 1995 [44] illustrated the tendency of

VEGA to create populations that congregate near the sections of the Pareto front that are near the axes of each objective. As pointed out by Coello in his 1996 thesis [15], this method may produce individuals with "middling" performance in all dimensions, but not solutions that represent a good compromise solution. These compromise solutions will never be chosen by the VEGA selection scheme, and will thus not be improved upon.

2.2.3 Elite Preserve Strategy

In 1995, Murata and Ishibuchi [44] proposed a method of selection in which the multiple objectives are combined into a single objective using a weighted sum of measures:

$$F(i) = \sum_{j=1}^{q} w_j f_j(i). \tag{25}$$

But, to force solutions to spread across the Pareto front, each time a pair of parents is selected, the weights are chosen for mapping from multiple objective space of dimension q to a single measure space $F(i)$ using:

$$w_j = \frac{random_j(\cdot)}{\sum_{k=1}^{q} random_k(\cdot)} \tag{26}$$

where $random_j(\cdot)$ is a non-negative random number. The fitness values of the entire gene pool are transformed to this new measure, and a pair of new parents is chosen using fitness proportionate selection. This method overcomes the cluster problems of VEGA but still has the same problem of all weighted-sum methods when dealing with concave portions of the Pareto front, as illustrated in Figure 4. The algorithm is:

```
// Loop, creating two new individuals at a time
for (j=1; j<|C|; j+=2) {                          // Comparison A
    // Initialize total weighted
    dTotalWeight=0;
    // Loop through each objective
    for (k=1;k<q;k++) {                           // Comparison B
      // Assign weight to random number
      weight[k] = rand();
      // Add to total weight
      dTotalWeight += weight[k];
```

```
        }
        // Calculate fitness
        // Loop through each individual in the population
14      for (i in P) {                                          // Comparison C
15          i.F = 0;
            // Loop through each objective
            for (k=1;k<q;k++) {                                 // Comparison D
                i.F += (i.f[k] * weight[k] / dTotalWeight);
            } // End of loop through objectives
20      } // End of loop through individuals i in P
        // Initialize Fitness Proportionate Selection
        FPS.init(P);                                            // Comparison E
        // Select two parents using FPS
        a = FPS(P);
25      b = FPS(P);
        // Crossover results in two children
        c,d = Crossover(a,b);
        // Assign new individuals to new popuation
        PNext += c;
30      PNext += d;
    } // End of loop through new children
```

From this listing, allow $|C|$ to be the size of the new population to be created, and allow $|P|$ to be the size of the input population, i.e., the current population plus the elite individuals. Comparison location A is responsible for the creation of the new individuals. Unlike the previous fitness algorithms, the fitness and selection process can not be separated. Therefore, N_A is removed from the total calculation, but the effects of the loop on the execution of other loops can not be neglected. This also causes the execution of the initialization of the fitness proportionate selection as indicated in line 20, which must execute a loop through all $|P|$ individuals once to find the minimum and maximum as given in Equations 5 and 6, resulting in $3|P|$ comparisons. It must then loop again another $|P|$ times to execute the sum given in Equation 7. Normally, the fitness proportionate selection initialization would be executed once. Instead it is executed N_A times, resulting in a total number of extra comparisons of $N_E = 4|P|(N_A - 1)$. The total number of comparisons for

the elite preserve fitness calculations is

$$\begin{aligned}
\text{(from line 2) } N_A &= |C|/2 \\
\text{(from line 6) } N_B &= qN_A \\
\text{(from line 14) } N_C &= |P|N_A \\
\text{(from line 17) } N_D &= qN_C = q|P|N_A \\
\text{(from line 22) } N_E &= 4|P|(N_A - 1) \\
N_{EPS,F} &= N_A + N_B + N_C + N_D + N_E - N_A \\
N_{EPS,F} &= \frac{|P||C|}{2}(q+5) + q\frac{|C|}{2} - 4|P| \qquad (27)
\end{aligned}$$

The term *Elite Preserve* for this method comes from the use of the second characteristic of the Murata/Ishibuchi method, which propagates some of the *elite* solutions that lie on the Pareto front of the current solution to the next generation.

The algorithm for finding Pareto individuals follows:

```
1  // Loop through each individual in the population
   for (i in P) {                                          // Comparison A
       // Initialize individuals as Pareto optimal
       i.bPareto = true;
5      // Loop through all other individuals in the population
       for (j in P, j != i) {                              // Comparison B
           // Loop through all objectives to see if i is dominated by j
           bool b_J_Dominates_I = true;
           for (k=1; k<=q; k++) {                          // Comparison C
10             // Not dominated if objective value is greater in any dimension
               if (i.f[k] > j.f[k]) {                      // Comparison D
                   // Set to dominated
                   b_J_Dominates_I = false;
                   // Since there is no need to check any other objective values,
15                 // break from loop through objectives
                   break;
               }
           } // End of loop through objectives
           // If j dominates i then i is not Pareto optimal
20         if (b_J_Dominates_I) {                          // Comparison E
               i.bPareto = false;
               break;
```

```
            }
24      } // End of loop through other individuals, j
25  } // End of loop through individuals, i
```

The upper limit on the total number of comparisons occurs when all loops must be executed a maximum number of times.

$$\begin{aligned}
\text{(from line 2) } N_A &= |P| \\
\text{(from line 6) } N_B &= |P|(|P|-1) \\
\text{(from line 9) } N_C &= qN_B \\
\text{(from line 11) } N_D &= N_C = qN_B \\
\text{(from line 20) } N_E &= N_B \\
N_{P,max} &= N_A + N_B + N_C + N_D + N_E \\
N_{P,max} &= (2+2q)|P|^2 - (1+2q)|P| \qquad (28)
\end{aligned}$$

The lower limit on the total number of comparisons occurs when all individuals are ordered as increasing in all objectives.

$$\begin{aligned}
\text{(from line 2) } N_A &= |P| \\
\text{(from line 6) } N_B &= |P| \\
\text{(from line 9) } N_C &= qN_B \\
\text{(from line 11) } N_D &= N_C = qN_B \\
\text{(from line 20) } N_E &= N_B \\
N_{P,min} &= N_A + N_B + N_C + N_D + N_E \\
N_{P,min} &= (3+2q)|P| \qquad (29)
\end{aligned}$$

From Equations 28 and 29, the total number of comparisons for Pareto optimality is bounded by:

$$(3+2q)|P| \leq N_P \leq (2+2q)|P|^2 - (1+2q)|P| \qquad (30)$$

Combining the results of Equation 27 and Equation 30 leads to a total number of comparisons for the elite preserve strategy as

$$N_{EPS} = N_{EPS,F} + N_P \tag{31}$$

$$\frac{|P||C|}{2}(q+5) + q\frac{|C|}{2} + (-1+2q)|P| \leq N_{EPS}$$

$$\leq (2+2q)|P|^2 + \frac{|P||C|}{2}(q+5) + q\frac{|C|}{2} - (5+2q)|P| \tag{32}$$

2.2.4 Niched Pareto Genetic Algorithms

In 1989, Goldberg [24] suggested use of the Pareto rank as the fitness measure instead of the single objective values or a weighted-sum of the objectives values. He also suggested the use of niche and speciation methods to create multiple sub-populations along the Pareto front.

In 1994 Horn, Nafpliotis, and Goldberg [30] presented a Niched Pareto Genetic Algorithm (NPGA) method. Experimentally, they found that simply using the Pareto rank as the fitness measure resulted in a population that congregated near the center of the Pareto front. Therefore, niching was introduced to create a set of solutions distributed more evenly along the Pareto front. The following subsections detail the Pareto rank and niching algorithms.

2.2.4.1 Pareto Rank

There are two methods for determining the relative importance of the dominated individuals. The first method is the *Pareto Rank Method 1* (PR1) and is the method used in the NPGA. Let P be a population of individuals. Then each individual i in P is dominated by some number of other points. Pareto optimal points are by definition not dominated by any points. Pareto rank 1 fitness values are one plus the number of points enclosed by the hypercube in front of the individual in question.

$$F_{pr1}(i) = 1 + |\{j | j \in P \wedge f(j) \in h_f(f(i))\}| \tag{33}$$

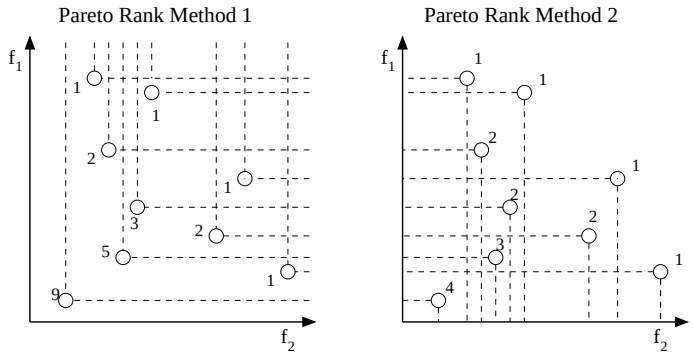

Figure 21: Pareto-optimal Ranking

The second method is the *Pareto Rank Method 2* (PR2). Although the PR2 method is not used by the NPGA, it is presented here for completeness. Like the Pareto rank method 1, the Pareto optimal solutions are given a rank of 1. The Pareto optimal individuals are then removed from the population, and the Pareto optimal individuals of the remaining population are given the next rank value of 2. This process is iterated until all members of the population have been assigned a rank. The following provides a recursive equation that can be used to find the PR2 rank:

$$F_{pr2}(i) = 1 + \max_{j|j \in P \wedge f(j) \in h_f(f(i))} F_{pr2}(j). \tag{34}$$

Figure 21 illustrates an example of the two Pareto rank methods. Note that because Pareto rank method 1 makes a measurement of the number of individuals dominating the point, it is making a measure of the density of individuals in front of the individual. In contrast, Pareto rank method 2 is less dependent on the density of elements in front of the individual.

The time required to calculate each of these methods is heavily dependent on the population being examined and the order of those individuals within the population. To examine the time constraints for PR1 method, let us first detail the algorithm using the pseudo-code given below.

```
1  // Loop through each individual in the population
```

```
for (i in P) {                                          // Comparison A
    // Initialize rank to one
    i.F = 1;
    // Loop through all other individuals in the population
    for (j in P, j != i) {                              // Comparison B
        // Loop through all objectives to see if i is dominated by j
        bool b_J_Dominates_I = true;
        for (k=1; k<=q; k++) {                          // Comparison C
            // Not dominated if objective value is greater in any dimension
            if (i.f[k] > j.f[k]) {                      // Comparison D
                // Set to dominated
                b_J_Dominates_I = false;
                // Since there is no need to check any other objective values,
                // break from loop through objectives
                break;
            }
        } // End of loop through objectives
        // If i dominates j then increment i's rank
        if (b_J_Dominates_I) i.F++;                     // Comparison E
    } // End of loop through other individuals, j
} // End of loop through individuals, i
```

The minimum number of comparisons occurs when the lowest level comparison requires only one comparison of an objective value. In this case,

$$\begin{aligned}
\text{(from line 2) } N_A &= |P| \\
\text{(from line 6) } N_B &= N_A(|P|-1) = |P|(|P|-1) \\
\text{(from line 9) } N_C &= N_B \\
\text{(from line 11) } N_D &= N_C = N_B \\
\text{(from line 20) } N_E &= N_B \\
N_{PR1,min} &= N_A + N_B + N_C + N_D + N_E \\
N_{PR1,min} &= 4|P|^2 - 3|P| \qquad (35)
\end{aligned}$$

The upper limit on the number of comparisons would require a comparison for each of the objectives for each of the individuals, thus requiring all loops to execute the maximum

number of times. This requires

$$\begin{aligned}
\text{(from line 2) } N_A &= |P| \\
\text{(from line 6) } N_B &= N_A(|P|-1) = |P|(|P|-1) \\
\text{(from line 9) } N_C &= qN_B \\
\text{(from line 11) } N_D &= N_C = qN_B \\
\text{(from line 20) } N_E &= N_B \\
N_{PR1,max} &= N_A + N_B + N_C + N_D + N_E \\
N_{PR1,max} &= (2+2q)|P|^2 - (1+2q)|P| \qquad (36)
\end{aligned}$$

From Equation 35 and Equation 36, the number of comparisons required for PR1 method can then be bounded by:

$$4|P|^2 - 3|P| \leq N_{PR1} \leq (2+2q)|P|^2 - (1+2q)|P| \qquad (37)$$

The PR2 algorithm, with nine comparison locations A through I, follows:

```
// Set the remaining population to the input population
R = P;                                                    // Comparison A
// Initialize current rank to one
iRank = 1;
// Loop until there are no remaining individuals
while (R.size() > 0) {                                    // Comparison B
    // Initialize current population to null
    C.null();
    // Loop through each individual in the population
    for (i in R) {                                        // Comparison C
        b_J_Dominates_I = true;
        // Loop through all other individuals in the population
        for (j in R, j != i) {                            // Comparison D
            // Loop through all objectives to see if i is dominated by j
            for (k=1; k<=q; k++) {                        // Comparison E
                // Not dominated if objective value is less in any dimension
                if (j.f[k] < i.f[k]) {                    // Comparison F
                    // Set to dominated
                    b_J_Dominates_I = false;
                    // Since there is no need to check any other objective values,
```

```
                    // break from loop through objectives
                    break;
                }
            } // End of loop through objectives
            // Break from this loop if individual i is dominated
            if (b_J_Dominates_I) {                    // Comparison G
                break;
            }
        } // End  of loop through all other individuals
        // If not dominated, then the individual, i, is Pareto optimal in the
        // remaining population, R.
        if (! b_J_Dominates_I) {                      // Comparison H
            // Assign fitness
            i.F = iRank;
            // Add individual to current population
            C.add(i);
        }
    } // End of loop through individual in the population
    // Remove the current population from the remaining population
    R.remove(C);                                      // Comparison I
    // Increment rank
    iRank++;
} // End of loop while there is individuals in the current population
```

From this the lower limit on the number of comparisons for the population occurs when all individuals are Pareto optimal and thus solved in a single iteration of the outer-most

loop. In this case, the total number of comparison for all $|P|$ individuals would be:

$$\begin{aligned}
\text{(from line 2) } N_A &= |P| \\
\text{(from line 6) } N_B &= 1 \\
\text{(from line 10) } N_C &= |P| \\
\text{(from line 13) } N_D &= |P|(|P| - 1) \\
\text{(from line 15) } N_E &= qN_D \\
\text{(from line 17) } N_F &= N_E = qN_D \\
\text{(from line 26) } N_G &= N_D \\
\text{(from line 32) } N_H &= N_C \\
\text{(from line 40) } N_I &= |P| = N_C \\
N_{PR2,min} &= N_A + N_B + N_C + N_D + N_E + N_F + N_G + N_H + N_I \\
N_{PR2,min} &= (1 + q)|P|^2 + (2 - 2q)|P| + 1 \qquad (38)
\end{aligned}$$

The calculation for N_C requires a different number of iterations for each loop, resulting in

$$N_C = \sum_{j=1}^{|P|} j$$

Likewise the calculation for N_D requires traversing the loop for all but the current individual, resulting in

$$N_D = \sum_{j=1}^{|P|} j(j - 1)$$

Fortunately, the sum of j and j^2 can be found using, ([53],[25]),

$$\sum_{j=1}^{n} j = \frac{n(n + 1)}{2} \qquad (39)$$

$$\sum_{j=1}^{n} j^2 = \frac{n(n + 1)(2n + 1)}{6} \qquad (40)$$

The calculation of the sum for $j(j-1)$ becomes

$$\begin{aligned}
\sum_{j=1}^{n} j(j-1) &= \sum_{j=1}^{n} j^2 - \sum_{j=1}^{n} j \\
&= \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2} \\
&= n(n+1)\left(\frac{(2n+1)}{6} - \frac{1}{2}\right) \\
\sum_{j=1}^{n} j(j-1) &= n(n+1)\frac{n-1}{3} \qquad (41)
\end{aligned}$$

The upper limit on the number of comparisons occurs when all individuals have different rank and are ordered in the last objective. This requires traversing all loops a maximum number of times, resulting in the following:

$$\begin{aligned}
\text{(from line 2) } N_A &= |P| \\
\text{(from line 6) } N_B &= |P| = N_A \\
\text{(from line 10) } N_C &= \sum_{j=1}^{|P|} j = \frac{|P|(|P|+1)}{2} \\
\text{(from line 13) } N_D &= \sum_{j=1}^{|P|} j(j-1) = \\
&= \frac{|P|(|P|+1)(|P|-1)}{3} \\
\text{(from line 15) } N_E &= qN_D \\
\text{(from line 17) } N_F &= N_E = qN_D \\
\text{(from line 26) } N_G &= N_D \\
\text{(from line 32) } N_H &= N_C \\
\text{(from line 40) } N_I &= |P| = N_A \\
N_{PR2,max} &= N_A + N_B + N_C + N_D + N_E + N_F + N_G + N_H + N_I \\
N_{PR2,max} &= \frac{2+2q}{3}|P|^3 + |P|^2 + \frac{10-2q}{3}|P|. \qquad (42)
\end{aligned}$$

From Equation 38 and Equation 42, the number of comparisons required for PR2

method can then be bounded by:

$$(1+q)|P|^2 + (2-2q)|P| + 1 \leq N_{PR2} \leq \frac{2+2q}{3}|P|^3 + |P|^2 + \frac{10-2q}{3}|P|. \tag{43}$$

From Equation 37 and Equation 43 then the order of the operations is:

$$N_{PR1} = O\left((2+2q)|P|^2\right) \tag{44}$$

$$N_{PR2} = O\left(\frac{2+2q}{3}|P|^3\right). \tag{45}$$

Therefore the PR1 method is much simpler to solve. Taking the ratio of N_{PR2}/N_{PR1} we find

$$\begin{aligned} \frac{N_{PR2}}{N_{PR1}} &= \frac{\left(\frac{2+2q}{3}|P|^3\right)}{((2+2q)|P|^2)} \\ &= |P|\frac{2+2q}{6+6q}. \end{aligned} \tag{46}$$

As the number of objectives q becomes large, then the ratio becomes

$$\frac{N_{PR2}}{N_{PR1}} \approx \frac{|P|}{3}. \tag{47}$$

Therefore, *without a compelling improvement in the performance of a MOEA with PR2 versus PR1 fitness, PR1 should be selected due to the additional complexity added by the PR2 method.*

2.2.4.2 Fitness Sharing

One method of creating niching pressure is *Fitness Sharing* (FS) [30]. With FS, the fitness measure is degraded by the *niche count* $m(f(i))$. The niche count for an individual i in a population of n genome is:

$$m(f(i)) = \sum_{j \in P, j \neq i} S_h(d(i,j)), \tag{48}$$

where $d(i,j)$ is the distance between two individuals i and j. The *Sharing Function* S_h is a decreasing function such that $S_h(0) = 1$ and $S_h(|d| \geq \sigma_{share}) = 0$. Figure 22 displays the most commonly used sharing function, the *triangular sharing function*, which is defined as:

$$S_h(d) = \begin{cases} 1 - |d|/\sigma_{share} & |d| \leq \sigma_{share} \\ 0 & d > \sigma_{share} \end{cases} \tag{49}$$

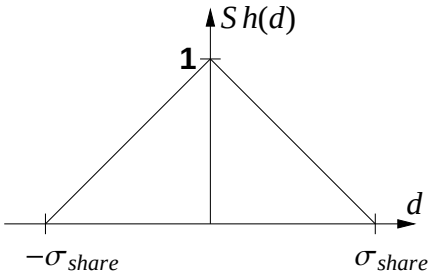

Figure 22: Triangular Sharing Function

Although other windowing functions could be used, a common function for the distance between two multi-objective fitness vectors $f(i)$ and $f(j)$ can be calculated using the Holder metric, which is:

$$d(f(i), f(j)) = \left(\sum_{k=1}^{q} (|f_k(i) - f_k(j)|)^p \right)^{\frac{1}{p}} \tag{50}$$

With $p = 2$, this results in the Euclidean distance. Figure 23 illustrates the effects of p for two dimensions. By setting $p < 2$ in the Holder equation, Euclidean distances are maximized along the objective axes. Niching with $p < 2$ results in larger niche counts for individuals along the objective axes than along the diagonal. Since solutions are de-weighted by the niche count, niching with $p < 2$ results in a lower fitness value and thus a lower probability of mating for those solutions clustered along the objective axes than those clustered diagonal to objective axes. Thus, the resulting population will have a more dense concentration of solutions diagonal to the objective axes than those parallel to the objective axes. This is a convenient method of finding solutions that are far from the objective axes.

The following details the fitness sharing algorithm, with four comparison locations A through D:

```
// Set the remaining population to the input population
R = P;                                                    // Comparison A
// Initialize current rank to one
iRank = 1;
// Loop until there are no remaining individuals
while (R.size() > 0) {                                    // Comparison B
```

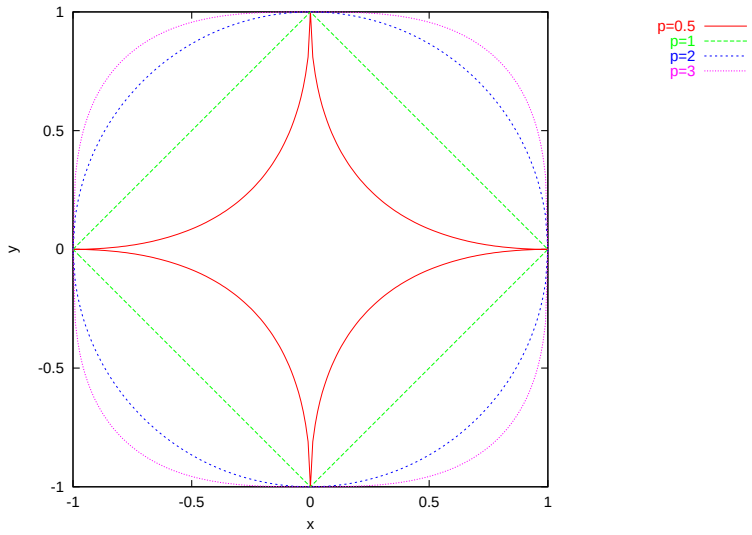

Figure 23: Holder Distance of 1.0 with Various Values of Holder Coefficient

```
    // Initialize current population to null
8   C.null();
    // Loop through each individual in the population
10  for (i in R) {                                    // Comparison C
        b_J_Dominates_I = true;
        // Loop through all other individuals in the population
        for (j in R, j != i) {                        // Comparison D
            // Loop through all objectives to see if i is dominated by j
15          for (k=1; k<=q; k++) {                    // Comparison E
                // Not dominated if objective value is less in any dimension
                if (j.f[k] < i.f[k]) {                // Comparison F
                    // Set to dominated
                    b_J_Dominates_I = false;
20                  // Since there is no need to check any other objective values,
                    // break from loop through objectives
                    break;
                }
            } // End of loop through objectives
25          // Break from this loop if individual i is dominated
            if (b_J_Dominates_I) {                    // Comparison G
                break;
            }
        } // End  of loop through all other individuals
30      // If not dominated, then the individual, i, is Pareto optimal in the
        // remaining population, R.
        if (! b_J_Dominates_I) {                      // Comparison H
             // Assign fitness
```

```
                i.F = iRank;
35              // Add individual to current population
                C.add(i);
            }
        } // End of loop through individual in the population
        // Remove the current population from the remaining population
40      R.remove(C);                                        // Comparison I
        // Increment rank
        iRank++;
    } // End of loop while there is individuals in the current population
```

From this the number of comparisons for the population is

$$\begin{aligned}
\text{(from line 2) } N_A &= |P| \\
\text{(from line 6) } N_B &= |P|(|P|-1) \\
\text{(from line 10) } N_C &= qN_B \\
\text{(from line 15) } N_D &= N_B \\
N_{FS} &= N_A + N_B + N_C + N_D \\
N_{FS} &= (2+q)|P|^2 - (1+q)|P| \qquad (51)
\end{aligned}$$

From Equation 51, evaluation of the niche count using Equation (48) for all $|P|$ individuals within the population requires $O(|P|^2)$ comparisons. A method of *continuously updated sharing* reduces the number of calculations by not calculating niche count $m(f(i))$ based on the entire current population, but rather a partially filled next generation. A method of *niche count sampling* further reduces the number of calculations by calculating the niche count only on a sampled subset of the next population. These methods have experimentally been shown to be effective approximations [30].

When the niche count is high, i.e., the density of solutions is high, the fitness value is decreased by the niche count, thus resulting in a lower probability of selection. The fitness then becomes:

$$F(i) = F_{PR1}\frac{f(i)}{m(f(i))} \qquad (52)$$

As an example, think of two individuals that are co-located in objective space. These individuals would have a niche count of 2, $m(i) = 2$. Thus, a maximization problem would divide the fitness values by 2. If a fitness proportionate selection method were utilized, the resulting probability of selection would be exactly a factor of two lower. Thus, the probability of selection would be spread more evenly among the Pareto optimal solutions.

The spread of the resulting solutions on the Pareto front can be controlled by the setting of the population size n and the niche radius σ_{share}. Assuming the desire to spread results across the entire Pareto front, then the area created by the Pareto front $Area_{pareto}$ should be divided into the number of niches N desired. Since the specification of σ_{share} specifies the niche volume in objective space of dimension q, then the intersection of the Pareto front with niche volume is a niche area of dimension $q - 1$. Since the niche area is proportional to $(\sigma_{share})^{q-1}$, then:

$$(\sigma_{share})^{q-1}\gamma\frac{Area_{pareto}}{N} \tag{53}$$

The $Area_{pareto}$ can be limited using knowledge of the limits of the current Pareto surface on each axis, $[(\min_{i\in P} f_1(i), \max_{i\in P} f_1(i)), \cdots, \left(\min_{i\in P} f_q(i), \max_{i\in P} f_q(i)\right)]$. A limit for the minimum area is the hyperplane containing these extremes. Thus, the lower limit becomes the Euclidean distance between the minimum and maximum points. For most problems, the Pareto front will be much more complex than a simple plane. A limit for the maximum area is the sum of the attribute ranges. Therefore,

$$\begin{aligned} Area_{pareto} &\geq \left(\sum_{j=1}^{q}\left(\max_{i\in P} f_j(i) - \max_{i\in P} f_j(i)\right)^2\right)^{\frac{1}{2}} \\ Area_{pareto} &< \sum_{j=1}^{q} |\max_{i\in P} f_j(i) - \max_{i\in P} f_j(i)|. \end{aligned}$$

Thus, the limits on σ_{share} are:

$$\sigma_{share} \geq \left(\frac{\left(\sum_{j=1}^{q} \left(\max_{i \in P} f_j(i) - \max_{i \in P} f_j(i) \right)^2 \right)^{\frac{1}{2}}}{N} \right)^{\frac{1}{q-1}} \tag{54}$$

$$\sigma_{share} < \left(\frac{\sum_{j=1}^{q} |\max_{i \in P} f_j(i) - \max_{i \in P} f_j(i)|}{N} \right)^{\frac{1}{q-1}} \tag{55}$$

In the two-dimensional case, the limits on the niching radius are:

$$\sigma_{share} \geq \frac{\sqrt{((\max_{i \in P} f_1(i) - \max_{i \in P} f_1(i))^2 + (\max_{i \in P} f_2(i) - \max_{i \in P} f_2(i))^2}}{N} \tag{56}$$

$$\sigma_{share} < \frac{|\max_{i \in P} f_1(i) - \max_{i \in P} f_1(i)| + |\max_{i \in P} f_2(i) - \max_{i \in P} f_2(i)|}{N} \tag{57}$$

The Holder Equation, given by Equation (50), treats each measure equally regardless of its scale. Utilizing a computer optimization example, imagine the cost is measured in dollars in the range $[1, 1000000]$ and the performance is measured in billions of floating point operations per second (GFLOP) in the range [0.1,10]. Thus a σ_{share} that samples the cost with several niches is not likely to have any distinct niches in the performance measure. A simple way to overcome this error is to scale each of the axes to a uniform range, for example $[0, 1]$.

If scaling is performed on each axis, then Equation 54 and 55 reduce to

$$\sigma_{share} \geq \left(\frac{\sqrt{q}}{N} \right)^{\frac{1}{q-1}} \tag{58}$$

$$\sigma_{share} < \left(\frac{q}{N} \right)^{\frac{1}{q-1}} \tag{59}$$

Figure 24 illustrates the reduction in complexity for the two-dimensional problem of scaling the multiple objectives.

The following listing details the objective scaling algorithm with seven comparison locations: A through G.

```
1  // Find min & max in each dimension
   // As a start, assign to first individual
```

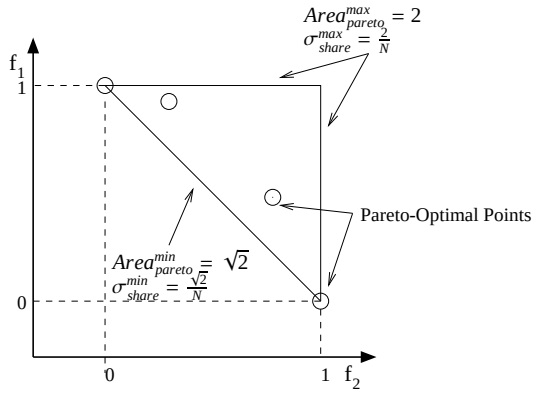

Figure 24: Two-Dimensional Minimum and Maximum Pareto Area with Objective Scaling

```
i=P[1];
for (k=1; k<=q; k++) {                                  // Comparison A
   fmin[k] = i.f[k];
   fmax[k] = i.f[k];
}
for (i in P; i != P[1] ) {                              // Comparison B
   for (k=1; k<=q; k++) {                               // Comparison C
      if (i.f[k] < fmin[k]) {                           // Comparison D
         fmin[k] = i.f[k];
      }
      if (i.f[k] > fmax[k]) {                           // Comparison E
         fmax[k] = i.f[k];
      }
   } // End of loop through objectives
} // End of loop through individuals in P
for (i in P) {                                          // Comparison F
   for (k=1; k<=q; k++) {                               // Comparison G
      i.f[k] = (i.f[k] - fmin[k]) / (fmax[k] - fmin[k]);
   } // End of loop through objectives
} // End of loop through individuals in P
```

From this the number of comparisons for the population is

$$\begin{aligned}
\text{(from line 4) } N_A &= q \\
\text{(from line 8) } N_B &= (|P| - 1) \\
\text{(from line 9) } N_C &= qN_B \\
\text{(from line 10) } N_D &= N_C = qN_B \\
\text{(from line 13) } N_E &= N_C = qN_B \\
\text{(from line 18) } N_F &= |P| \\
\text{(from line 19) } N_G &= qN_F \\
N_{OS} &= N_A + N_B + N_C + N_D + N_E + N_F + N_G \\
N_{OS} &= (2 + 4q)|P| - (1 + 2q) \qquad (60)
\end{aligned}$$

Combining Equations 37, 51 and 60, for the Pareto rank 1, fitness sharing and objective scaling algorithms respectively, results in

$$N_{NPGA} = N_{PR1} + N_{OS} + N_{FS} \qquad (61)$$

$$(6 + q)|P|^2 + (-2 + 3q)|P| - (1 + 2q) \leq N_{NPGA} \leq (4 + 3q)|P|^2 + q|P| - (1 + 2q) \qquad (62)$$

2.2.5 Strength Pareto Evolutionary Algorithm

In 1999 Zitzler [61, 64] developed the *Strength Pareto Evolutionary Algorithm (SPEA)*, which contains two important concepts. First, elitism allows those individuals determined to have the best fitness to propagate their genetic material into multiple future generations. With SPEA the elite gene pool contains only individuals from the current Pareto front. The second important concept of SPEA is *clustering*, which strives to maintain a finite elite pool size that is distributed on the Pareto front. With many problems the Pareto front is continuous, and thus the number of possible Pareto front points can become infinite. Also, as in the NPGA, the desire is to spread the points along the Pareto front. Too many

points congregated in an area of the Pareto front can result in a disproportionate number of individuals created in this area.

2.2.5.1 Clustering Algorithm

The purpose of the clustering algorithm is to prevent the number of Pareto optimal individuals in the elite population from becoming too large. The clustering algorithm operates by first grouping Pareto optimal individuals into a group of clusters, and then selecting a representative individual from each cluster. Define E_t as the current elite gene pool, or the current set of all Pareto individuals, and N_E as the maximum number of individuals desired in the new set of Pareto individuals E_{t+1}. Clustering is performed by first initializing a set of clusters $D = \{c_1, c_2, \cdots\}$ using each individual $i \in E_t$, thus $D = \bigcup_{i \in E_t} \{i\}$. The clusters with the minimal distance between pairs of clusters are then be amalgamated. Define $|c_1|$ as the number of Pareto individuals in set c_1. The distance d_c between two clusters c_1 and $c_2 \in D$ is given as the average distance between pairs of individuals across the two clusters

$$d_c = \frac{1}{|c_1| \cdot |c_2|} \cdot \sum_{i_1 \in c_1, i_2 \in c_2} d(f(i_1), f(i_2)), \tag{63}$$

where the function d is the distance, defined below, between two individuals i_1, and i_2 in objective space.

Although other windowing functions could be used, a common function for the distance d between two multi-objective fitness vectors $f(i)$ and $f(j)$ can be calculated using the Holder metric, given by Equation (50),

If the number of clusters $|D| > N_E$, then two clusters must be amalgamated into a larger cluster. This is done by choosing the two clusters c_1 and c_2 with the minimum distance d_c and replacing these clusters with $c_1 \cup c_2$. If $|D|$ is still larger than N_E, then the process of calculating d_c and amalgamating clusters is repeated.

Once the $|D| \leq N_E$, then a representative individual from each cluster is chosen to be a member of the new Pareto set E_{t+1}. The representative individual is the centroid (the individual with minimum average distance to all other individuals in the cluster).

The following listing details the SPEA clustering algorithm with fourteen comparison locations: A through N.

```
// No need to perform any operations if we already have the desired
// number of individuals
if (|E| <= N_E) {                                        // Comparison A
   return();
}
// Initialize |E| clusters to each contain one individual
for (i in E) {                                           // Comparison B
   Clusters.init(i);
}
// Loop until we have reduced to the desired number of individuals
while (|Clusters| > N_E) {                               // Comparison C
   // Initialize minimum distance as a large number
   dMininum = LARGE_NUMBER;
   // Loop through all cluster pairs
   for (iCluster in Clusters) {                          // Comparison D
      for (jCluster in Clusters,                         // Comparison E
                    jCluster != iCluster) {
         // Calculate distance between clusters by calculating
         // the average distance between individuals
         d = 0;
         for (iIndividual in iCluster) {                 // Comparison F
            for (jIndividual in jCluster) {              // Comparison G
               // Calculate Euclidean distance
               dSum = 0.0;
               for (k=1; k<=q; k++) {                    // Comparison H
                  dSum += (iIndividual.f[k] -
                                jIndividual.f[k])^2;
               }
               d += sqrt(dSum);
            } // End of loop through jIndividual
         } // End of loop through iIndividual
         // Scale distance
         d = d / (|iCluster| * |jCluster|);
         // Compare to find minimum
         if (d < dMinimum) {                             // Comparison I
             dMinimum = d;
             iMinimum = iCluster;
             jMinimum = jCluster;
         }
      } // End of loop through jCluster
   } // End of loop through iCluster
```

```
        // Join two clusters and remove one from main cluster list
43      iMinimum.add(jMinimum);
        Clusters.remove(jMinimum);
45  } // End of loop to reduce number of clusters
    // Select representative individual from each cluster
    // Clear current elite pool and then add back individuals
    E.null();
    for (iCluster in Clusters) {                        // Comparison J
50      // Initialize minimum distance as a large number
        dMininum = LARGE_NUMBER;
        for (iIndividual in iCluster) {                 // Comparison K
            // Calculate average distance to
            d = 0;
55          for (jIndividual in iCluster,               // Comparison L
                      jIndividual!=iIndividual) {
                // Calculate Euclidean distance
                dSum = 0.0;
                for (k=1; k<=q; k++) {                  // Comparison M
60                  dSum += (iIndividual.f[k]
                                - jIndividual.f[k])^2;
                }
                d += sqrt(dSum);
            } // End of loop through jIndividual
65          if (d < dMinimum) {                         // Comparison N
              dMinimum = d;
              iMinimum = iIndividual;
            }
        } // End of loop through iIndividual
70      // Add minimum individual to elite pool
        E.add(iMinimum);
    } // End of loop through clusters
```

The number of comparisons for A through E are:

$$\begin{aligned}
\text{(from line 3) } N_A &= 1 \\
\text{(from line 7) } N_B &= |P| \\
\text{(from line 11) } N_C &= |P| - |E| \\
\text{(from line 15) } N_D &= \sum_{j=|E|+1}^{|P|} j \\
&= \frac{1}{2}|P|^2 + \frac{1}{2}|P| + \frac{1}{2}|E|^2 + \frac{3}{2}|E| + 1 \\
\text{(from line 16) } N_E &= \sum_{j=|E|+1}^{|P|} j(j-1) = \frac{|P|(|P|+1)(|P|-1) - (|E|+1)(|E|+2)(|E|)}{3} \\
&= \frac{1}{3}|P|^3 - \frac{1}{3}|P| - \frac{1}{3}|E|^3 - |E|^2 - \frac{2}{3}|E|
\end{aligned} \tag{64}$$

From the above listing, the calculation of N_F results in the following sums:

$$\begin{aligned}
\text{(from line 21) } N_F &= \sum_{j=|E|+1}^{|P|} j(j-1)(|P|+1-j) \\
&= \sum_{j=|E|+1}^{|P|} -j^3 + j^2(|P|+2) - (|P|+1)j
\end{aligned} \tag{65}$$

Equation 65 requires the sum for j^3, which is given, ([53],[25]), as:

$$\sum_{j=1}^{n} j^3 = \left(\frac{n(n+1)}{2}\right)^2 \tag{66}$$

Equations 40 and 66 for the sum of j^2 and j^3 can be expanded to a polynomial as:

$$\sum_{j=1}^{n} j^2 = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n \tag{67}$$

$$\sum_{j=1}^{n} j^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2 \tag{68}$$

(69)

Likewise, the sum for j^2 and j^3 are needed for $n+1$. Equations 40 and 66 result in

$$\sum_{j=1}^{n+1} j^2 = \frac{1}{3}n^3 + \frac{3}{2}n^2 + \frac{13}{6}n + 1 \tag{70}$$

$$\sum_{j=1}^{n+1} j^3 = \frac{1}{4}n^4 + \frac{3}{2}n^3 + \frac{13}{4}n^2 + 3n + 1 \tag{71}$$

Using Equations 39, 40, 66, 67, 68, 70 and 71 for N_F results in the following:

$$\begin{aligned} N_F &= \sum_{j=|E|+1}^{|P|} j(j-1)(|P|+1-j) \\ &= \sum_{j=|E|+1}^{|P|} -j^3 + j^2(|P|+2) - (|P|+1)j \\ &= \frac{1}{12}|P|^4 + \frac{1}{6}|P|^3 - \frac{1}{12}|P|^2 - \frac{1}{6}|P| + \frac{1}{4}|E|^4 + \frac{5}{6}|E|^3 + \frac{3}{4}|E|^2 + \frac{1}{6}|E| \\ &\quad +|P|\left(-\frac{1}{3}|E|^3 + -1|E|^2 - \frac{2}{3}|E|\right) \end{aligned} \tag{72}$$

From line 22, the calculation of N_G results in the following sums:

$$\begin{aligned} N_G &= \sum_{j=|E|+1}^{|P|} j(j-1)(|P|+1-j)(|P|+1-j) \\ &= \sum_{j=|E|+1}^{|P|} j^4 + (-2|P|-3)j^3 + (|P|^2+4|P|+3)j^2 - (|P|+1)^2 j \end{aligned} \tag{73}$$

Equation 73 requires the sum for j^4, which is given, ([53],[25]), as:

$$\sum_{j=1}^{n} j^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} \tag{74}$$

Equation 74 can be expanded to a polynomial as:

$$\sum_{j=1}^{n} j^4 = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n \tag{75}$$

Also needed is the polynomial relation for the sum of $(n+1)^4$, resulting in the following:

$$\sum_{j=1}^{n+1} j^4 = \frac{1}{5}n^5 + \frac{3}{2}n^4 + \frac{13}{3}n^3 + 6n^2 + \frac{119}{30}n + 1 \tag{76}$$

Using Equations 39, 40, 66, 67, 68, 70, 71, 73, 75 and 76 N_G result in the following:

$$\begin{aligned} N_G &= \sum_{j=|E|+1}^{|P|} j(j-1)(|P|+1-j)(|P|+1-j) \\ &= \sum_{j=|E|+1}^{|P|} j^4 + (-2|P|-3)j^3 + (|P|^2+4|P|+3)j^2 - (|P|+1)^2 j \\ N_G &= +\frac{1}{30}|P|^5 + \frac{1}{12}|P|^4 - \frac{1}{12}|P|^2 - \frac{1}{30}|P| - \frac{1}{5}|E|^5 - \frac{3}{4}|E|^4 - \frac{5}{6}|E|^3 - \frac{1}{4}|E|^2 + \frac{1}{30}|E| \\ &\quad +|P|^2\left(-\frac{1}{3}|E|^3 - 1|E|^2 - \frac{2}{3}|E|\right) + |P|\left(+\frac{1}{2}|E|^4 + \frac{5}{3}|E|^3 + \frac{3}{2}|E|^2 + \frac{1}{3}|E|\right) \end{aligned} \quad (77)$$

The K^{th} comparison must loop through $|E|$ times for each cluster, plus an additional $|P|-|E|$ times for clusters with multiple individuals. The L^{th} comparison must loop through $|E| - 1$ by default and then an additional $|P| - |E| - 1$ times for the additional individuals. This results in the number of comparisons for the remaining items as:

$$\begin{aligned} \text{(from line 25) } N_H &= qN_G \\ \text{(from line 35) } N_I &= N_E \\ \text{(from line 49) } N_J &= |E| \\ \text{(from line 52) } N_K &= |E| + |P| - |E| = |P| = N_B \\ \text{(from line 55) } N_L &= |E| - 1 + |P| - |E| - 1 = |P| - 2 \\ \text{(from line 59) } N_M &= qN_L \\ \text{(from line 65) } N_N &= N_K = N_B \end{aligned}$$

Combining the results of the previous equations:

$$
\begin{aligned}
N_{SPEA,Clustering} &= N_A + 3N_B + N_C + N_D + 2N_E + N_F + (1+q)N_G + N_J + (1+q)N_L \\
&= 1 + 3|P| + |P| - |E| + \frac{1}{2}|P|^2 + \frac{1}{2}|P| + \frac{1}{2}|E|^2 + \frac{3}{2}|E| + 1 \\
&\quad +2\left(\frac{1}{3}|P|^3 - \frac{1}{3}|P| - \frac{1}{3}|E|^3 - |E|^2 - \frac{2}{3}|E|\right) \\
&\quad +\frac{1}{12}|P|^4 + \frac{1}{6}|P|^3 - \frac{1}{12}|P|^2 - \frac{1}{6}|P| + \frac{1}{4}|E|^4 + \frac{5}{6}|E|^3 + \frac{3}{4}|E|^2 + \frac{1}{6}|E| \\
&\quad\quad +|P|\left(-\frac{1}{3}|E|^3 + -1|E|^2 - \frac{2}{3}|E|\right) \\
&\quad +(1+q)\left(\frac{1}{30}|P|^5 + \frac{1}{12}|P|^4 - \frac{1}{12}|P|^2 - \frac{1}{30}|P|\right) \\
&\quad\quad +(1+q)\left(-\frac{1}{5}|E|^5 - \frac{3}{4}|E|^4 - \frac{5}{6}|E|^3 - \frac{1}{4}|E|^2 + \frac{1}{30}|E|\right) \\
&\quad\quad +(1+q)|P|^2\left(-\frac{1}{3}|E|^3 - 1|E|^2 - \frac{2}{3}|E|\right) \\
&\quad\quad +(1+q)|P|\left(+\frac{1}{2}|E|^4 + \frac{5}{3}|E|^3 + \frac{3}{2}|E|^2 + \frac{1}{3}|E|\right) \\
&\quad +|E| + (1+q)(|P| - 2)
\end{aligned}
\tag{78}
$$

Combining like terms the final equation for the number of comparisons for SPEA clustering becomes:

$$
\begin{aligned}
N_{SPEA,Clustering} &= \frac{1}{30}(1+q)|P|^5 + \left(\frac{1}{12}q + \frac{1}{6}\right)|P|^4 + \left(\frac{5}{6}\right)|P|^3 + \left(-\frac{1}{12}q + \frac{1}{3}\right)|P|^2 \\
&\quad\quad +\left(\frac{29}{30}q + 4\frac{19}{30}\right)|P| - 2q \\
&\quad +\left(-\frac{1}{5}(1+q)\right)|E|^5 + \left(-\frac{3}{4}q - \frac{1}{2}\right)|E|^4 + \left(-\frac{5}{6}q - \frac{2}{3}\right)|E|^3 + \left(-\frac{1}{4}q - 1\right)|E|^2 \\
&\quad\quad +\left(\frac{1}{30}q + \frac{11}{30}\right)|E| \\
&\quad +|P|^2\left(-\frac{1}{3}(1+q)|E|^3 - (1+q)|E|^2 - \frac{2}{3}(1+q)|E|\right) \\
&\quad +|P|\left(\frac{1}{2}(1+q)|E|^4 + \left(\frac{5}{3}q + \frac{4}{3}\right)|E|^3 + \left(\frac{3}{2}q + \frac{1}{2}\right)|E|^2 + \left(\frac{1}{3}q - \frac{1}{3}\right)|E|\right)
\end{aligned}
\tag{79}
$$

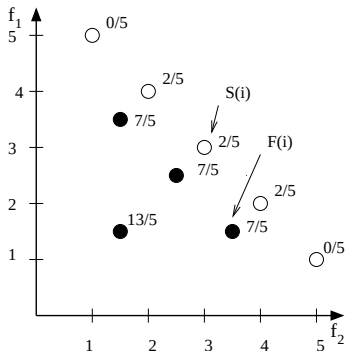

Figure 25: SPEA Fitness Examples

Equation 79 shows the number of comparisons as order $|P|^5$, which is much larger than the order $|P|^3$ of any other algorithm.

2.2.5.2 Strength Calculation

Once the new Pareto set E_{t+1} is selected, the fitness of individuals in E_{t+1} and C, the current population, can be assigned. The fitness for each individual $i \in E_{t+1}$, called the strength, is defined as a real value $S(i) \in [0, 1)$, where $S(i)$ is proportional to the number of population members dominated by i in P_t, the current set of dominated individuals. Letting $|C|$ be the number of members in C,

$$S(i) = \frac{|\{j | j \in C \wedge f(i) \succeq f(j)\}|}{|C| + 1}. \tag{80}$$

The fitness for an individual $j \in C$ is then calculated as the sum of the strengths of all individuals $i \in E_{t+1}$ that dominate j

$$F(j) = 1 + \sum_{i \in E_t, f(i) \succeq f(j)} S(i). \tag{81}$$

Note that the fitness, $F(j) \in [1, |E_t|)$, for the dominated individuals, $j \in C$, is always greater than the fitness, $S(i) \in [0, 1)$, for any non-dominated individuals, $i \in E_{t+1}$, and thus has less probability of selection. An example of S and F is illustrated in figure 25.

The following listing details the SPEA strength algorithm with five comparison locations: A through E.

```
1  // Loop through each individual in the population
2  for (i in E) {                                          // Comparison A
       // Initialize to zero
       i.F = 0;
5      // Loop through all other individuals in the population
       for (j in C) {                                      // Comparison B
           // Loop through all objectives to see if i is dominated by j
           bool b_I_Dominates_J = true;
           for (k=1; k<=q; k++) {                          // Comparison C
10             // Does not dominate if objective value is greater in any dimension
               if (j.f[k] > i.f[k]) {                      // Comparison D
                   // Set to dominated
                   b_I_Dominates_J = false;
                   // Since there is no need to check any other objective values,
15                 // break from loop through objectives
                   break;
               }
           } // End of loop through objectives
           // If i dominates j then increment fitness
20         if (b_I_Dominates_J) i.F++;                     // Comparison E
       } // End of loop through other individuals, j
       // Adjust fitness by number of possible individuals
       i.F = i.F / |C|;
   } // End of loop through individuals, i
```

This algorithm implements Equation 80 for the calculation of the fitness of the elite individuals. From this the number of comparisons for the population is

$$\begin{aligned}
\text{(from line 2) } N_A &= |E| \\
\text{(from line 6) } N_B &= |E||C| \\
\text{(from line 9) } N_C &= qN_B \\
\text{(from line 11) } N_D &= N_C = qN_B \\
\text{(from line 20) } N_E &= N_B \\
N_{SPEA,Strength} &= N_A + N_B + N_C + N_D + N_E \\
N_{SPEA,Strength} &= -(2+2q)|E|^2 + |E| + |P|(2+2q)|E| \qquad (82)
\end{aligned}$$

The following listing details the SPEA fitness algorithm with five comparison locations:

A through E.

```
1  // Loop through each individual in the population
   for (i in C) {                                          // Comparison A
       // Initialize fitness to one
       i.F = 1.0;
5      // Loop through all individuals in the elite population
       for (j in E) {                                      // Comparison B
           // Loop through all objectives to see if i is dominated by j
           bool b_J_Dominates_I = true;
           for (k=1; k<=q; k++) {                          // Comparison C
10             // Does not dominate if objective value is greater in any dimension
               if (i.f[k] > j.f[k]) {                      // Comparison D
                   // Set to dominated
                   b_J_Dominates_I = false;
                   // Since there is no need to check any other objective values,
15                 // break from loop through objectives
                   break;
               }
           } // End of loop through objectives
           // If i dominates j then add strength of elite individual
20         if (b_J_Dominates_I) i.F += j.F;                // Comparison E
       } // End of loop through other individuals, j
   } // End of loop through individuals, i
```

This algorithm implements Equation 81 for the calculation of the fitness of the dominated individuals. From this the number of comparisons for the population is

$$\begin{aligned}
\text{(from line 2) } N_A &= |C| \\
\text{(from line 6) } N_B &= |C||E| \\
\text{(from line 9) } N_C &= qN_B \\
\text{(from line 11) } N_D &= N_C = qN_B \\
\text{(from line 20) } N_E &= N_B \\
N_{SPEA,Fitness} &= N_A + N_B + N_C + N_D + N_E \\
N_{SPEA,Fitness} &= |P| - (2+2q)|E|^2 - |E| + |P|(2+2q)|E| \qquad (83)
\end{aligned}$$

The total number of calculations for the SPEA algorithm is the sum of Equations 30, 79,

82 and 83

$$
\begin{aligned}
N_{SPEA} &= N_P + N_{SPEA,Clustering} + N_{SPEA,Strength} + N_{SPEA,Fitness} \\
N_{SPEA} &= +\frac{1}{30}(1+q)|P|^5 + \left(\frac{1}{12}q + \frac{1}{6}\right)|P|^4 + \left(\frac{5}{6}\right)|P|^3 + \left(1\frac{11}{12}q + 2\frac{1}{3}\right)|P|^2 \\
&\quad + \left(1\frac{1}{30}q + 4\frac{19}{30}\right)|P| - 2q \\
&\quad + \left(-\frac{1}{5}(1+q)\right)|E|^5 + \left(-\frac{3}{4}q - \frac{1}{2}\right)|E|^4 + \left(-\frac{5}{6}q - \frac{2}{3}\right)|E|^3 + \left(-4\frac{1}{4}q - 5\right)|E|^2 \\
&\quad + \left(\frac{1}{30}q + \frac{11}{30}\right)|E| \\
&\quad + |P|^2\left(-\frac{1}{3}(1+q)|E|^3 - (1+q)|E|^2 - \frac{2}{3}(1+q)|E|\right) \\
&\quad + |P|\left(\frac{1}{2}(1+q)|E|^4 + \left(\frac{5}{3}q + \frac{4}{3}\right)|E|^3 + \left(\frac{3}{2}q + \frac{1}{2}\right)|E|^2 + \left(4\frac{1}{3}q + 3\frac{2}{3}\right)|E|\right) \quad (84)
\end{aligned}
$$

2.2.6 Strength Pareto Evolutionary Algorithm II

In 2001 Zitzler [63] built upon the SPEA algorithm to improve its performance. This algorithm contains three important differences. First, the fitness is calculated differently. Second, the SPEAII algorithm contains an explicit density measurement calculation for de-weighting individuals near each other. Third, the SPEAII algorithm keeps a fixed elite pool size with a different method for limiting the size.

2.2.6.1 Raw Fitness Calculation

The fitness is calculated by first finding the *strength* of each point in the current and elite ($C + E_t$) pool. The *strength* for each member of the current and elite pool is the number of individuals dominated by that individual.

$$S(i) = |\{j | j \in (C \cup E_t) \wedge f(i) \geq f(j)\}| \quad (85)$$

Each individual in the current and elite pool is assigned a *raw fitness* that is the sum of the *strengths* of those individuals that dominate it.

$$R(i) = \sum_{j \in (C \cup E_t), f(j) \geq f(i)} S(f(j)) \quad (86)$$

The following listing details the SPEAII raw fitness algorithm with ten comparison locations: A through J.

```
// Loop through each individual calculating the strength, S
for (i in P) {                                          // Comparison A
    i.S = 0;    // Initialize to zero
    // Loop through all other individuals in the population
    for (j in P, j!=i) {                                // Comparison B
        // Loop through all objectives to see if i is dominated by j
        bool b_I_Dominates_J = true;
        for (k=1; k<=q; k++) {                          // Comparison C
            // Does not dominate if objective value is greater in any dimension
            if (j.f[k] > i.f[k]) {                      // Comparison D
                b_I_Dominates_J = false; // Set to not dominated
                break; // No need to check any other objective values.
            }
        } // End of loop through objectives
        // If i dominates j then increment fitness
        if (b_I_Dominates_J) i.S++;                     // Comparison E
    } // End of loop through other individuals, j
} // End of loop through individuals, i
// Calculate the raw fitness, F, for each individual
for (i in P) {                                          // Comparison F
    i.F = 0; // Initialize fitness to zero
    // Accumulate strengths of those individuals
    // that dominate the current individual i
    for (j in P, j!=i) {                                // Comparison G
        // Loop through all objectives to see if i is dominated by j
        bool b_J_Dominates_I = true;
        for (k=1; k<=q; k++) {                          // Comparison H
            // Does not dominate if objective value is greater in any dimension
            if (i.f[k] > j.f[k]) {                      // Comparison I
                b_J_Dominates_I = false; // Set to not dominated
                break; // No need to check any other objective values.
            }
        } // End of loop through objectives
        // If i dominates j then increment fitness
        if (b_J_Dominates_I) i.F += j.S;                // Comparison J
    } // End of loop through other individuals, j
} // End of loop through individuals, i
```

This algorithm implements Equations 85 and 86 for the calculation of the raw fitness.

From this listing, the ten comparison locations, A through J, result in the number of comparisons as:

$$\begin{aligned}
\text{(from line 3) } N_A &= |P| \\
\text{(from line 7) } N_B &= |P|(|P|-1) \\
\text{(from line 10) } N_C &= qN_B \\
\text{(from line 10) } N_D &= N_C = qN_B \\
\text{(from line 12) } N_E &= N_B \\
\text{(from line 21) } N_F &= |P| = N_A \\
\text{(from line 32) } N_G &= |P|(|P|-1) = N_B \\
\text{(from line 35) } N_H &= qN_G = qN_B \\
\text{(from line 47) } N_I &= N_H = qN_B \\
\text{(from line 46) } N_J &= N_G = N_B \\
N_{SPEAII,Raw} &= N_A + N_B + N_C + N_D + N_E + N_F + N_G + N_H + N_I + N_J \\
N_{SPEAII,Raw} &= (4+4q)|P|^2 - (2+4q)|P| \qquad (87)
\end{aligned}$$

2.2.6.2 Density Calculation

The SPEAII algorithm then utilizes a density measure to de-weight the raw fitness values. The density estimation technique is a variation of the k^{th} nearest neighbor method. For each individual i in the current and elite pool $P_t \cup E_t$, the distances in objective space to all individuals j in the current and elite pool are calculated and stored in a list. This list is then sorted in increasing order. The distance to the k^{th} individual, denoted as $\sigma_{i,k}$ is then selected, where k is the square root of the total number of individuals

$$k = \sqrt{|P_t \cup E_t|}. \qquad (88)$$

The density is then defined as:

$$D(i) = \frac{1}{\sigma_{i,k} + 2}. \qquad (89)$$

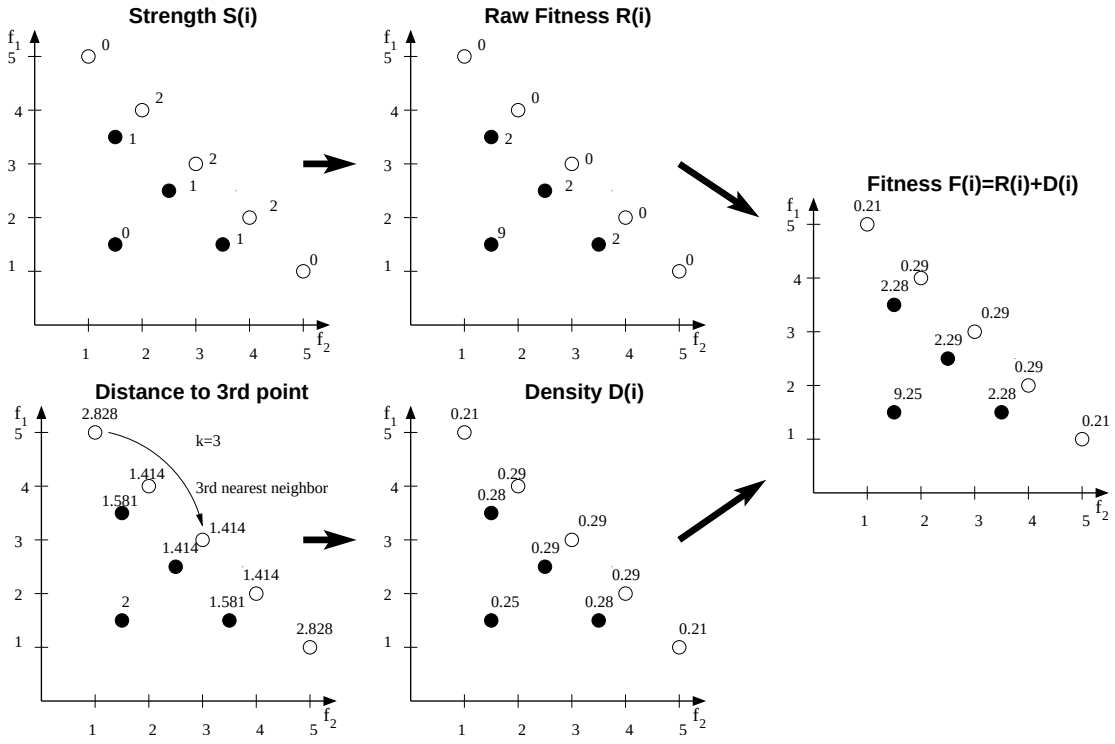

Figure 26: SPEAII Fitness Examples

Note that the density will always be between 0 and 1/2.

The following listing details the SPEAII density algorithm with six comparison locations: A through F.

```
// Calculate k (assume P is current and elite population)
k=sqrt(|P|);
// Loop through each individual in the population
for (i in P) {                                          // Comparison A
    // Loop through all other individuals in the population
    // Calculating distance
    iCount=0;
    for (j in P, j!=i) {                                // Comparison B
        // Initialize elements
        iCount++;
        d[iCount] = 0.0;
        // Loop through all objectives
        for (m=1; m<=q; m++) {                          // Comparison C
            d[iCount] += (i.f[k] - j.f[k])^2;
        } // End of loop through objectives
        d[iCount] = sqrt(d[iCount]);
```

```
    } // End of loop through other individuals, j
    // Sort list based on distance
    for (m=1; m<|P|; m++) {                          // Comparison D
        dMin = d[m];
        iMin = m;
        // Search all individual above present for minimum
        for (n=m+1; n<=|P|; n++) {                   // Comparison E
            if (d[n] < dMin) {                       // Comparison F
              dMin = d[n];
              iMin = n;
            }
        }
        // Swap values
        d[iMin] = d[m];
        d[m] = dMin;
    }
    // Calculate Density
    i.D = 1.0 / (d[k] + 2.0);
} // End of loop through individuals, i
```

This algorithm implements Equations 88 and 89 for the calculation of the density. From this listing, the six comparison locations, A through F, result in the number of comparisons as:

$$\begin{aligned}
\text{(from line 4) } N_A &= |P| \\
\text{(from line 8) } N_B &= |P|(|P|-1) = |P|^2 - |P| \\
\text{(from line 13) } N_C &= qN_B \\
\text{(from line 19) } N_D &= |P|(|P|-1| = N_B \\
\text{(from line 23) } N_E &= |P| \sum_{j=1}^{|P|-1} j = |P| \frac{(|P|-1)|P|}{2} \\
\text{(from line 24) } N_F &= N_E \\
N_{SPEAII,Density} &= N_A + N_B + N_C + N_D + N_E + N_F \\
N_{SPEAII,Density} &= |P|^3 + (1+q)|P|^2 - (1+q)|P| \qquad (90)
\end{aligned}$$

The final fitness value is then defined as:

$$F(i) = R(i) + D(i). \qquad (91)$$

The number of calculations for Equation 91 becomes the sum of Equations 87 and 90.

$$\begin{aligned} N_{SPEAII} &= N_{SPEAII,Raw} + N_{SPEAII,Density} \\ &= (4+4q)|P|^2 - (2+4q)|P| + |P|^3 + (1+q)|P|^2 - (1+q)|P| \\ N_{SPEAII} &= |P|^3 + (5+5q)|P|^2 - (3+5q)|P| \end{aligned} \quad (92)$$

2.2.7 Complexity Comparisons

Table 1 accumulates the results of the previous six sub-sections into a single location for comparison purposes. The number of elite individuals is $|E|$. The number of individuals in the total population is $|P|$. The number of individuals in the current population is $|C| = |P| - |E|$. The number of objectives is q.

Table 1: Number of Comparisons for Various MOEA Algorithms

Algorithm	**Number of Comparisons**
Weighted Sum	$(1+q)\|P\|$
VEGA	$2\|P\|$
Elite Preserve	$(2+2q)\|P\|^2 + \frac{\|P\|\|C\|}{2}(q+5) + q\frac{\|C\|}{2} - (5+2q)\|P\|$
NPGA	$(4+3q)\|P\|^2 + q\|P\| - (1+2q)$
SPEA	$+\frac{1}{30}(1+q)\|P\|^5 + \left(\frac{1}{12}q + \frac{1}{6}\right)\|P\|^4 + \left(\frac{5}{6}\right)\|P\|^3 + \left(1\frac{11}{12}q + 2\frac{1}{3}\right)\|P\|^2$ $+\left(1\frac{1}{30}q + 4\frac{19}{30}\right)\|P\| - 2q$ $+\left(-\frac{1}{5}(1+q)\right)\|E\|^5 + \left(-\frac{3}{4}q - \frac{1}{2}\right)\|E\|^4 + \left(-\frac{5}{6}q - \frac{2}{3}\right)\|E\|^3$ $+\left(-4\frac{1}{4}q - 5\right)\|E\|^2 + \left(\frac{1}{30}q + \frac{11}{30}\right)\|E\|$ $+\|P\|^2\left(-\frac{1}{3}(1+q)\|E\|^3 - (1+q)\|E\|^2 - \frac{2}{3}(1+q)\|E\|\right)$ $+\|P\|\left(\frac{1}{2}(1+q)\|E\|^4 + \left(\frac{5}{3}q + \frac{4}{3}\right)\|E\|^3 + \left(\frac{3}{2}q + \frac{1}{2}\right)\|E\|^2 + \left(4\frac{1}{3}q + 3\frac{2}{3}\right)\|E\|\right)$
SPEAII	$\|P\|^3 + (5+5q)\|P\|^2 - (3+5q)\|P\|$

Figure 27 illustrates the dependencies of the equations in Table 1 on the number of objectives q, by holding the sizes of the populations constant, ($E = 100, C = 500, P = 600$), and varying the number of objectives from 1 to 25. Because VEGA simply chooses a single objective, it does not have a dependence on the number of objectives. Compared to the remaining algorithms, the SPEAII method illustrates only a slight dependency on q. This can be seen in the equation as well with the dependency on $|P|^2$ term instead of the $|P|^3$

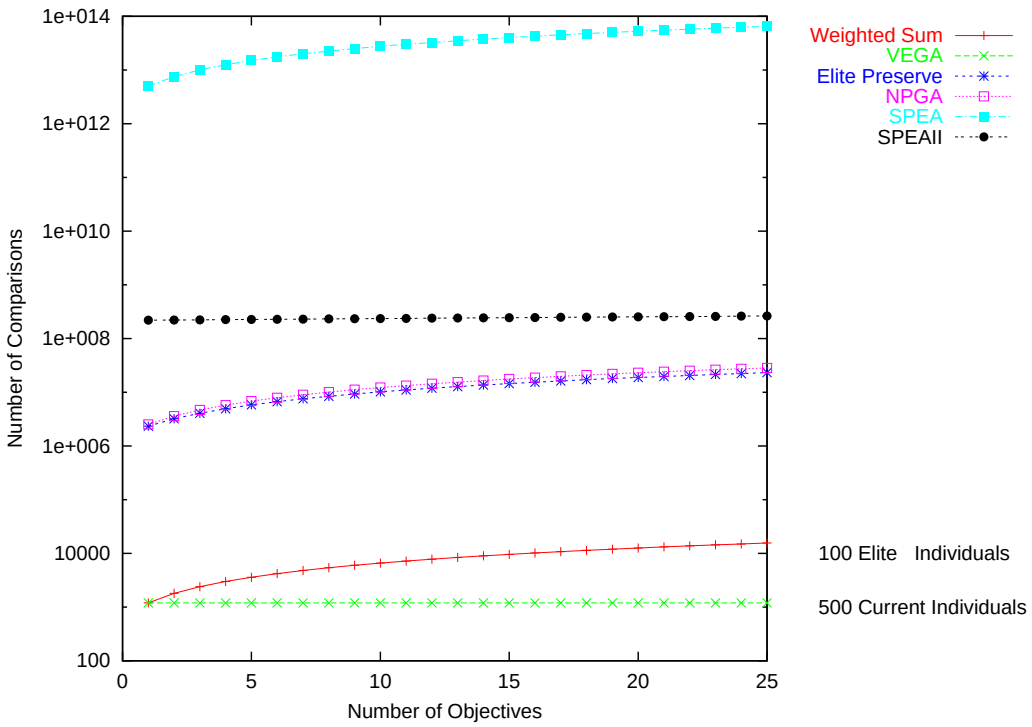

Figure 27: Effects of the Number of Objectives on the Number of Comparisons for Various MOEAs

term.

Also illustrated in Figure 27 is the overall complexity of the algorithms with a representative population size. The weighted-sum and VEGA methods, with their dependency on $|P|$, require 10^3 to 10^4 comparisons. The NPGA and elite preserve methods are very close in complexity with their dependency on $|P|^2$, and thus require 10^6 to 10^8 comparisons. The SPEAII method has a dependency on $|P|^3$ and requires 10^8 to 10^9 comparisons. But, the SPEA algorithm, with its dependency on $|P|^5$, shows the highest complexity, and thus is the slowest running, requiring 10^{12} to 10^{14} comparisons.

To get a feel for these changes in magnitude, let the comparisons of a weighted-sum or VEGA method, requiring 10^3 to 10^4 comparisons, take 1 to 10 milliseconds to evaluate. Then, the NPGA and elite preserve methods would require 1 to 100 seconds. The SPEAII method would require 100 to $1,000$ seconds, or approximately 2 to 16 minutes. The SPEA algorithm would then require 277 to $27,777$ hours.

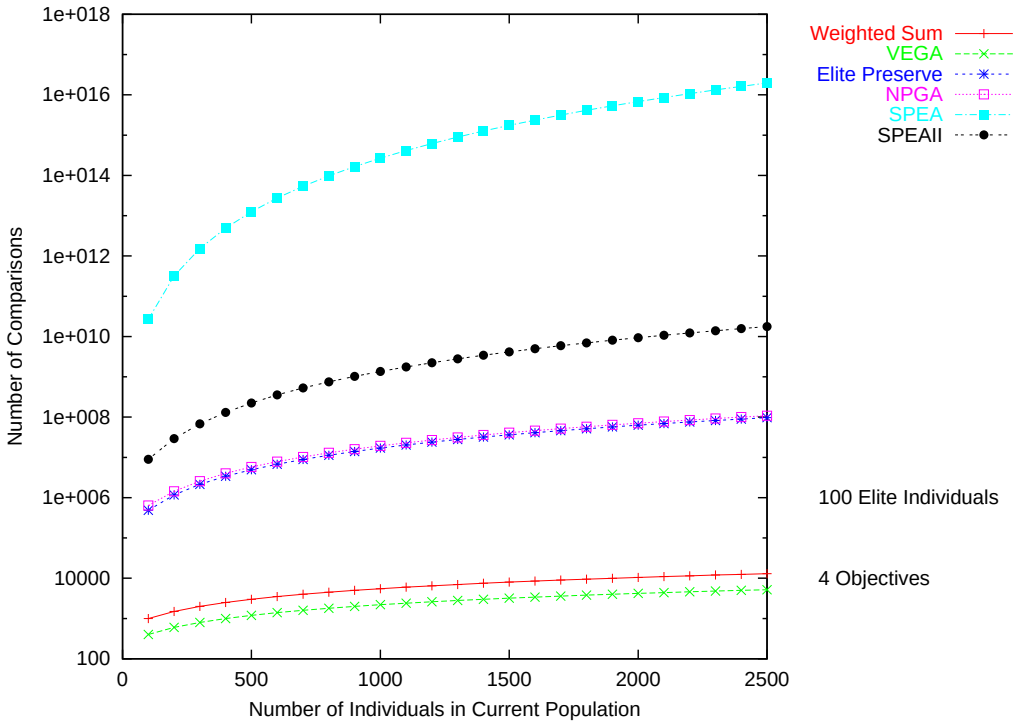

Figure 28: Effects of the Number of Individuals in Current Population on the Number of Comparisons for Various MOEAs

Figure 28 further illustrates the dependencies on $|P|$ by holding the number of objectives and number of elite individuals constant, ($q = 4$ and $|E| = 100$), and varying the number of individuals in the current population from 100 to $2,500$ in increments of 100. Note that the SPEA method suffers the largest increase in complexity as more individuals are added.

Figure 29 illustrates that the size of the elite pool can not be increased without also increasing the number of comparisons. In this case, the size of the current pool is set to 500 and the number of objectives is set a value of 4. The number of elite individuals is varied from 100 to $1,000$ in increments of 100.

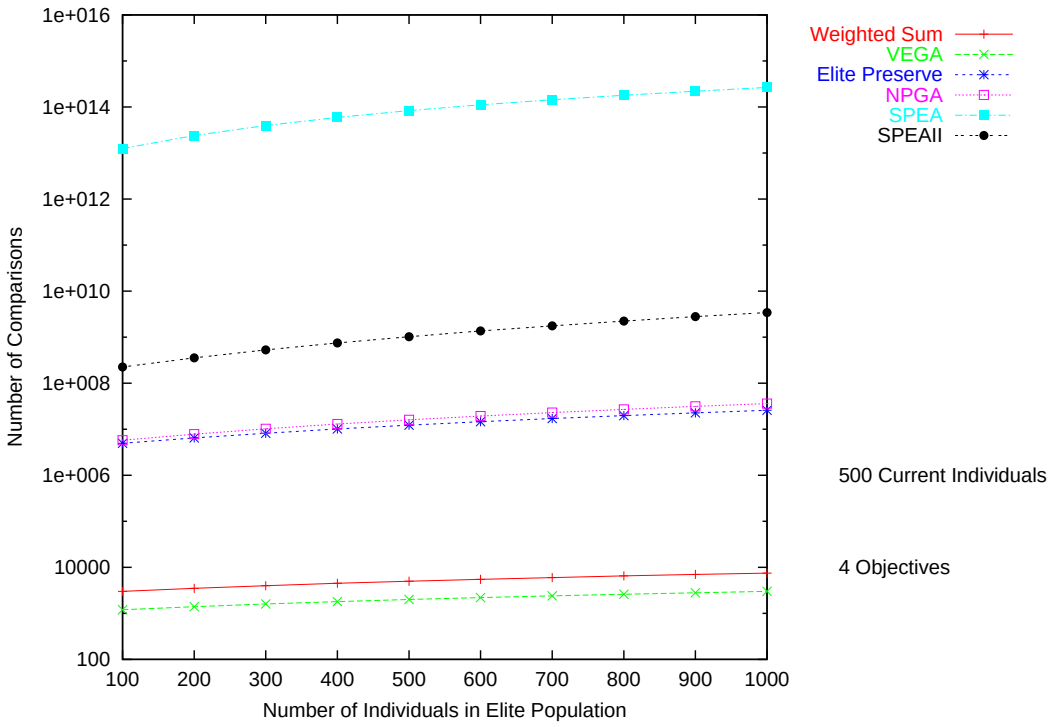

Figure 29: Effects of the Number of Individuals in Elite Population on the Number of Comparisons for Various MOEAs

Table 2 details the number of comparisons for various components of the MOEA methods. Because the research in Chapter 4 allows for the mixing of various algorithm components, this table allows insight into the complexity trade-offs of these components.

Table 2: Number of the Comparisons for Various MOEA Components

Algorithm	**Number of Comparisons**
Pareto Optimality	$(2+2q)\|P\|^2-(1+2q)\|P\|$
Pareto Rank 1	$(2+2q)\|P\|^2-(1+2q)\|P\|$
Pareto Rank 2	$\frac{2+2q}{3}\|P\|^3+\|P\|^2+\frac{10-2q}{3}\|P\|$
Fitness Sharing	$(2+q)\|P\|^2-(1+q)\|P\|$
Objective Scaling	$(2+4q)\|P\|-(1+2q)$
SPEA Clustering	$\frac{1}{30}(1+q)\|P\|^5+\left(\frac{1}{12}q+\frac{1}{6}\right)\|P\|^4+\left(\frac{5}{6}\right)\|P\|^3+\left(-\frac{1}{12}q+\frac{1}{3}\right)\|P\|^2$ $+\left(\frac{29}{30}q+4\frac{19}{30}\right)\|P\|-2q$ $+\left(-\frac{1}{5}(1+q)\right)\|E\|^5+\left(-\frac{3}{4}q-\frac{1}{2}\right)\|E\|^4+\left(-\frac{5}{6}q-\frac{2}{3}\right)\|E\|^3$ $+\left(-\frac{1}{4}q-1\right)\|E\|^2+\left(\frac{1}{30}q+\frac{11}{30}\right)\|E\|$ $+\|P\|^2\left(-\frac{1}{3}(1+q)\|E\|^3-(1+q)\|E\|^2-\frac{2}{3}(1+q)\|E\|\right)$ $+\|P\|\left(\frac{1}{2}(1+q)\|E\|^4+\left(\frac{5}{3}q+\frac{4}{3}\right)\|E\|^3+\left(\frac{3}{2}q+\frac{1}{2}\right)\|E\|^2+\left(\frac{1}{3}q-\frac{1}{3}\right)\|E\|\right)$
SPEA Strength	$-(2+2q)\|E\|^2+\|E\|+\|P\|(2+2q)\|E\|$
SPEA Fitness	$\|P\|-(2+2q)\|E\|^2-\|E\|+\|P\|(2+2q)\|E\|$
SPEAII Raw Fitness	$(4+4q)\|P\|^2-(2+4q)\|P\|$
SPEAII Density	$\|P\|^3+(1+q)\|P\|^2-(1+q)\|P\|$

Figure 30 holds the population sizes fixed, ($|E| = 100|$, and $|C| = |500|$), and varies the number of objectives from 1 to 25. Note that the SPEAII density algorithm shows the least dependency to the number objectives. Also note the banding of the algorithm components into four groups. First, the objective scaling algorithm is only dependent on $|P|$, thus requiring only 10^3 to 10^5 comparisons. Second, the Pareto Optimality, Pareto Rank 1, fitness sharing, SPEA strength, SPEA fitness and SPEAII fitness methods are all dependent $|P|^2$, requiring 10^6 to 10^8 comparisons. Third, the SPEAII density and Pareto Rank 2 methods are dependent on $|P|^3$, requiring 10^8 to 10^{10} comparisons. Finally, the SPEA clustering method is dependent on $|P|^5$, requiring 10^{12} to 10^{14} comparisons.

An obvious question is, "Why does the number of comparisons required for the SPEA clustering method not render the SPEA method useless?". During the testing performed for

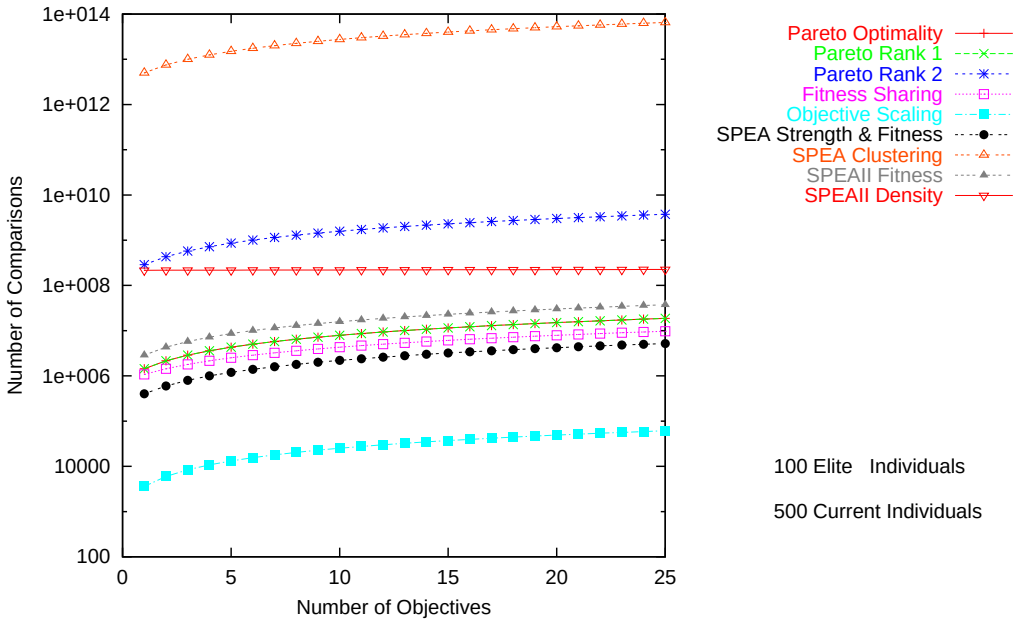

Figure 30: Effects of the Number of Objectives on the Number of Comparisons for Various MOEA Components

evaluation, (see Chapter 4), it was noted that the clustering algorithm was rarely required because the number of Pareto optimal individuals found rarely exceeded the number maximum size of the elite pool, $|E|$. Therefore, only the times required for Pareto optimality, SPEA fitness and SPEA strength calculations were required, which only require an order of $|P|^2$ comparisons. Although this observation is true for low dimension problems such as the 0/1 knapsack problem, it is not true for problems with a large number of objective dimensions. For these problems, many individuals are Pareto optimal due to the increased size of the Pareto front.

Figure 31 further illustrates the dependencies on $|P|$ by holding the number of objectives and number of elite individuals constant, ($q = 4$ and $|E| = 100$), and varying the number of individuals in the current population from 100 to $2,500$ in increments of 100.

Figure 32 illustrates the dependencies on the $|E|$ by holding the size of the current pool to 500 and the number of objectives to a value of 4. The number of elite individuals is then

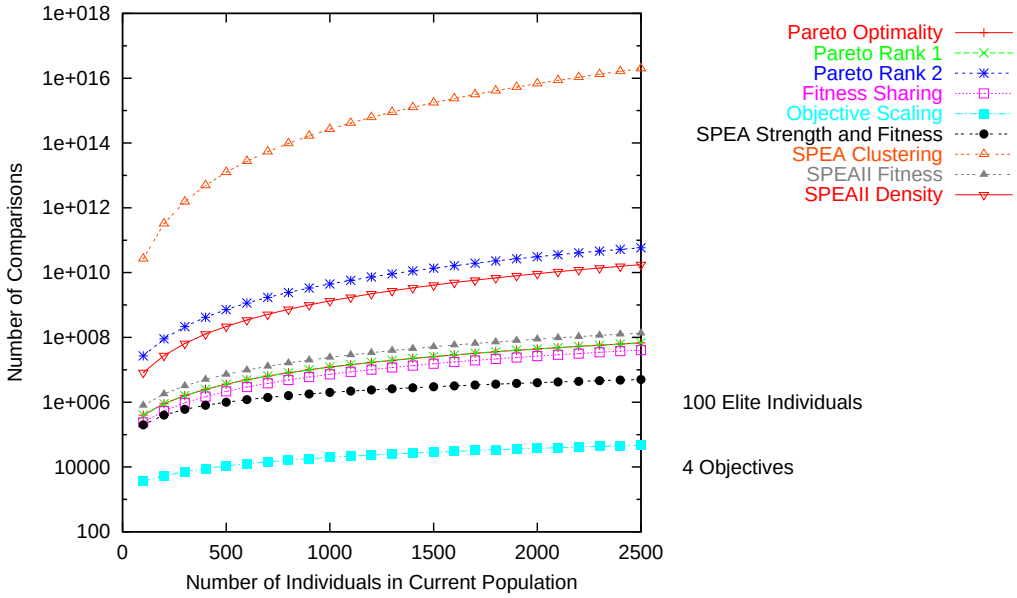

Figure 31: Effects of the Number of Individuals in Current Population on the Number of Comparisons for Various MOEA Components

varied from 100 to 1,000 in increments of 100.

2.3 Conclusions

The overall objective of this thesis is to decrease the time required to evolve solutions that reside within a region of interest in the objective space for independent computationally expensive objectives. Although this chapter is intended to provide an overview of existing EA and MOEA methods, this chapter makes several important analyses that will be useful in the next chapter in design of new algorithms for independent computationally expensive objectives.

There are four basic components of MOEA algorithms: fitness evaluation, selection, mating and mutation. MOEA methods can only improve their performance further by improving the selection of parents from previous generation for creation of the next generation of children. This selection is effected by two processes: the fitness evaluation, and

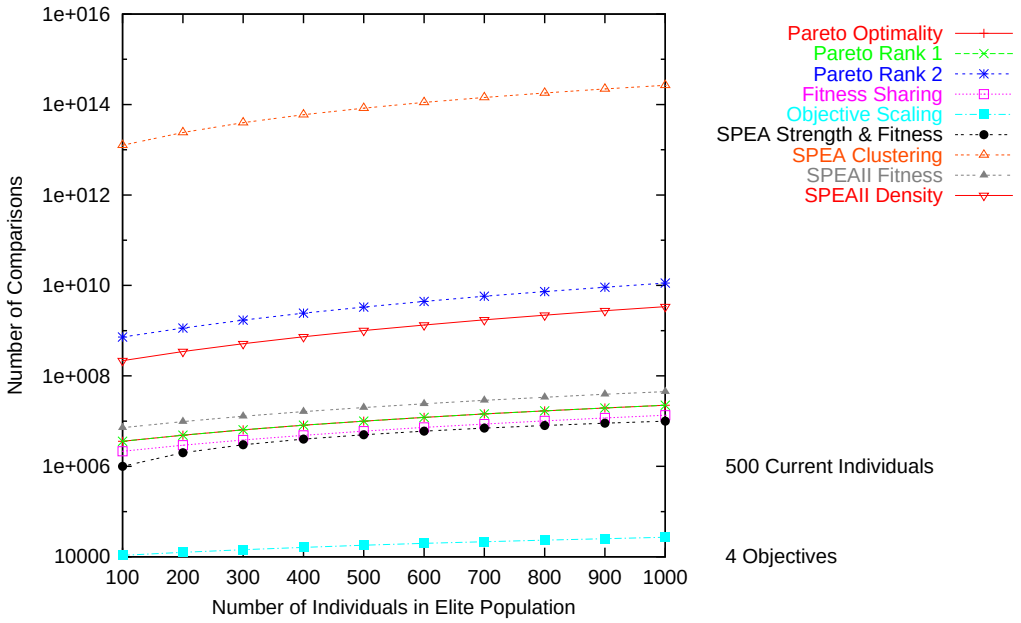

Figure 32: Effects of the Number of Individuals in Elite Population on the Number of Comparisons for Various MOEA Components

the selection method.

To fully understand the selection process, Section 2.1.2 provided an analysis of the probability of selection for various selection methods that has not been found in the literature. From this analysis several important conclusions were reached.

First, Section 2.1.2.2 provided a detailed analysis of the probability of selection for the tournament selection method. During this section it was shown that for binary tournament selection without replacement, the probability of selection for the 2^{nd} to $|P|^{th}$ individual approached $\frac{1}{n-1}$ as $|C|$ approached $|P|-1$, resulting in the loss of the influence on the fitness values on the probability of selection. Therefore, *binary tournament selection without replacement is not recommended*.

Second, the probability of selection is always dependent on the fitness value. But, different fitness functions provide different distributions of fitness values. Therefore, the coupling of these distributions with the selection methods can not be ignored. Because the

tournament selection method is dependent on the ranking of individuals, *the tournament selection method can be used with any fitness function*. Fitness proportionate selection is very dependent on the distribution fitness values. Therefore, as detailed in Section 2.1.2.3, *fitness proportionate selection should not be used with fitness functions that do not provide a limited range of fitness values*. Of the EA fitness methods analyzed, the weighted-sum is the most likely to contain a high variation in fitness values and should therefore use tournament selection if the range of fitness values is not well distributed. Pareto rank techniques can use either fitness proportionate or tournament selection.

Two methods for crossover were described: N-Point and cloning. The important issue for N-Point crossover is the avoidance of mutation effects by the crossover operator due to crossover in the middle of an attribute value. The next chapter will address additional methods to avoid this and other mutation effects.

Three methods for mutation were investigated: bitwise, uniform, and normal. Although bitwise mutation is commonly discussed in the literature, it is most commonly used with simple bit attributes. As problems become more complex and attributes must take on more complex descriptions, such as IEEE float point representations, the mutation methods result in undesired effects. Therefore, since uniform mutation provides the functionality of bit-wise mutation for simple bit representations and does not have undesired consequences for more complex attribute descriptions, *uniform mutation is recommended as a replacement for bitwise mutation*.

Section 2.1.6 discussed parallel evaluation of fitness evaluations using either a single gene pool or multiple gene pool method. Theoretical equations for the optimal number of processors from Cantu-Paz and Goldberg [10] were examined and challenged. The argument was made that due to synchronization times, *for problems with complex fitness evaluations and complex processor configurations, the optimal number of processors can not be found in a closed form solution*. Although this Chapter discredited the current thoughts on the optimal number of processors, the next chapter will provide a solution that removes the

need for synchronization and thus allows a more scalable solution.

Section 2.2 provided an analysis of the weighted-sum, VEGA, elite preserve, NPGA, SPEA, and SPEAII algorithms. For each of these algorithms two artifacts were created: a pseudo-code listing, and an evaluation of the number of comparisons required by each algorithm. Analysis also broke the algorithms apart into smaller components that can and will be reused and recombined in novel ways in future chapters. From this analysis it was determined that since the Pareto rank method 2 is of order $|P|^3$ and Pareto rank method 1 is of order $|P|^2$ *without a compelling improvement in the performance of a MOEA with PR2 versus PR1 fitness, PR1 should be selected due to the additional complexity added by the PR2 method.* Also of importance is that the SPEA clustering algorithms has the poorest performance with an order of $|P|^5$ comparisons. Although clustering is rarely required with simple problems, *SPEA clustering could cause severe run-time problems with complex problems with many objectives*.

CHAPTER III

RESEARCH CONTRIBUTIONS

Chapter 1 introduced the definitions of single objective optimization, multiple objective optimization problems, search space, objective space, Pareto optimality, and a region of interest in the objective space. Chapter 2 then detailed EA operators of fitness evaluation, selection, mating and mutation, as well as the basics of genome encoding and parallel evaluation. Chapter 2 also presented several MOEA methods with a particular emphasis placed on the complexity of each component of the methods.

Given the context of the previous two chapters, this Chapter is able to detail the research contributions of this thesis. The overall thrust of this research is to decrease the time required to find solutions that reside within a region of interest in the objective space for independent computationally expensive objectives. Although all changes were made for this specific class of problems, some of the research contributions are applicable to more general problems. The research contributions can be divided into four categories.

First, several contributions that are applicable to all EA problems, regardless of whether the problem is computationally expensive or contains multiple objectives, are outlined in Section 3.1. These contributions include improvements to genome encoding that allows a hierarchical description of the genome. The hierarchical description then allows association of the genome component names with crossover and mutation operators. Also fitting this category of contributions is a transform scaler that allows transformation of fitness values to different ranges based on a set of linear interpolated data. The final contribution in this category is a modular design and implementation that allows both implementation of existing MOEA methods and reconfiguration of these algorithms for new and novel combinations.

The second category of contributions given in Section 3.2 are those applicable to computationally expensive (CE) objectives that may or may not be independent. Included is the ability to interface to external objective evaluation functions. Interfacing to external objective evaluators allows existing objective evaluation simulation to quickly interface to the MOEA software. In practice this has been done by wrapping legacy simulations to translate the search space location into proper inputs for the simulation and extract objective values from the program output for transmittal back to MOEA software. Also falling into the category of computationally expensive objective contributions is a hybrid parallel evaluation mechanism. This method obviates the timing problems of synchronization outlined in the previous chapter by not forcing all individuals in a generation to be evaluated before the next generation is created. This solution also decouples the number of populations from the number of processors and still allows transfer of genetic material between populations.

The third category of contributions requires knowledge of the region of interest (ROI) in objective space to improve performance, but does not require objectives to be independent where independent implies that multiple objectives must be evaluated in separate evaluations. This category of contributions includes the hypercube distance scaling algorithm that scales fitness values by the distance to the hypercube defined by the region of interest. This scaling gives a higher probability of mating to those individuals that reside closest to the region of interest.

The fourth and final category of contributions requires both knowledge of the region of interest as well as independence of objectives to improve performance. These contributions, outlined in Section 3.4, include dynamic objective thresholding and dynamic objective scaling. These two contributions work together to improve performance by identifying those individuals that are not likely to improve performance after the evaluation of only a few objectives and by preventing evaluation of the remainder of the objectives.

Section 3.5 completes this chapter with the conclusions reached during this chapter.

3.1 Research Contributions for All Multiple Objective Evolutionary Algorithms

This section details enhancements that do not require problems to be either computationally expensive or independent to allow for improvement. A method that allows the hierarchical description of a genome was developed. This hierarchical description then allows the association of parts of the genome to the crossover and mutation operators. Additionally, a method for translating fitness values into different ranges using linear interpolated data was developed. Finally, the advancements and possibilities of the modular design and implementation are examined.

3.1.1 Genome Encoding

Ultimately, a genome is encoded as a string of attributes. The set of all feasible strings of attributes defines the search space. Evolutionary operators such as crossover and mutation then perform operations on these strings of attributes. There are two motivations for the development of a generic genome description. First, a description of the genome that can be modified external to the software allows tailoring to new domains without recompilation of the software. Second, providing a generic system description hides the underlying complexity of the evolutionary operators from the end-user, e.g., the end-user does not need to worry about how many bits are required to encode various attributes or how they are stored or communicated.

The most general search-space feasibility region for a MOP is:

$$\begin{aligned} \boldsymbol{x} \quad &= (x_1, x_2, \cdots, x_n) \in \boldsymbol{X} \subset \mathbb{R}^n \\ \text{Subject to} \quad &\boldsymbol{e(x)} = (e_1(\boldsymbol{x}), e_2(\boldsymbol{x}), \cdots, e_m(\boldsymbol{x})) \leq \boldsymbol{0}. \end{aligned}$$

The m constraints $\boldsymbol{e(x)}$ allow the specification of any region within the space $\mathbb{R}^q$. The

implementation of this thesis does not allow this full flexibility. Instead the abstract description is reduced to a set of n ranges, one for each attribute in the search space.

$$\begin{aligned} & \boldsymbol{x} && = (x_1, x_2, \cdots, x_n) \in \boldsymbol{X} \subset \mathbb{R}^n \\ \text{Subject to} \quad & \boldsymbol{e(x)} && = (e_1(x_1), e_2(x_2), \cdots, e_n(x_n)) \leq \boldsymbol{0} \\ \text{Where for } k = 1 \text{ to } n \quad & e_k(x_k) && = \begin{cases} 1 & x_k \in [x_{k,min}, x_{k,max}] \\ 0 & otherwise \end{cases} \end{aligned}$$

The implementation developed for this research provides a generic XML description of a genome as a hierarchy or inverted tree structure of each of the possible attribute types. The syntax of an XML file is controlled by the document type description (DTD). Use of a DTD forces the search space description to be syntactically correct before processing begins. But as with any language, being syntactically correct does not guarantee communication of the desired semantics. It is up to the user to be semantically correct. Other methods such as XML schema that allow the specification of the type and bounds of entries beyond the *parsed character data* of the DTD may have obviated the need for developing this specialized DTD. These other methods were not used due to their lack of maturity when software development began.

The following listing itemizes the DTD for the search space.

```
<!ELEMENT Genome    (Name,(Define|Instance|Genome|Case|↓
→     Value|Double|Enumerate)+)>
<!ELEMENT Define    (Name,(Define|Instance|Genome|Case|↓
→     Value|Double|Enumerate)+)>
<!ELEMENT Instance  (#PCDATA)>
<!ELEMENT Case      (Name,(Define|Instance|Genome|Case|↓
→     Value|Double|Enumerate)+)>
<!ELEMENT Value     (Name, Start, End, Delta)>
<!ELEMENT Double    (Name, Start?, End?)>
<!ELEMENT Enumerate (Name,Map*)>
<!ELEMENT Name      (#PCDATA)>
<!ELEMENT Map       (#PCDATA)>
<!ELEMENT Start     (#PCDATA)>
<!ELEMENT End       (#PCDATA)>
```

```
<!ELEMENT Delta      (#PCDATA)>
```

The following listing provides an example of a syntactically correct XML search space description.

```
<?xml version="1.0"?>
<!DOCTYPE Genome SYSTEM "gene.dtd" [ ]>
<!-- This example search space contains 3 attributes -->
<Genome>
        <Name>SearchSpace</Name>
        <Double>
                <Name>Attribute1</Name>
                <Start>0.0</Start>
                <End>100.0</End>
        </Double>
        <Genome>
                <Name>SubComponent</Name>
                <Value>
                        <Name>Attribute2</Name>
                        <Start>0.0</Start>
                        <End>10.0</End>
                        <Delta>0.1</Delta>
                </Value>
                <Enumerate>
                        <Name>Attribute3</Name>
                        <Map>On</Map>
                        <Map>Off</Map>
                        <Map>Pause</Map>
                        <Map>Record</Map>
                        <Map>Play</Map>
                </Enumerate>
        </Genome>
</Genome>
```

The software translates the search space description into a hierarchical description for a specific individual. The following listing provides an example of one of the many individuals that meets the search space description given above.

```
<SearchSpace>
        <Attribute1>33.4</Attribute1>
        <SubComponent>
```

```
                    <Attribute2>9.1</Attribute2>
                    <Attribute3>Pause</Attribute3>
            </SubComponent>
    </SearchSpace>
```

Referring to the DTD in listing above, note that the leaves of the inverted tree structure must be an attribute description. Attribute descriptions of types *Value*, *Double*, and *Enumerate* are supported. Each attribute description contains a *Name* value that becomes the tag used to identify the attribute to the fitness evaluator as seen in the example given above.

The *Value* description is used to describe sampled values. To describe sampled values the *Start*, *End* and *Delta* elements are used to describe the encoding of the value. The number of bits required to encode a *Value* attribute is:

$$b = \lceil log_2 \left(\frac{\text{Start} - \text{End}}{\text{Delta}} + 1 \right) \rceil \tag{93}$$

A value V can then be converted to the bit encoding V_b in the range $[0, 2^b - 1]$ using:

$$V_b = \lceil \frac{V - \text{Start}}{\text{Delta}} + 0.5 \rceil \tag{94}$$

Likewise, the value of V can be extracted from the bit encoding using:

$$V = V_b \cdot \text{Delta} + \text{Start} \tag{95}$$

The definition of the *Start* and *End* boundaries defines the feasibility region for this attribute. The boundaries also allow the software to verify that attribute values are in the feasibility region during each EA operation, i.e., genome reading, crossover, and mutation.

The *Double* leaf node encodes variables in the internal 64 bit IEEE double precision encoding. Each *Double* node contains *Start* and *End* elements. Note that unlike the *Value* node, there is not a *Delta* element. Thus, encoding of information is only allowed at the highest resolution provided by the IEEE standard. Also note that the *Start* and *End* elements are optional, allowing, but not forcing the user to define the feasibility region of

the attribute. If the *Start* and *End* boundaries are not specified, the limits are those of the IEEE double precision encoding. Unlike the *Value* element, the encoding requires 64 bits regardless of the boundaries. But similar to the *Value* element, the boundaries for a *Double* node allow the software to verify that attribute values are in the feasibility region after each EA operator.

The *Enumerate* node allows the mapping of a list of names to an attribute. In addition to the *Name* node, the user must specify a list of *Map* nodes that contains the list of possible strings to be enumerated. This simple definition allows the specification of several ASCII strings to be associated as possible values for a search dimension. Let n be the number of *Map* entries, then the number of bits required to encode the attribute is $b = log_2(n)$. Since n may not be a power of 2, there are bit instances in the range of the 2^b bits, $[0, 2^b - 1]$, that are not in the range of the list of strings, $[0, n]$. Therefore, like the *Value* and *Double* attributes, the software must verify that attribute values are in the feasibility region after each EA operator.

As illustrated in genome DTD and the genome example listed above, the *Genome* node allows wrapping a set of attributes, or other Genome nodes, to create higher level components. Each *Genome* has an associated *Name* that is used for the XML system description and can be referenced by the EA operators. Figure 33 illustrates the hierarchy and bit encoding for example genome listed above. The ability to organize a system description in a hierarchy allows a system description that better matches a system as a set of sub-systems. For example in the flare pattern cases examined in Chapter 5, a flare pattern is made up of a set of flares, each with three attributes: the ejection time, ejector location on the aircraft, and type of flare ejected. The warning receiver optimization examined in Chapter 6 likewise divides up the list of attributes based on the part of the overall warning receiver affected, e.g., source processing, or track processing. But the best example of use of hierarchy is in the evolutionary programming techniques found in Chapter 7. In this chapter a system is developed that allows the modification of the attributes of a sub-system and

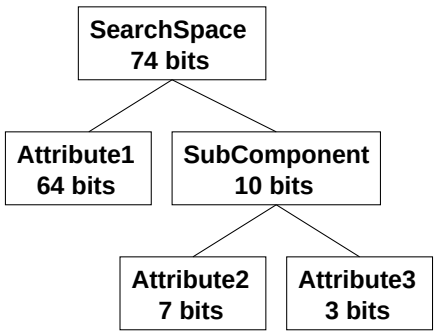

Figure 33: Example Search Space Hierarchy

the sub-system interconnectivity, which is in turn translated into a Ptolemy II hierarchical description.

In practice, various portions of the hierarchy may need duplicated. To meet this need the concept of *Define* and *Instance* nodes is included to allow multiple instantiation of components at any level. Again using the flare pattern design problem of Chapter 5 as an example, a flare can be defined as a sub-system with its three attributes of ejection time, ejector location and type of flare. The flare is then instantiated as many times as required by the search space.

The final feature in the search space definition is the need for conditional replacement. This support is allowed using *Case* nodes. These nodes allow for different dimensions of the search space to be conditionally switched in and out. This allows specification of very complex modal search spaces. An example of the use of this capability is found in the evolutionary programming problem of Chapter 7, where each sub-system description in the genome contains descriptions for each of the possible sub-systems. But, only one of the sub-system descriptions is expressed for each instance of the system.

3.1.2 Location Crossover

The location crossover method was developed to eliminate the mutation effects of the N-Point crossover method when dealing with attributes that are not a single bit, as detailed in Equation 17 of Section 2.1.3.2. Automated methods for finding the best crossover locations

have been developed by Van Veldhuizen [57] in order to find the *building blocks* of the underlying genome. Unfortunately, these methods require the initial evaluation of many individuals to determine these locations. Because this research is especially targeted for computationally expensive objectives, it was decided to use the domain knowledge of the user instead.

The location crossover method allows the specification of the exact locations for crossover. As described in the previous section, the search space description allows the building up of a hierarchy of attributes. The search space description also assigns a mnemonic to each component of the hierarchy, which defines the search space dimensions as well as the hierarchy components. These names can be supplied to the location crossover as locations for crossover. This allows the user to allow crossover at any single attribute value or to limit crossover to higher level components within the gene description.

Just as the argument was made in Section 2.1.3.2 that crossover in the middle of an attribute results in a mutation of the system, mutation in the middle of a system description may also be deleterious. For example, a higher level component may combine two double precision attributes together as a complex number, or as an x, y location. In either case, combining the real value from one parent and the imaginary value from a second parent, or the x value from one parent and the y value from a second parent, could and most likely will result in poor performing children.

The location crossover method allows the user to use domain knowledge to identify the locations within the genome that should not result in a mutation of the genome if split at this location. Using the names of the elements in the hierarchical description, the location crossover method identifies the locations in the genome for crossover. It then fills each resulting section of the genome by randomly choosing a parent from the list of selected parents.

3.1.3 Mutation

Early evolutionary algorithms only discussed the use of the bitwise mutation, which at its best results in a uniform distribution. Fortunately, others such as Obayashi, Takahashi and Takeguchi [46, 47] discussed the use of normal and uniform mutation methods. Uniform distribution is probably best used for enumerated variables, as any possible value should be equally important. For sampled and double precision variables though, a normal distribution makes more sense as it performs a small transition of the variable. The selection of normal mutation still requires the selection of the standard deviation. But often, different variables should have different standard deviations.

This research contribution allows the specification of multiple mutation operators each with a type of bitwise, normal, or uniform. Also, much as with the location crossover operator, this implementation allows the association of normal and uniform mutation operator with a list of mnemonics from the genome hierarchy. Thus, an attribute can be associated with multiple mutation operators, e.g., a single uniform distribution and multiple normal distributions with different standard deviations and different mutation rates.

Again, as with the location operator, this allows the use of domain knowledge by the end user to tailor the operators for a specific problem. The capability was most important with the tuning of the AAR-44A algorithms, (see Section 6.2), where uniform mutations were very deleterious, and thus a different normal mutation operator was associated with each variable.

3.1.4 Transform Scaler

The transform fitness scaler method allows the specification of a set of points for mapping solutions as desired using linear interpolation. Use of the linear interpolated data (LID) allows a single implementation for any number of transformation ideas.

The *Transform* node specification simply contains a list of *Point* nodes. Each *Point* node contains an *In* element and *Out* element that contains the input and output values

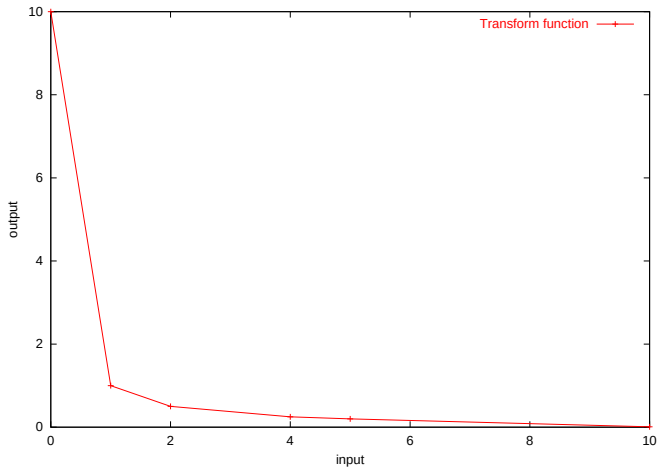

Figure 34: Example Transform Fitness Scaler

respectively. The list of points must contain monotonic increasing values for the input.

As the example in Figure 34 illustrates, the transform can translate a minimizing fitness function into a maximizing fitness function. Thus the output from a minimizing fitness function such as SPEA and Pareto fitness functions, where the *best* solutions have low fitness values, can be translated into maximizing fitness function. The fitness value can then be used to determine selection using maximizing techniques, where the individuals with the highest probability of selection have the highest fitness values.

The following listing details this algorithm and five comparison locations A through E.

```
// Loop through each individual in the population
for (i in P) {                                          // Comparison A
    // Start at beginning of list
    point1 = pointList[1]
    point2 = pointList[2]
    iPoint = 2;
    bFound = false;
    // Check for point before input data starts
    if (i.F < point1.in) {                              // Comparison B
        i.Fnew = point1.out;
        bFound = true;
    }
    // Loop until point is found
    while (!bFound) {                                   // Comparison C
        // Assign fitness if between point1 and point2
```

```
        if ( (i.F >= point1.in) && (i.F < point2.in) ) {
17                                                      // Comparison D
            i.Fnew = point1.out +
                     point1.slope * (i.F - point1.in)
20          bFound = true;
        // Advance to the next interval if not found
        } else {
            point1 = point2
            iPoint++
25          // Make sure we have not passed the end of data
            if (iPoint > iNumPoints) {          // Comparison E
               i.Fnew = point1.out;
               bFound = true;
            }
30          point2 = pointList(iPoint);
        }
    } // End of loop through points
} // End of loop through population
```

Allowing $|LID|$ to be the number of points in the linear interpolated data transformation list, the total number of comparisons is:

$$\begin{aligned}
\text{(from line 2) } N_A &= |P| \\
\text{(from line 9) } N_B &= |P| = N_A \\
\text{(from line 14) } N_C &= (|LID| - 1) * |P| \\
\text{(from line 16) } N_D &= N_C \\
\text{(from line 26) } N_E &= N_C \\
N_{Transform} &= N_A + N_B + N_C + N_D + N_E \\
N_{Transform} &= (2 + 3(|LID| - 1))\,|P|. \qquad (96)
\end{aligned}$$

3.1.5 Modular Multiple Objective Evolutionary Algorithm Framework

Much like the DTD for the genome encoding, the software implementation developed for this research called *GTMOEA* is also configured by an XML file. Appendix A itemizes this DTD. A separate users manual details all of the options available with this DTD [20]. Most importantly for this research, the modular design allows various components

to be configured separately using either standard or new innovative methods. Only those methods of interest for this research will be listed here. But as listed in the DTD, other research methods exist as well.

The modular design allows the base fitness values to be calculated using one of the following fitness methods: PR1 or PR2 (see Section 2.2.4.1), SPEA (see Section 2.2.5.2), SPEAII raw fitness (see Section 2.2.6.1), or the weighted-sum (see Section 2.2.1). The fitness values can then be scaled using any number of the following: hypercube distance scaling (see Section 3.3), the niching algorithm from NPGA (see Section 2.2.4), the transform scaler (see Section 3.1.4), and the SPEAII density (see Section 2.2.6.2). Note that fitness scaling methods can be applied multiple times.

Elitism rules are applied to a combination of the current population and the current elite population to create the next elite population. Elitism rules can be thought of as a set of pruning routines. The pruning rules include: pruning of individuals with identical fitness values, pruning of individuals with identical objective values, the Pareto Rank, which can be set to 1 to limit to Pareto optimal individuals (see Section 2.2.4.1), and the SPEA clustering algorithm (see Section 2.2.5.1). The final elitism rule, which is always applied, reduces the size of the elite population to the desired size by selecting individuals with the best fitness values.

Common crossover methods are included: clone crossover (see Section 2.1.3.1), N-Point crossover (see Section 2.1.3.2), and the newly developed location crossover (see Section 3.1.2). Mutation methods include: bitwise mutation (see Section 2.1.4.1), uniform mutation (see Section 2.1.4.2), and normal mutation (see Section 2.1.4.3).

This modular design and implementation allows not only the implementation of the existing MOEA methods, but recombinations of the fundamentals of these algorithms to create new methods. Bosman and Thierens investigated the relationship between density preservation and selection of non-dominated solutions in objective space [9], and indicate

the inability of SPEA algorithm to separately control these features. Allowing the combinations of the fundamentals of each of these algorithms overcomes these deficiencies. Various combinations are quantitatively investigated in Chapter 4.

3.2 Research Contributions for Computationally Expensive Objectives

This section details the contributions made for computationally expensive objectives, i.e., those objectives that require a substantial cost to evaluate as compared to the time required for EA processes. Two important contributions have been made. First, a method has been developed that allows the attachment to multiple remote processes potentially running on multiple processors for the evaluation of objective values. Second, a method has been developed that allows for evolution without large synchronization times between generations.

3.2.1 External Objective Evaluation

The ability to interface to an external evaluator allows the decoupling of the MOEA software from the objective evaluation software. As discussed in Section 3.1.1 each individual is translated into an XML description based on the search space description. For legacy software, the user then provides a translation layer of software that translates the individual XML description into the input format expected by the legacy software. After evaluation of the desired objective, the translation layer then translates the output from the legacy software into a simple objective name and objective value format for return to the MOEA software. These methods have been used for flare pattern design, AAR-44A OFP optimization, and evolutionary programming, (see Chapters 5, 6 and 7 respectively). Often the translation layer can be accomplished using common unix tools such as shell scripts and awk [7, 3, 18]. If new objective evaluators are required then they can be written to meet the XML implementation directly using available XML libraries.

The user is able to configure the external evaluator in several ways. The configuration file, (see Appendix A), allows the user to supply the name of the remote executable to be

started. Likewise, the user can supply additional strings before and after each individual objective is requested. In practice, this is often used to supply the names of the input training data files to be used for evaluation of the desired objective.

The user also supplies a list of machines on which to start the remote executables. The remote executable is started on each of the machines in the list using the remote shell command *rsh*. The user should make sure that *rsh* connections without a request for a password are available for each of the remote machines. The first command sent to the remote machine is a request to change directory to the same directory as the local machine. The existence of the directory of the same name should also be verified.

As an example, the actual command executed for the evaluator *test.sh* for the unix machine *albania* executed in the directory */home/moea* follows:

```
1 /usr/bin/rsh albania (echo \$\$; cd /home/moea;
2                       while test \$? -eq 0; do test.sh; done)
```

The while loop is included to allow continued execution of the objective evaluator even if it terminates normally.

Optionally, a temporary directory for running the evaluator may be required for some legacy executables that create temporary files that produce name conflicts if executed in the same directory. This is especially important when using several machines on a networked drive. If a temporary directory is indicated, a directory with the name of the processor followed by an underscore and an incrementing number is created in current directory, e.g., *localhost_1*.

In practice, it has been found that the use of the local drive in */tmp* for evaluator output files is advantageous. Using the local drive removes the need for cluttering the network directory with the sub-directories described in the previous paragraph. Likewise, the local directory can dramatically improve performance when the legacy software creates large temporary files, which can now be written to the local drive instead of the networked drive.

3.2.2 Parallel Evaluation

Section 2.1.6 outlined the current methods for single and multiple gene pool parallel objective evaluations. Also detailed in this section was the effect of the large synchronization times required at the end of each generation. These synchronization times become most detrimental for this particular class of problems where the computation time for objective evaluations is large in comparison to the time required to perform the EA operations, the evaluation time may be dependent on the genome being evaluated, and processes are running on many machines which may have various processing speeds due to their processing speed and loading.

To overcome the synchronization problems, this research contribution has designed and implemented an approach that obviates the need for synchronization much as the need for synchronization is eliminated in nature. The ability of an organism to reproduce is not necessarily dependent on the state of all other organisms in the population. There is no rule in nature that *all* organisms in the population must finish evaluation before *any* organism can begin the selection, mating and mutation processes required to create another individual. In this implementation, as in nature, an individual's ability to mate and raise the next generation is limited by their evaluation resources, not by those of the entire population.

To implement this approach, each processor is given an individual to evaluate. Remaining individuals in the current generation, *C*, are given to a processor for evaluation as they finish the evaluation of their current individual. When there are no more individuals in the current generation that need evaluating, or are not in the process of being evaluated, then the software creates the next generation based on the current population and the elite population. Without generation synchronization, the software also becomes more fault tolerant, e.g., if a processor goes down for some reason, the individual being evaluated is lost, but the entire evolutionary process is not halted as it would be with generation synchronization. The software can still be configured to run with generation synchronization, which

can be important for comparisons with published results. In particular, generation synchronization with the same MOEA inputs allows for deterministic behavior, i.e., rerunning the software will result in identical evolutionary paths, as long as the objective evaluator is also deterministic. Without generation synchronization, the software can not be guaranteed to be deterministic. This lack of deterministic behavior stems from the inability to guaranty the order that individuals will be returned from the machines performing the objective evaluations, and thus the inability to guaranty the individuals available for selection.

Figure 35 illustrates the processing flow of this hybrid method. As indicated in the figure, the processing flow contains three basic loops: objective evaluation, genome evaluation, and evolutionary algorithm. The following sub-sections examine each of the processing loops.

3.2.2.1 Objective Evaluation Loop

From Figure 35, the innermost loop of processing, the objective evaluation loop, can be found on the right-hand side of the figure. The processing titled "Listen for Machine" is responsible for listening to all slave machines and waiting on a reply from one of the processes that it has completed its evaluation of an objective. The listen operation is implemented using the "select" C function, which allows the process to halt until an input is received on any of a list of file descriptors [54, 55]. With this implementation one file descriptor is assigned to each of the remote processes that were started using the methods discussed in the Section 3.2.1.

Once a reply has been received, the processor must associate the file descriptor with an individual and extract the objective value into the data structures of the individual. These processes are indicated in the "Retrieve Objective Information" and "Update Objective Value" elements of the figure. These processing elements are also responsible for error detection and accounting for the remote processes. Three special cases may occur, which are not included on the figure to reduce clutter. First, the connection may be terminated

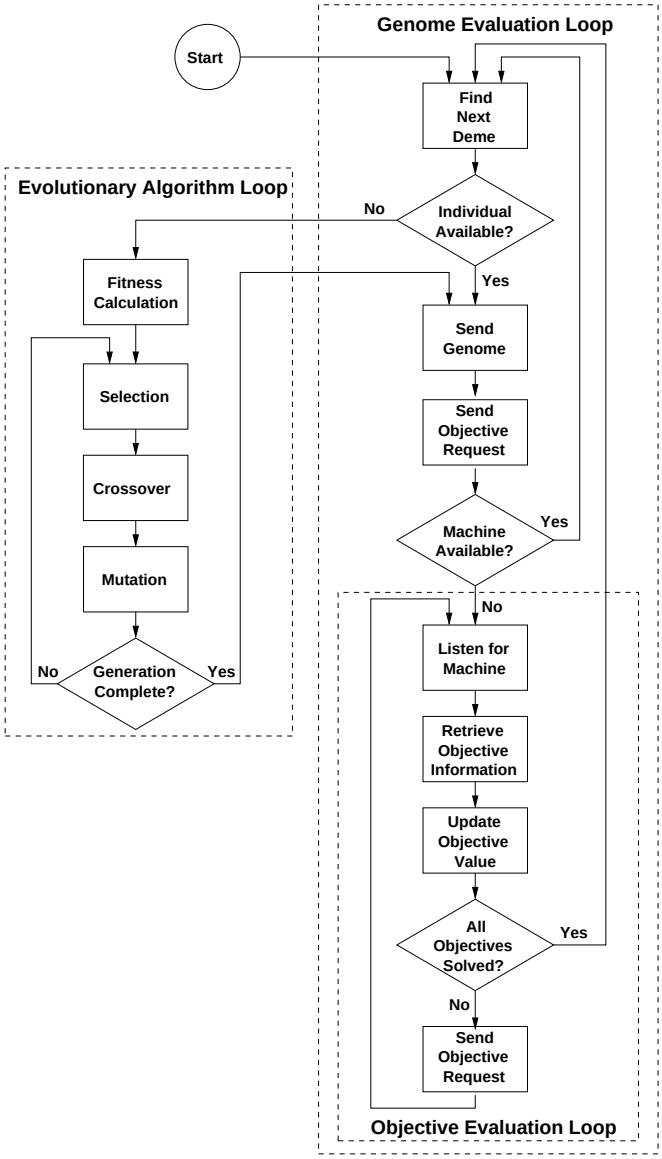

Figure 35: Hybrid Parallel MOEA Processing

due to problems with the remote evaluator or remote processors. Second, the individual genotypic description sent to the remote process for evaluation may not reside in the true feasibility region of the objective evaluator. The user does his best to define the feasibility region using the search space definition as defined in Section 3.1.1. But, due to the complexity of the system, it may not be possible to determine the complete feasibility region until objective evaluation. If an individual is not feasible, it is identified as a fatal allele to prevent further propagation of the genetic material. The third special case is an inappropriate reply from the remote objective evaluator. When this case is detected, the individual is also marked as fatal.

The exit criteria for this loop occurs if all objectives of the current individual have been evaluated. If all objectives have been evaluated, then the remote process associated with the individual is flagged as available, and processing control is returned to the genome processing loop discussed in the next sub-section. If all objectives of the current individual have not been completed, then a request for the next objective is made from the remote process, and control is returned to the listen element until the next input from a remote process is received.

3.2.2.2 Genome Evaluation Loop

From Figure 35, the genome evaluation loop is responsible for the evaluation of all individuals in the current generation. The support for multiple populations is inherent in the design, and is indicated in the "Find Next Deme" block. Unlike the single-gene pool parallel methods, (see Section 2.1.6.1), that only support one deme, and the island method, (see Section 2.1.6.2), that supports creation of *n* populations on *n* processors, this hybrid method supports *n* populations or demes running on *m* processors. In addition, *n* can be greater than, less than, or equal to *m*.

The equality of *n* and *m* requires the scheduling of individuals from each deme to a

processor in a scheme that provides each deme a roughly equivalent fraction of the processors, and thus processing time. To implement the scheduling algorithm, when a machine completes the evaluation of an individual and it is time for another individual, the scheduler totals the number of machines currently assigned to each deme. The scheduler then procures an individual from the deme with the least machines currently assigned, and assigns the individual to the machine. Logic is also included to prevent the selection of the same deme multiple times in case of a tie.

If there are no remaining individuals in the selected deme, then the evolutionary algorithm loop for the selected deme must be executed as discussed in the next sub-section. Once an individual is available, then the process sends the XML genotypic description of the individual, as well as the name of the first objective to be evaluated, to the remote process.

This entire process is repeated until all available remote processes have received an individual for processing. Once all remote processes are processing individuals, the master process enters the "listen" process of the objective evaluation loop discussed in the previous sub-section.

3.2.2.3 Evolutionary Algorithm Loop

When an individual is required from a deme that does not have any addition individuals available, the evolutionary algorithm processing is executed for that particular deme. As indicated in Appendix A, each deme can be configured with separate fitness, selection, crossover and mutation operators. The first process, the calculation of the fitness for the current and elite populations of the deme, requires proper comparison of individuals for selection.

The current population may contain individuals that are not completely evaluated, but

still may have a fitness value assigned based on the partial evaluation, and thus have a probability of mating. This is accomplished by setting the objective values for unsolved objectives to their minimum value. By using the minimum value, evaluation of the remaining objective values cannot decrease the fitness value. Thus, the probability of mating increases as more objectives are evaluated. Use of individuals that are in progress is most important for the independent, computationally expensive objective class of problems, where the total evaluation of an individual may require minutes to hours of time, but the evaluation of a single objective may only require several minutes.

After the fitness is calculated, the EA operators of selection, crossover and mutation are repeated until the desired number of individuals is created. The only new concept added is the transfer of genetic material between demes. This is implemented much as breeding is performed in domestic animal populations, i.e., a fixed percentage of adults are retrieved from other populations. Other implementations allow for the migration from other populations of individuals with their associated objective values. Because for our system there is *not* a requirement for multiple populations to be working on the same objectives, if a good individual of another population is migrated to the current population it may result in a poor fitness value because the objectives of the current population may not have been evaluated. Thus, the migrated individual would have a low probability of mating resulting from the "apples to oranges" comparison between demes.

By changing from a migration to a breeding analogy, selection of the individual can occur by comparing individuals only to those within the same population. Therefore, when an individual is desired from a second population, the selection of the individual occurs with the fitness values of the second population. To prevent recalculation of the fitness values in other populations, only the elite individuals of the second population are considered for breeding.

The number of populations, number of individuals in the current population $|C|$, and the number of individuals in the elite population $|E|$, can all be configured by the user. Four

considerations should be addressed when selecting these values. First, the time required to calculate the fitness values is related to the number of individuals in the elite and current population. From Section 2.2.7, the time required for fitness calculations is of order $(|E| + |C|)^2$ or worse. Second, lowering the number of individuals in the current population allows incorporation of new individuals with better performance as parents much quicker. Note the trade-off that lowering the number of individuals allows quicker incorporation of improved individuals, but it also requires recalculation of fitness values more often. Third, increasing the number of demes decreases the number of individuals that can be calculated for each deme since they must be spread across the same number of processors. Fourth, increasing the number of demes can allow decreasing the number of elite individuals in each deme $|E|$ while still maintaining a large overall elite population. Assuming the use of n populations and a constant number of overall elite individuals $|E|$, then the overall fitness complexity can be reduced from $(|E| + |C|)^2$ to $n(\frac{|E|}{n} + |C|)^2$.

3.3 Research Contributions for Objective Space Region of Interest

Section 1.4 first introduced the concept of a region of interest in the objective space. The region of interest is defined by the lower boundary L_k for each of the q objectives. These boundaries, derived from the requirements of the optimization problem, define the region of interest, $\boldsymbol{G} \subset \boldsymbol{Y}$, as those locations in $\boldsymbol{y}$ such that for all $k = 1$ to q, $y_k \geq L_k$. Thus the region of interest is the region enclosed by the hypercube $h_f(\boldsymbol{L})$.

The hypercube distance HCD scaler method is a research contribution that utilizes the distance to the hypercube to scale fitness values. The basic premise of the HCD scaler method is that those individuals that reside closest to the region of interest are most likely to produce children that reside closer to the region of interest. Since the region of interest is defined by a hypercube in objective space, the closer the individual is to the hypercube the higher the probability of selection.

In addition to the region of interest defined by $h_f(\boldsymbol{L})$, the minimum acceptable objective values for all dimensions are required. A hypercube $h_f(\boldsymbol{S})$ is identified as a region in the objective space where solutions are known to be possible. The q values of S_j identify the minimum value acceptable for each of the q objectives. Setting of these values requires domain knowledge of the specific problem being evaluated in order to specify limits that are known to be obtainable.

Distance is calculated using the Holder measure as defined in Equation 50. The specification of the HCD scaler, as shown in Appendix A, allows for the specification of the Holder coefficient p, and an offset α. The offset allows modification the relative importance of HCD scaling. The distance to the hypercube is then:

$$HCD(i, \boldsymbol{L}, \boldsymbol{S}) = \frac{\alpha + \left(\sum_{j=1}^{q} (d_j(i, \boldsymbol{L}, \boldsymbol{S}))^p\right)^{\frac{1}{p}}}{1 + \alpha} \tag{97}$$

where q is the number of dimensions, $d_j(i, \boldsymbol{L}, \boldsymbol{S})$ is the distance of individual i in the j^{th} dimension to the hypercube $h_f(\boldsymbol{L})$ with a lower limit defined by the hypercube $h_f(\boldsymbol{S})$ and Holder coefficient p.

The distance to the hypercube $h_f(\boldsymbol{L})$ for a dimension d_j is calculated by normalizing the distance to the hypercube for each objective to a range of $[0, 1]$ using:

$$d_j(i, \boldsymbol{L}, \boldsymbol{S}) = \begin{cases} 0 & f_j(i) \geq L_j \\ \frac{L_j - f_j(i)}{L_j - S_j} & S_j < f_j(i) < L_j \\ 1 & f_j(i) \leq S_j \end{cases} \tag{98}$$

Combining Equations 97 and 98 results in the hypercube distance of:

$$HCD(i, \boldsymbol{L}, \boldsymbol{S}) = \frac{\alpha + \left(\sum_{j=1}^{q} \left(\begin{cases} 1 & f_j(i) \leq S_j \\ \frac{L_j - f_j(i)}{L_j - S_j} & S_j < f_j(i) < L_j \\ 0 & f_j(i) \geq L_j \end{cases}\right)^p\right)^{\frac{1}{p}}}{1 + \alpha} \tag{99}$$

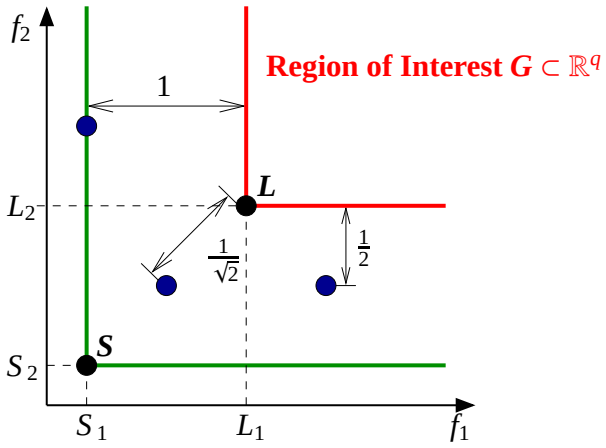

Figure 36: Hypercube Distance Examples

Normalization of the distances also implies that the HCD scaler is in the range

$$\left[\frac{\alpha}{1+\alpha}, \frac{\alpha + q^{\frac{1}{p}}}{1+\alpha}\right].$$

Figure 36 illustrates several examples of the HCD calculation with α set to zero.

If maximization of the fitness values is desired then the fitness value is *divided* by the HCD. This results in an increase in fitness for those individuals near the region of interest, and a decrease in the fitness for those individuals far away from the region of interest. For maximization problems the scale amount or gain becomes

$$G_{HCD} = \frac{1+\alpha}{\alpha + \left(\sum_{j=1}^{q} \left(\begin{cases} 1 & f_j(i) \le S_j \\ \frac{L_j - f_j(i)}{L_j - S_j} & S_j < f_j(i) < L_j \\ 0 & f_j(i) \ge L_j \end{cases} \right)^p \right)^{\frac{1}{p}}}. \tag{100}$$

The maximum gain occurs as the solutions approach the region of interest and thus the gain approaches

$$G_{HCD} = \frac{1+\alpha}{\alpha} = 1 + \frac{1}{\alpha}. \tag{101}$$

Likewise, if minimization of the fitness value is desired then the fitness value is *multiplied* by the HCD. This results in a decrease in fitness for those individuals near the region

of interest, and an increase in the fitness for those individuals far away from the region of interest.

The following listing details the HCD scaler algorithm and four comparison locations A through D.

```
1  // Loop through each member of the population
   for (i in P) {                                          // Comparison A
      dSum = 0.0;
      // Loop through each objective
5     for (k=1; k<=q; k++) {                               // Comparison B
         // Set to minimum zero if below minimum
         if (i.f[k] <= S[k]) {                            // Comparison C
            fNorm = 0.0;
         // Set to minimum zero if above maximum
10       } else if (i.f[k] > L[k]) {                      // Comparison D
            fNorm = 1.0;
         // Other, in valid range. Normalize
         } else {
            fNorm = (i.f[k] - S[k] ) / (L[k] - S[k] );
15       }
         // Increment Hypercube
         dSum += pow( 1.0 - fNorm, HolderCoefficent )
      } // End of loop through objectives
      // Update fitness assuming minimize problem
20    i.FNew = i.F * pow( dSum, 1.0 / HolderCoefficent);
   } // End of loop through population
```

Allowing n to be the number of points in the transformation list, the total number of comparisons is:

$$\begin{aligned}
\text{(from line 2) } N_A &= |P| \\
\text{(from line 5) } N_B &= qN_A = q|P| \\
\text{(from line 7) } N_C &= N_B = q|P| \\
\text{(from line 10) } N_D &= N_B = q|P| \\
N_{HCD-Scaler} &= N_A + N_B + N_C + N_D \\
N_{HCD-Scaler} &= (1 + 3q)\,|P|. \qquad (102)
\end{aligned}$$

3.4 Research Contributions for Independent Computationally Expensive Objectives

Independent, Computationally Expensive Objectives (ICEO) is the final and most specialized category of research contributions. In the previous section, evaluation was defined as computationally expensive when it required a substantial cost to evaluate as compared to the time required for EA operations. For objectives to be independent implies that a separate time-consuming objective evaluation is required for each objective. Two important contributions for ICEO problems are detailed in the following subsections. First, dynamic objective thresholding utilizes historical information to eliminate objective evaluations for individuals that are not expected to produce children closer to the region of interest in objective space. Second, dynamic objective ordering uses historical information to order objectives for evaluation such that the dynamic objective thresholding is able to eliminate poor-performing individuals faster than with the default objective ordering.

3.4.1 Dynamic Objective Thresholding

Dynamic objective thresholding is the core method that allows using the MOEA software for independent computationally expensive objectives (ICEO) to increase the speed at which solutions are found in the region of interest. Dynamic objective thresholding is used to terminate the evaluation of an individual before all objectives are evaluated if it is determined based on the evaluated objectives that the individual is not likely to improve performance. It is only by not evaluating all objectives for every individual that speed improvements can be made. Other methods [23] have used human intervention to limit the size of the objectives space during early evolution. But, no other method automatically determines when the number of objectives to be solved on an individual basis.

This method creates a dynamic threshold for each objective. If, after the evaluation of an objective for a specific individual, it is determined that the objective value does not exceed the threshold, then the remaining objectives for the individual are not evaluated. An

individual that does not have all objectives evaluated still has a fitness value. The fitness value is obtained with the remaining objectives values set to the minimum value for the objective.

The dynamic capability is enabled by keeping a historical population H_T of the $|H_T|$ individuals most recently evaluated. The calculation of the dynamic threshold occurs after the evaluation of each set of D_T individuals. In addition, the hypercube definitions for the region of interest $h_f(\boldsymbol{L})$ and the minimum acceptable values defined by the hypercube $h_f(\boldsymbol{S})$ are required, (see Sections 1.4 and 3.3 respectively for definitions of these hypercubes). The final attributes required are a set of q weights $\boldsymbol{W}$, one for each objective. Each weight W_j specifies the fraction of individuals that *fail* the objective test.

The dynamic threshold for the j^{th} objective is implemented by sorting into increasing objective values $f_j(i)$ the list of individuals in H_T. The k^{th} individual i_k in the sorted list of $|H_T|$ individuals in H_T is then chosen to set the new threshold where $k = W_j|H_T|$. Thus the threshold becomes

$$T_j(H_T) = \begin{cases} S_j & f_j(i_k) \le S_j \\ f_j(i_k) & S_j < f_j(i_k) < L_j \\ L_j & f_j(i_k) \ge L_j \end{cases} \tag{103}$$

The dynamic thresholding method must be robust enough to handle the condition that solutions in the region of interest may not exist; i.e., just because it is desired to find a solution with a particular performance does not mean that a system with the desired performance is attainable. Therefore, the thresholding methods must find solutions closer and closer in objective space to the region of interest. But the thresholding methods must degrade such that if solutions in the region of interest are not found, then the user has a set of Pareto optimal solutions near the region of interest from which to choose.

Figure 37 illustrates the desired behavior of the dynamic thresholds over the course of evolution. Note that during Stage 1 $f_1(i)$ is less than S_1, resulting in the evaluation of only

a single objective. During Stage 2 the objective values $f_1(i)$ reach S_1 for some individuals, allowing evaluation of $f_2(i)$ for these individuals. During Stage 3 the population has gained enough individuals, $k = W_1|H_T|$, passing the minimum threshold S_1 that the dynamic objective threshold $T_1(H_T)$ is used. Finally, during Stage 4 the population reaches the region of interest $h_f(\boldsymbol{L})$.

Figure 38 illustrates the modifications to the parallel processing first illustrated in Figure 35 required to implement the dynamic objective thresholding. The processing for "Apply Thresholds" applies the current thresholds to the current individual to determine if evaluation of additional objectives is required after the evaluation of each objective. Since the thresholds may have been recalculated since the start of evaluation for the current objective, the comparison is required for all solved objectives.

As evolution proceeds, it is anticipated that the threshold values steadily increase. Thus, if any objective is solved and does not exceed the current thresholds, then the remaining objectives in the list are marked to be ignored regardless of whether they have been solved or not. Note that the "Apply Thresholds" processing is also applied to the entire population of each deme before the creation of the next generation in the "Evolutionary Algorithm Loop". This recalculation assures application of a common set of thresholds to all individuals to ensure proper comparisons during fitness calculations, e.g., an individual should not have a better fitness value simply because it had an objective solved before the threshold for that objective was raised.

There are two situations that could result in a decrease of the threshold values. First, if little progress is made during the previous D_T individual evaluations, then the threshold value may be slightly less that the current value. Second, if the dynamic objective ordering described in Section 3.4.2 is used, then the recalculation of the threshold values for objectives which are now earlier in the list may force ignoring of objectives that are now later in the objective list that were previously not ignored. This circumstance could easily result in drastically lowering the dynamic threshold for the now later objective.

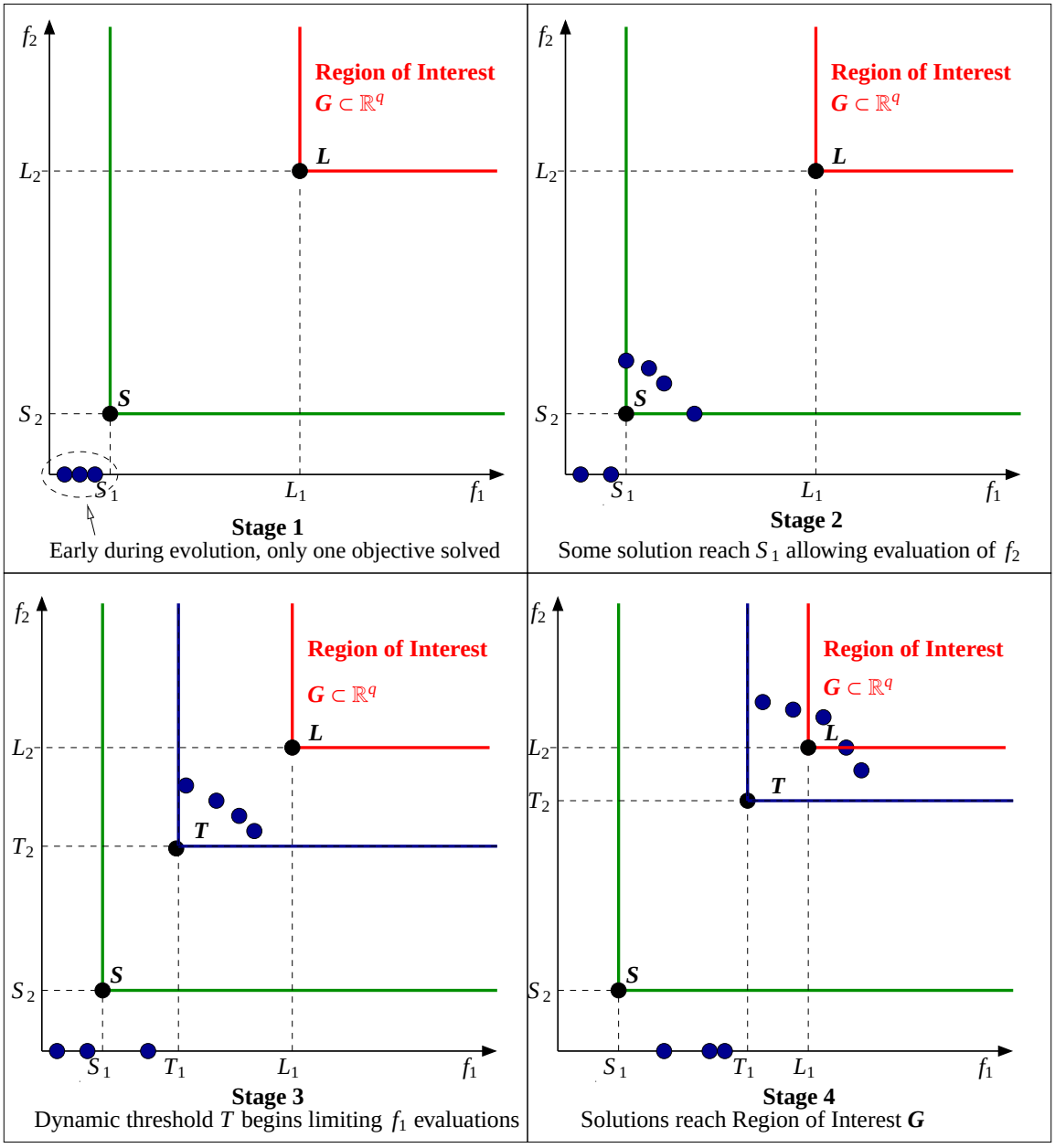

Figure 37: Dynamic Objective Thresholding Examples

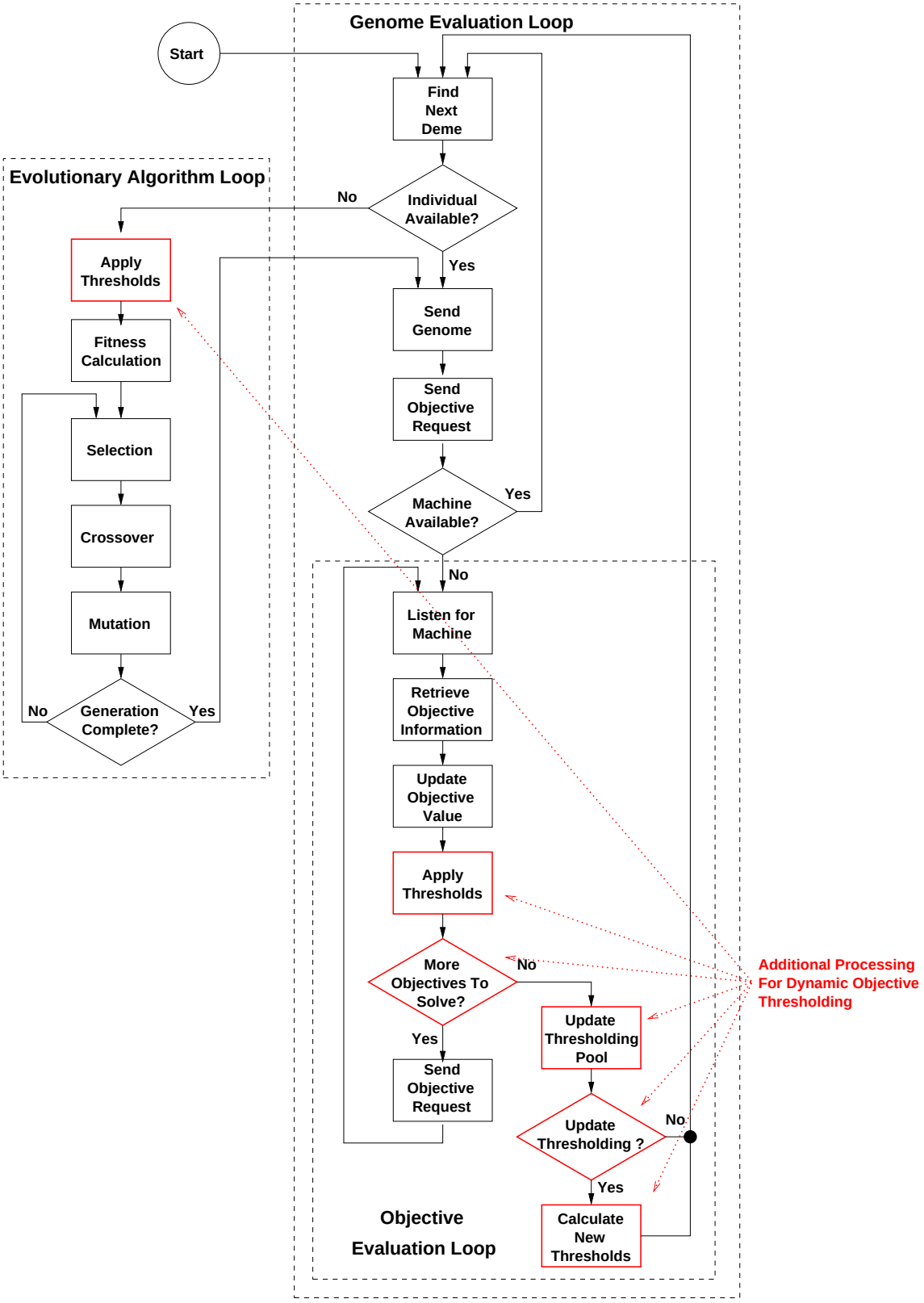

Figure 38: Dynamic Objective Thresholding Processing

The "More Objectives To Solve?" decision is affirmative if there is another objective for the individual that has not been evaluated and has not been marked to be ignored. In the affirmative case, the processing advances to send a request for the next objective to the remote process. Otherwise, the genome is considered solved, and "Update Threshold Pool" processing is invoked to add the solved genome to the dynamic thresholding population H_T.

The "Update Thresholds?" decision is affirmative if D_T individuals have been added to the threshold population since the last recalculation of thresholds. If an update of the thresholds is not required then the processing is returned to the "Genome Evaluation Loop" to begin the evaluation of the next individual. If an update of the thresholds $\boldsymbol{T}$ is required then the "Calculate New Thresholds" processing is invoked to apply calculations of Equation 103.

The following listing details the calculation of the dynamic thresholds from the historic population H_T.

```
// Loop through each objective
for (k=1; k<=q; k++) {                                  // Comparison A
    // Sort list based on kth objective
    for (m=1; m<(|H_T|-1); m++) {                       // Comparison B
        dMin = H_T[m].f[k]
        iMin = m;
        // Search all individual above present for minimum
        for (n=m+1; n<=|H_T|; n++) {                    // Comparison C
            if (H_T[n].f[k] < dMin) {                   // Comparison D
              dMin = H_T[n].f[k];
              iMin = n;
            }
        } // End of loop through remainder of H_T
        // Swap values
        iTmp = H_T[iMin];
        H_T[iMin] = H_T[m];
        H_T[m] = iTmp;
    } // End of loop through population H_T
    // Find individual with desired threshold
    m = W[k] * |H_T|;
    // Set threshold to desired value
    T[k] = H_T[m].f[k];
```

```
} // End of loop through objectives
```

Using the four comparison locations A through D, the total number of comparisons is:

$$\begin{aligned}
\text{(from line 2) } N_A &= q \\
\text{(from line 4) } N_B &= q(|H_T| - 1) \\
\text{(from line 8) } N_C &= q \sum_{j=1}^{|H_T|-1} j = q\frac{|H_T|(|H_T| - 1)}{2} \\
\text{(from line 9) } N_D &= N_C \\
N_{Threshold}^{Calculate} &= N_A + N_B + N_C + N_D \\
N_{Threshold}^{Calculate} &= q|H_T|^2. \qquad (104)
\end{aligned}$$

The following listing details the application of the dynamic thresholds to a population of individuals as is required before the recalculation of fitness values.

```
1  // Apply thresholds to population
   for (i in H_T) {                                    // Comparison A
      bIgnoreRemaining = false;
      // Loop through each objective
5     for (k=1; k<=q; k++) {                           // Comparison B
         // If below threshold set state of this and remaining
         // objectives to INGORE
         if ( i.f[k] < T[k] ) {                       // Comparison C
            bIgnoreRemaining = true;
10       }
         // If we are to ignore remaining simply set the state
         if (bIgnoreRemaining) {                      // Comparison D
            i.state[k] = IGNORE;
            i.f[k] = MinValue[k];
15       }
      } // End of loop through objectives
   } // End of loop through population
```

Using the four comparison locations A through D, the total number of comparisons is:

$$\begin{aligned}
\text{(from line 2) } N_A &= |H_T| \\
\text{(from line 5) } N_B &= qN_A = q|H_T| \\
\text{(from line 8) } N_C &= N_B = q|H_T| \\
\text{(from line 12) } N_D &= N_B = q|H_T| \\
N_{Threshold}^{Apply} &= N_A + N_B + N_C + N_D \\
N_{Threshold}^{Apply} &= (1 + 3q)\,|H_T|. \qquad (105)
\end{aligned}$$

3.4.2 Objective Ordering

Objective ordering allows the user to preset the order and the software to automatically modify the order in which objectives are evaluated. This capability feeds two purposes. First, the software can dynamically order the objectives allowing objective orders that further improve speed performance beyond that of dynamic objective thresholding with the default objective ordering alone. Second, the user can force different orderings of objectives for different demes, allowing the different demes to focus on different desired attributes of the final individual. Note that either method requires the dynamic objective thresholding algorithm of the previous section to eliminate the evaluation of objectives. *Without dynamic objective thresholding, the objective ordering can not provide any performance improvement.*

There are currently two dynamic objective ordering algorithms: hypercube distance and auto ordering. The hypercube distance ordering algorithm orders objectives based on their average distance from the region of interest. This allows concentration on objectives that are furthest from the region of interest. The auto ordering method takes into account the time to evaluate each objective, and tries to order objectives in an order that reduces the time to evaluate a total population.

The determination of when to recalculate the objective orders is the same for both methods and similar to the determination made for dynamic objective thresholding. The historic

population for dynamic objective ordering, H_O, contains $|H_O|$ of the previous individuals solved. The number of individuals in the historic population should span several generations. A default value of 1000 has been selected for $|H_O|$. The recalculation of objective orders is calculated every D_O individuals. Due to the complexity of the calculations for the dynamic objective ordering and the need to remove noise from the measured statistics that drive the objective ordering processes, the number of individuals solved before recalculation is likewise large with a default value of 1000.

Some objectives may be required to be evaluated, or other times it may be desired to specify the exact order in which all objectives are evaluated. To allow this non-dynamic ordering of some objectives, both dynamic methods support the ability to "force" a list of objectives to be evaluated first. The ordering for the remaining list of objectives is then subject to algorithms of one of the two ordering methods. By default, the initial order of the remaining objectives is the order given in the specification of the list of objectives, (see Appendix A). But, if there is no known advantage to this order, the user can specify the order of the remaining objectives be initialized in a random order.

Figure 39 indicates the modifications to Figure 38 required to implement either dynamic objective ordering processing method. The "Update Ordering?" decision is affirmative if D_O individuals have been added to the threshold population since the last recalculation of objective ordering. If an update of the objective ordering *is not* required then the processing is forwarded to the application of the objective thresholds.

If an update of objective order is required then the selected objective ordering algorithm is invoked. The specifics of the hypercube distance ordering and auto ordering methods are discussed in the following two subsections. Once objective orders have been recalculated, then the dynamic objective thresholds must be recalculated to allow the effects of the ordering change to affect the thresholds. After recalculation of the objective thresholds, the processing advances to the application of objective thresholds and then to the fitness calculation as normal.

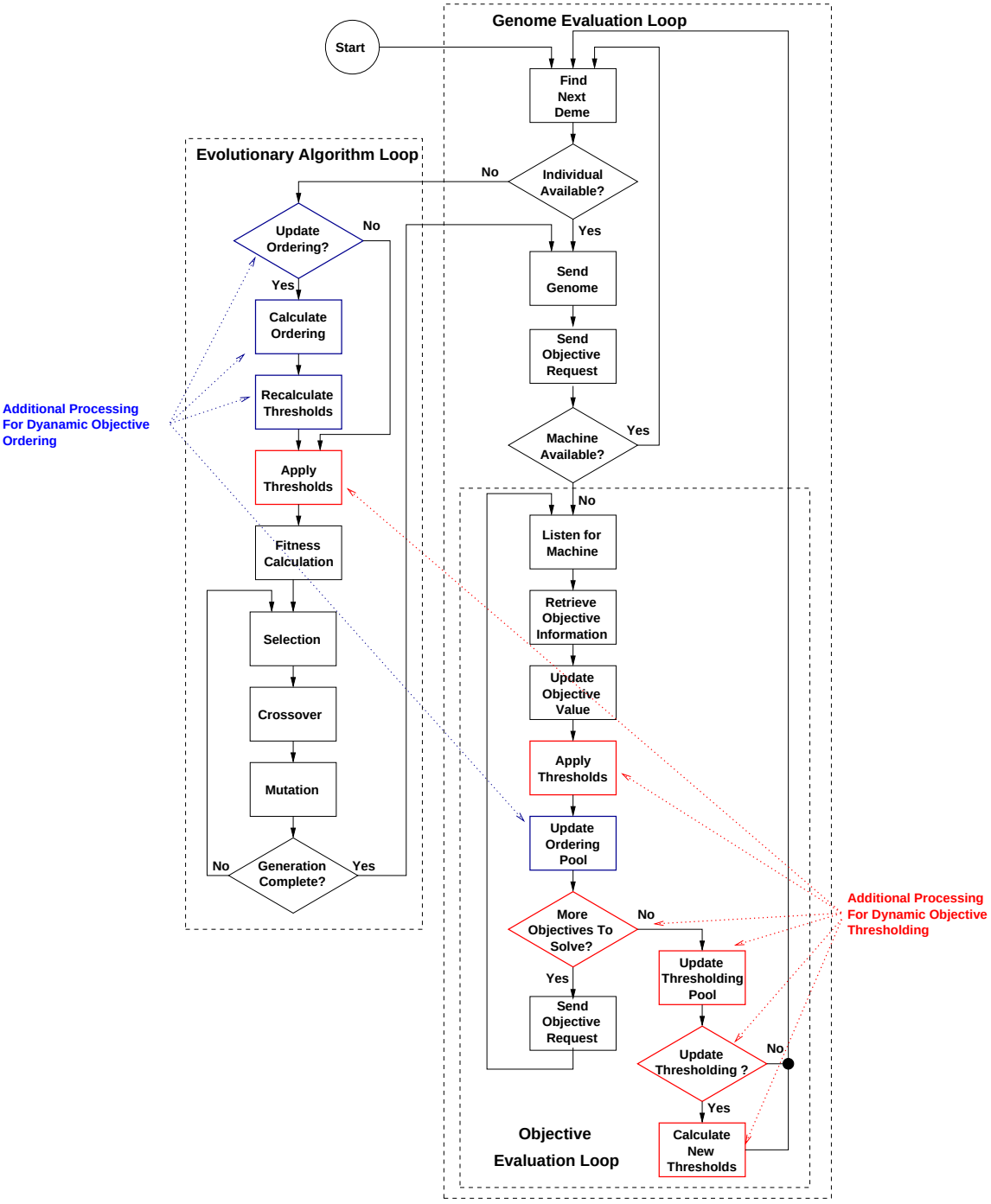

Figure 39: Dynamic Objective Ordering Processing

3.4.2.1 *Hypercube Distance Cube Ordering*

Each time the HCD ordering method is called, the average objective distance to the region of interest is calculated for those individuals with solved objective values in each objective dimension. The objective with the largest average distance to the region of interest is then placed at the front of the objective list. Likewise, the remaining objectives are ordered to contain successively decreasing average distances to the region of interest.

Similar to the HCD scaler method presented in Section 3.3, the HCD average distance uses a normalized value of the distance based on the two hypercubes $h_f(\boldsymbol{S})$ and $h_f(\boldsymbol{L})$. By using the user specified definitions of the minimum value acceptable for each objective $h_f(\boldsymbol{S})$ and the region of interest $h_f(\boldsymbol{L})$, any disparities in the ranges of objective values can be removed. The normalized distance for each dimension of objective j for an individual i is equivalent to the HCD scaler method.

$$d_j(i, \boldsymbol{L}, \boldsymbol{S}) = \begin{cases} 1 & f_j(i) \leq S_j \\ \frac{L_j - f_j(i)}{L_j - S_j} & S_j < f_j(i) < L_j \\ 0 & f_j(i) \geq L_j \end{cases} \tag{106}$$

The following listing details the application of the HCD ordering method to a population of individuals.

```
// Apply thresholds to population
for (i in H_T) {                                    // Comparison A
   bIgnoreRemaining = false;
   // Loop through each objective
   for (k=1; k<=q; k++) {                           // Comparison B
      // If below threshold set state of this and remaining
      // objectives to INGORE
      if ( i.f[k] < T[k] ) {                        // Comparison C
         bIgnoreRemaining = true;
      }
      // If we are to ignore remaining simply set the state
      if (bIgnoreRemaining) {                       // Comparison D
         i.state[k] = IGNORE;
         i.f[k] = MinValue[k];
```

```
15          }
16      } // End of loop through objectives
    } // End of loop through population
```

The maximum number of calculations occurs if there are not any forced objective orders. Using the nine comparison locations A through I, the maximum number of comparisons is:

$$
\begin{aligned}
\text{(from line 2) } N_A &= q \\
\text{(from line 9) } N_B &= |H_O| N_A = q|H_O| \\
\text{(from line 11) } N_C &= N_B = q|H_O| \\
\text{(from line 13) } N_D &= N_B = q|H_O| \\
\text{(from line 15) } N_E &= N_B = q|H_O| \\
\text{(from line 26) } N_F &= N_A = q \\
\text{(from line 33) } N_G &= q - 1 \\
\text{(from line 37) } N_H &= \sum_{j=1}^{q-1} j = \frac{q(q-1)}{2} \\
\text{(from line 38) } N_I &= N_H \\
N^{HCD}_{Ordering} &= N_A + N_B + N_C + N_D + N_E + N_F + N_G + N_H + N_I \\
N^{HCD}_{Ordering} &= 4|H_O|q + q^2 + 2q - 1 \qquad (107)
\end{aligned}
$$

3.4.2.2 Auto Ordering

The auto ordering method attempts to order objectives not just based on the objective values, but also based on the time required to evaluate each objective. With this method the evaluation process records a time stamp for the time when the objective was requested from the external objective evaluator and the time when the resulting objective value was received. The time required for the objective evaluation is then calculated as the time between these time stamps. The individuals maintained in the historic population H_O are then "replayed" to determine the objective order that minimizes the time required to evaluate the

historic population and still result in a population of individuals where the Pareto optimal solutions are similar to the Pareto optimal solutions of current objective ordering. Relative to the HCD ordering method, the auto ordering method is much more complex. Figure 40 illustrates the internal processing for recalculating the objective order.

The first step is to establish the possible orders. There are at most $q!$ possible orders for the q objectives. This number can be reduced in two ways. First, any forced objectives can be removed from those that can be reordered. Secondly, because of dynamic objective thresholding, many objectives at the end of the current objective ordering may not have been solved for any of the individuals in the historic population H_O. Thus, there is not enough information for analysis of the timing effects of reordering for these unsolved objectives. If after the forced and unsolved objectives are removed there are one or less objectives in the remaining objective list then the current order must be maintained because there is not enough timing information to make any comparisons among possible orders. If the number of objectives in the remaining list, r, is two or more then the $r!$ possible orderings of these lists are created.

Once the possible objective orderings have been enumerated, the time required to evaluate each of the objective orders must be calculated. Auto objective ordering seeks to find the order that eliminates poor performers with the least objective evaluation time. It is the effects of dynamic objective thresholding that allow the improvement in performance by removing the solution of objectives. Since thresholds are applied in the order of the objectives, the order of the objectives can effect the values of the thresholds, i.e., when an objective fails to reach the threshold then the objective values for the remainder of the list are marked to be ignored and interpreted as the minimum value. Thus, for each objective order the dynamic threshold must be recalculated and applied to the historic population H_O. The time required to evaluate each objective is then summed for each individual in the population.

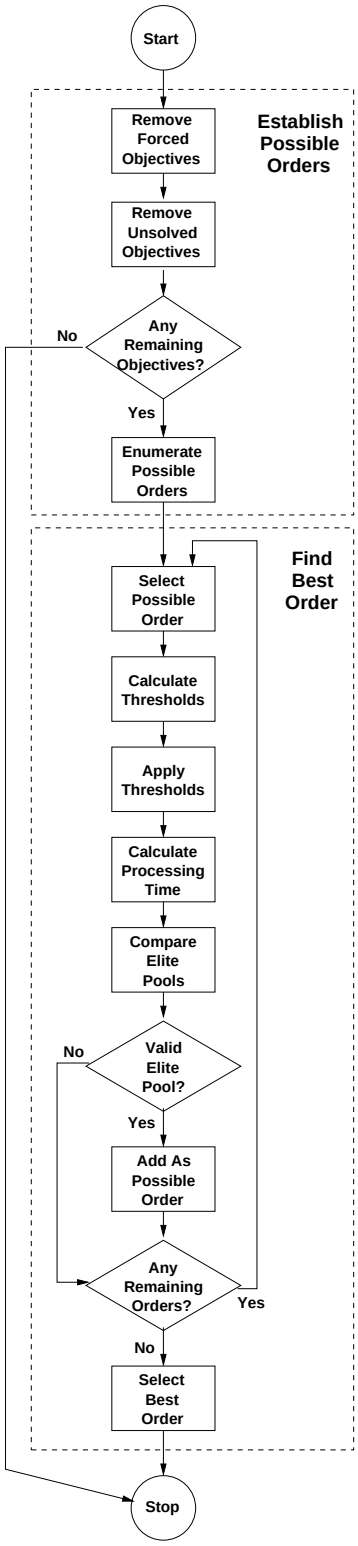

Figure 40: Auto Ordering Processing

If the time required to evaluate the total populations with the proposed objective ordering is less than that of any of the valid orders then a check must be made to check the validity of the proposed objective order. The assumption of the validity test is that the "important" individuals of the historic population H_O with the current ordering should also be the "important" individuals when the new objective order is applied. One definition of "important" would be those individuals with the highest probability of mating. But, due to mappings such as fitness sharing and clustering that can dramatically reduce the fitness values for individuals that are close together in objective space, this definition was not selected. Instead, a population of s individuals that are closest to the Pareto front are selected using the Pareto Rank 2 method as detailed in Section 2.2.4.1.

Note that the validity of orders must be calculated to prevent fast running, but less meaningful objective ordering selections. For example, if a fast running objective with a threshold that is not being exceeded by any of the current individuals in H_O is placed first in the objective list then this would be the fastest running order. Unfortunately, this would result in a trivial Pareto optimal population with a single individual, and would not likely provide the same set of s "important" individuals.

Once the s individuals are selected from the current objective order to create a population A and the proposed objective order to create a population B, then the difference D in the populations can be solved by totaling the number of individuals that are not in the other population.

$$\begin{aligned} D &= |(A \cap \overline{B}) \cup (B \cap \overline{A})| \\ &= |(A \cap \overline{B})| + |(B \cap \overline{A})| \\ &= |\{a \in A | a \notin B\}| + |\{b \in B | b \notin A\}| \end{aligned} \tag{108}$$

Therefore, if A and B contain the same individuals $D = 0$, and if there is no intersection between A and B then $D = |A| + |B| = 2s$. It is rarely expected that A and B contain the same individuals, therefore the user must set a value m the maximum allowable value for

D.

The value of *s* should be set to a number larger than the expected number of Pareto optimal individuals, which varies based on the type of optimization problem. The default value of *s* is 10. The maximum number of differences *m* should be set a number in the range $[0, 2s]$. A value of *m* that is at less than 50 percent of *s* is recommended. A future enhancement would be to calculate *m* from the statistics of the historic population H_O instead of from a user setting, e.g., *m* could be set to the number of rank 1 and 2 individuals in H_O.

The following listing details the application of the auto objective ordering algorithm.

```
// Find end of non-forced and solved objectives
for (j=iForced+1; j<=q; j++) {                    // Comparison A
   iEnd = j;
   bFound = false;
   // Loop through history to see if solved
   for (i in H_O) {                               // Comparison B
      if (i.state[iOrder[j]] == SOLVED) {         // Comparison C
         bFound = true;
         break;
      }
   } // End of loop through history pool
   // If not found we are just past the end of the solved objectives
   if (! bFound) {                                // Comparison D
      iEnd = j - 1;
      break;
   }
}
// Return if no objectives to reorder
r = iEnd - iForced;
if ( r <= 1 ) {                                   // Comparison E
   return;
}
// Enumerate possible objectives                  // Comparison F
ListOfOrders = FindAllOrders(iOrder,q,iForced,iEnd);
// Calculate base s individuals for later comparison
basePool = H_O.BestPareto(s);                     // Comparison G
basePool.sortID();                                // Comparison H
dMinTime=LARGE_NUMBER;
// Loop through all objective orders
```

```
for ( p=1; p<r!; p++ ) {                          // Comparison I
   for (k=1; k<=q; k++) {                         // Comparison J
      iOrder[k] = ListOfOrders[p][k];
   }
   CalculateThresholds(H_O)                       // Comparison K
   ApplyThresholds(H_O)                           // Comparison L
   // Calculate Processing Time
   dCurTime = 0.0
   for (i in H_O) {                               // Comparison M
      for (k=1;k<=q;k++) {                        // Comparison N
         if (i.state[iOrder[k]] == SOLVED){       // Comparison O
            dCurTime += i.time[iOrder[k]];
         }
      } // End of loop through objecitves
   } // End of loop through history pool
   // Check for new minimum time
   if (dCurTime < dMinTime) {                     // Comparison P
      // Check for validity of s Pareto points
      comparePool = H_O.BestPareto(s);            // Comparison Q
      comparePool.sortID();                       // Comparison R
      // Now, compare resulting pools
      iDiff = 0;
      j = 1;
      k = 1;
      while( (j <= s) && (k <= s) ) {             // Comparison S
         // If the same simply advance both indicies
                                                  // Comparison T
         if (basePool[j].ID == comparePool[k].ID) {
            j++;
            k++;
         // If different advance counter          // Comparison U
         } else if (basePool[j].ID < comparePool[k].ID){
            j++;
            iDiff++;
         } else {
            k++;
            iDiff++;
         }
      } // End of loop through pools
      // Add remaining individuals
      iDiff += (s - j + 1);
      iDiff += (s - k + 1);
      // Calculate difference
      dDiff = iDiff / s;
      // If difference is acceptable set time
```

```
75          if (dDiff < m) {                                    // Comparison V
76              dMinTime = dCurTime
                for (k=1; k<=q; k++) {                          // Comparison W
                    bestOrder[k] = iOrder[k]
                }
80          }
        } // End of new minimum time
    } // End of loop through orders
    // Set order
    for (k=1; k<=q; k++) {                                      // Comparison X
85      iOrder[k] = bestOrder[k]
    }
```

The maximum number of calculations occurs if there are not any forced objective orders and all individuals dominate the next individual. There are twenty-four comparison locations A through X. The first five comparisons are:

$$\begin{aligned}
\text{(from line 2) } N_A &= q \\
\text{(from line 6) } N_B &= N_A|H_O| = q|H_O| \\
\text{(from line 7) } N_C &= N_B = q|H_O| \\
\text{(from line 13) } N_D &= N_A = q \\
\text{(from line 20) } N_E &= 1
\end{aligned}$$

To enumerate all possible orders requires $q!$ comparisons:

$$\text{(from line 24) } N_F = q!$$

The calculation of the best Pareto individuals is equivalent to the calculation of the PR2 method. Therefore, from Equation 42:

$$\text{(from line 26) } N_G = \frac{2+2q}{3}|H_O|^3 + |H_O|^2 + \frac{10-2q}{3}|H_O|.$$

The sorting of the ID list requires:

$$\text{(from line 27) } N_H = s + \sum_{j=1}^{s-1} j = s + \frac{s(s-1)}{2} = \frac{1}{2}(s^2+s)$$

At worst case, the loop through all possible orders requires:

$$\text{(from line 30) } N_I = q!$$

$$\text{(from line 31) } N_J = qN_I = q(q!)$$

The number of calculations from the calculation and applications of thresholds can be found from Equations 104 and 105 respectively.

$$\text{(from line 34) } N_K = N_I q|H_O|^2 = q(q!)|H_O|^2$$

$$\text{(from line 35) } N_L = N_I\,(1+3q)\,|H_O| = (q!)\,(1+3q)\,|H_O|$$

The remaining number of calculations for N_L through N_W are:

$$\text{(from line 38) } N_M = N_I|H_O| = (q!)|H_O|$$

$$\text{(from line 39) } N_N = qN_M = q(q!)|H_O|$$

$$\text{(from line 40) } N_O = N_N = q(q!)|H_O|$$

$$\text{(from line 46) } N_P = N_I = q!$$

$$\text{(from line 48) } N_Q = q!\left(\frac{2+2q}{3}|H_O|^3 + |H_O|^2 + \frac{10-2q}{3}|H_O|\right)$$

$$\text{(from line 49) } N_R = q!\left(s + \sum_{j=1}^{s-1} j\right) = (q!)\frac{1}{2}(s^2+s)$$

$$\text{(from line 54) } N_S = N_I 2s = 2s(q!)$$

$$\text{(from line 57) } N_T = N_S = 2s(q!)$$

$$\text{(from line 61) } N_U = N_S = 2s(q!)$$

$$\text{(from line 75) } N_V = N_I = q!$$

$$\text{(from line 77) } N_W = qN_V = q(q!)$$

$$\text{(from line 84) } N_X = q$$

Combining the twenty-three terms:

$$
\begin{aligned}
N_{Ordering}^{Auto} &= N_A + N_B + N_C + N_D + N_E + N_F + N_G + N_H + N_I + N_J + N_K + N_L + N_M \\
&\quad + N_N + N_O + N_P + N_Q + N_R + N_S + N_T + N_U + N_V + N_W + N_X \\
N_{Ordering}^{Auto} &= |H_O|^3 \frac{2\,(q!(q+1)+q+1)}{3} + |H_O|^2\,(q!(q+1)+1) \\
&\quad + |H_O| \frac{(13q+16)q! + 4q + 10}{3} \\
&\quad + s^2 \frac{q!+1}{2} + s \frac{13q!+1}{2} + q!(2q+4) + q(3) + 1
\end{aligned} \tag{109}
$$

The calculation of $N_{Ordering}^{Auto}$ reveals that there are two main drivers for the complexity of this calculation: the $\frac{2}{3}|H_O|^3(q!)q$ originating from the Pareto Rank 2 comparisons and the $|H_O|^2(q!)q$ terms originating from the recalculation of the dynamic objective thresholds for each objective ordering. For example, with values of $|H_O| = 1000$ and $q = 10$ the complexity of $\frac{2}{3}|H_O|^3(q!)q$ term results in $2.4e10^{16}$ comparisons, and the $|H_O|^2(q!)q$ term results in $3.6e10^{13}$ comparisons. Fortunately, Equation 109 is for the worst case scenario, and performance is typically much better than the worst case.

3.5 Conclusions

This chapter details a total of eleven research contributions to the standard evolutionary algorithms processes applicable to four classes of EA optimization problems. The class of problems in order of increasing specialization are:

1. All Multiple Objective Evolutionary Algorithms (MOEA)
2. Problems that are Computationally Expensive (CE)
3. Problems with a Region of Interest (ROI)
4. Problems with Independent Computationally Expensive Objectives (ICEO)

Table 3 provides a brief summary of the eleven research contributions.

Table 3: Summary of Research Contributions

EA Class	Algorithm	Brief Description
MOEA	Genome Encoding	Hierarchical XML description of search space
	Location Crossover	Crossover at selected attribute locations
	Mutation	Association of mutation operators with selected attributes
	Transform Scaler	Transformation of fitness values using Linear Interpolated Data
	Modular Multiple Objective Algorithm	Allows recombination of many MOEA components in novel ways
CE	External Objective Evaluation	Attachment of MOEA to external objective evaluators
	Parallel Evaluation	Removal of synchronization time delay at end of generation
ROI	HCD Scaler	Use of distance of solutions from ROI to scale fitness values
ICEO	Dynamic Objective Thresholding	Truncates evaluation of individuals before all objectives are evaluated
	HCD Dynamic Objective Ordering	Use of average distance of solutions from ROI to order objectives
	Auto Dynamic Objective Ordering	Use of time required to evaluate objectives to determine order of objectives

Of the research innovations, the transform scaler, HCD scaler, calculation of thresholds, application of thresholds, HCD objective ordering, and the auto objective order algorithms all modify the calculation of the fitness values. Thus, in the respective section of this chapter there was a calculation of the number of comparisons. Table 4 gives an aggregate of the complexity calculations for various MOEA components given in Table 2 with the addition of the complexity calculations for these research innovations. The definitions of q as the number of objectives, $|P|$ as the number of individuals in the current population, $|E|$ as the number of individuals in the elite population, the definitions of $|LID|$ as the number of points in the linear interpolated data transformation list, $|H_T|$ as the number of individuals in historic threshold population, $|H_O|$ as the number of individuals in historic ordering population, and s as the number of "best" individuals to compare between the current objective ordering and proposed objective orders are used.

Table 4: Number of the Comparisons for Various MOEA Components

Algorithm	Number of Comparisons
Pareto Optimality	$(2+2q)\|P\|^2-(1+2q)\|P\|$
Pareto Rank 1	$(2+2q)\|P\|^2-(1+2q)\|P\|$
Pareto Rank 2	$\frac{2+2q}{3}\|P\|^3+\|P\|^2+\frac{10-2q}{3}\|P\|$
Fitness Sharing	$(2+q)\|P\|^2-(1+q)\|P\|$
Objective Scaling	$(2+4q)\|P\|-(1+2q)$
SPEA Clustering	$\frac{1}{30}(1+q)\|P\|^5+\left(\frac{1}{12}q+\frac{1}{6}\right)\|P\|^4+\left(\frac{5}{6}\right)\|P\|^3+\left(-\frac{1}{12}q+\frac{1}{3}\right)\|P\|^2$ $+\left(\frac{29}{30}q+4\frac{19}{30}\right)\|P\|-2q$ $+\left(-\frac{1}{5}(1+q)\right)\|E\|^5+\left(-\frac{3}{4}q-\frac{1}{2}\right)\|E\|^4+\left(-\frac{5}{6}q-\frac{2}{3}\right)\|E\|^3$ $+\left(-\frac{1}{4}q-1\right)\|E\|^2+\left(\frac{1}{30}q+\frac{11}{30}\right)\|E\|$ $+\|P\|^2\left(-\frac{1}{3}(1+q)\|E\|^3-(1+q)\|E\|^2-\frac{2}{3}(1+q)\|E\|\right)$ $+\|P\|\left(\frac{1}{2}(1+q)\|E\|^4+\left(\frac{5}{3}q+\frac{4}{3}\right)\|E\|^3\right)$ $+\|P\|\left(\left(\frac{3}{2}q+\frac{1}{2}\right)\|E\|^2+\left(\frac{1}{3}q-\frac{1}{3}\right)\|E\|\right)$
SPEA Strength	$-(2+2q)\|E\|^2+\|E\|+\|P\|(2+2q)\|E\|$
SPEA Fitness	$\|P\|-(2+2q)\|E\|^2-\|E\|+\|P\|(2+2q)\|E\|$
SPEAII Raw Fitness	$(4+4q)\|P\|^2-(2+4q)\|P\|$
SPEAII Density	$\|P\|^3+(1+q)\|P\|^2-(1+q)\|P\|$
Transform Scaler	$\|P\|\,(2+3(\|LID\|-1))$
HCD Scaler	$(1+3q)\,\|P\|$
Threshold Calculation	$q\|H_T\|^2$
Threshold Application	$(1+3q)\,\|H_T\|$
HCD Objective Ordering	$4\|H_O\|q+q^2+2q-1$
Auto Objective Ordering	$\|H_O\|^3\frac{2(q!(q+1)+q+1)}{3}+\|H_O\|^2\,(q!(q+1)+1)$ $+\|H_O\|\frac{(13q+16)q!+4q+10}{3}$ $+s^2\frac{q!+1}{2}+s\frac{13q!+1}{2}+q!(2q+4)+q(3)+1$

With a detailed discussion of the eleven research contributions, the next three chapters of this thesis explore the use and impact of these contributions on a set of diverse problems: the 0/1 knapsack problem and Deb's T functions in Chapter 4, the flare pattern design problem in Chapter 5, and the warning receiver optimization problem in Chapter 6. Each of these research innovations is not applicable to all of the problems.

Although the genome encoding is utilized for all problems, it is not required by the 0/1 knapsack and Deb's T function problems. The genome encoding is utilized by the flare pattern design and warning receiver optimization problems in order to implement named

access to crossover and mutation operators. Likewise, the ability to associate mutation and crossover operators with named attributes of the genome is not required by the 0/1 knapsack problem or the Deb's T function problems but is used by the flare pattern and warning receiver optimization problems.

Chapter 4 provides an evaluation of the 0/1 knapsack and Deb's T functions against a myriad of the combinations of various MOEA techniques in an effort to find the combinations that have the best performance. This quantitative analysis tests the capabilities of the transform scaler and modular MOEA design research contributions. The recommended combinations are then utilized in the flare pattern design and warning receiver optimization chapters.

The use of the external objective evaluation and parallel evaluation research contributions is only required for computationally expensive objectives and thus utilized in the flare pattern design and warning receiver optimization chapters.

The HCD scaler and dynamic objective thresholding algorithm is evaluated with the 0/1 knapsack and Deb's T functions problems and then extensively utilized by the flare pattern design and warning receiver design problems. The HCD dynamic and Auto ordering techniques are evaluated with flare pattern design problem. Table 5 provides a summary of the section where each research contribution is detailed and where each contribution is evaluated and utilized.

Table 5: Locations of Evaluation and Utilization of Research Contributions

EA Class	Research Contribution	Section Describing Algorithm	Description	0/1 Knapsack and Deb's T Function Evaluation (Chapter 4)	Flare Pattern Evaluation (Chapter 5)	0/1 Knapsack and Deb's T Function Utilization (Chapter 4)	Flare Pattern Utilization (Chapter 5)	Warning Receiver Utilization (Chapter 6)
MOEA	Genome Encoding	3.1.1	Hierarchical XML search space description				X	X
	Location Crossover	3.1.2	Crossover at selected attribute locations				X	X
	Mutation Specification	3.1.3	Association of mutation operators with selected attributes				X	X
	Transform Scaler	3.1.4	Transformation of fitness using LID	X	X	X	X	X
	Modular Multiple Objective Algorithm	3.1.5	Allows recombination of many MOEA components in novel ways	X	X	X	X	X
CE	External Objective Evaluation	3.2.1	Attachment of MOEA to external objective evaluators				X	X
	Parallel Evaluation	3.2.2	Removal of synchronization time delay at end of generation		X		X	X
ROI	HCD Scaler	3.3	Fitness scaled by HCD to ROI	X	X	X	X	X
ICEO	Dynamic Objective Thresholding	3.4.1	Truncates evaluation of individuals before all objectives are evaluated	X	X	X	X	X
	HCD Dynamic Objective Ordering	3.4.2.1	Use of average distance of solutions from ROI to order objectives	X	X	X	X	
	Auto Dynamic Objective Ordering	3.4.2.2	Use of time required to evaluate objectives to determine order of objectives		X	X	X	

CHAPTER IV

ALGORITHM EVALUATION

Chapter 1 introduced the definitions of single objective optimization, multiple objective optimization, search space, objective space, Pareto optimality, and a region of interest in the objective space. Chapter 2 added detailed descriptions for the EA operators of fitness evaluation, selection, mating and mutation, as well as the basics of genome encoding and parallel evaluation. Chapter 2 also presented the details of several of the existing MOEA methods. Chapter 3 presented the details of the algorithms contributed by this thesis.

Given the foundation of the previous chapters, this chapter presents quantitative analysis of the performance of various combinations of the existing and new MOEA components for reaching a region of interest in objective space against a fast running problem before they are applied to computationally expensive problems in the remaining chapters.

Section 4.1 provides quantitative analysis against the 0/1 Knapsack problem extended to multiple dimensions of Martello and Toth [42]. Because the 0/1 Knapsack problem does not contain independent objectives, nor is it computationally expensive, the number of objectives required is minimized instead of minimizing the time required to evolve. This section is divided into Subsection 4.1.1 detailing the 0/1 Knapsack problem, Subsection 4.1.2 itemizing the combinations of algorithm components to be evaluated, Subsection 4.1.3 discussing the evaluation of the ICEO algorithm components, and Subsection 4.1.4, which summarizes the results of the quantitative analysis.

Section 4.2 then takes those combinations of algorithm components that are found to be effective in Section 4.1.4 and evaluates them against the six MOPs developed by Deb [17] to ensure that the combinations are effective against a robust set of problems. This section is divided into six subsections; one for each of the T functions.

Section 4.3 provides a summary of the recommended MOEA combinations for ICEO problems. These recommended MOEA combinations are then further evaluated against the ICEO flare pattern problem in Chapter 5 and utilized for the optimization of AAR-44A OFP in Chapter 6.

4.1 0/1 Knapsack Problem

The 0/1 Knapsack problem described by Martello and Toth in 1990 [42] has several desirable features. First, it is a non-trivial NP-hard problem, and as such can be configured to require evaluation of over one hundred thousand individuals to converge. The problem is repeatable, easy to understand, and easy to formulate. The 0/1 Knapsack problem can also be scaled to any desired number of search space and objective space dimensions, and provides a quick-running analysis with published results for some existing MOEA algorithms [61, 64, 63].

4.1.1 Description of 0/1 Knapsack Problem

In the single objective case of the 0/1 Knapsack problem, there is a single knapsack with capacity c and a set of n items. Each item j has an associated profit p_j and a weight w_j. The problem is then to find the vector $x = (x_1, x_2, \cdots, x_n) \in \{0, 1\}^n$ where $x_j = 1$ implies item j is in the knapsack, and likewise $x_j = 0$ implies item j is not in the knapsack. This vector x must satisfy the capacity constraint that the total weight of the items in the knapsack does not exceed the capacity of the knapsack:

$$e(x) = \sum_{j=1}^{n} w_j \cdot x_j \leq c. \tag{110}$$

The fitness of the individual x for each objective i is then defined as the total profit of the items in the knapsack:

$$f(x) = \sum_{j=1}^{n} p_j \cdot x_j. \tag{111}$$

Figure 41 illustrates an example of the single objective 0/1 Knapsack with four items.

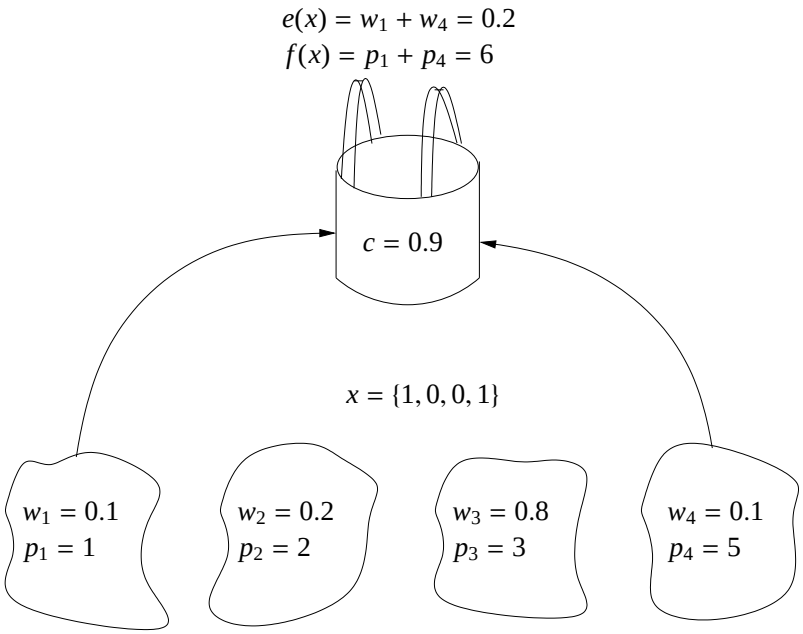

Figure 41: Single Objective Example of 0/1 Knapsack Problem

The 0/1 Knapsack problem is extended to multiple objectives by increasing the number of knapsacks from one to the number of desired objectives q. Each knapsack has a capacity c_k. Each item is assigned a set of profits $p_{k,j}$ and weights $w_{k,j}$ for item j in knapsack k. In the multiple objective case when the j^{th} member of the vector x is 1, $x_j = 1$, item j is in *all* knapsacks, and likewise $x_j = 0$ implies item j is *not in any* knapsacks. The vector x must satisfy the capacity constraint of all knapsacks ($1 \leq k \leq q$).

$$e_k(x) = \sum_{j=1}^{n} w_{k,j} \cdot x_j \leq c_k \tag{112}$$

The fitness of the individual x is then defined as:

$$f_k(x) = \sum_{j=1}^{n} p_{k,j} \cdot x_j. \tag{113}$$

Figure 42 illustrates an example of the two objective 0/1 Knapsack problem with four items. The size of the search space can be scaled by increasing the number of items. Likewise the size of the objective space can be increased by increasing the number of knapsacks.

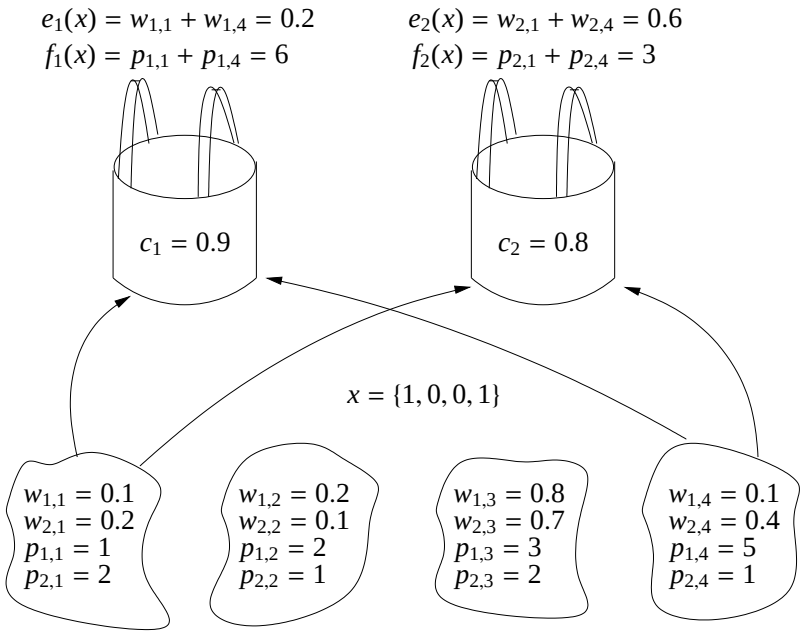

Figure 42: Multi-Objective Example of 0/1 Knapsack Problem

The feasibility region of the search space is limited by the constraints of each knapsack. One method of handling the condition that an individual violates one of the constraints would be would be to mark the individual as fatal. Instead the 0/1 Knapsack problem uses a method of "genome correction". Genome correction is performed by removing the item with the minimum profit to weight ratio, $p_{k,j}/w_{k,j}$ for the knapsacks with exceeded capacity from *all* knapsacks. This method for removing items is repeated until the capacity of *all* knapsacks is no longer exceeded.

The setting for the profits, weights and constraints are performed the same as Zitzler [61] who followed the suggestions of Martello and Toth [42]. Uncorrelated profits and weights are selected as random integers in the interval $[10, 100]$. The knapsack capacities are then set half of the total possible weight in the knapsack:

$$c_k = \frac{1}{2} \sum_{j=1}^{n} w_{k,j}. \tag{114}$$

The actual weights used for the two knapsack 250 item problem and the four knapsack 750 item problem are downloaded from Zitzler's web-site to maximize correlation with his

results. The weights, profits and constraints for the ten knapsack 1000 item problems are derived using the above methods.

4.1.2 Combinations of MOEA Components for 0/1 Knapsack Problem

Chapters 2 and 3 presented currently used and proposed MOEA components. Section 3.1.5 detailed the modular framework that has been developed for components in novel ways. Four categories of operations effect the probability of mating: the calculations of the "raw" fitness operation, the scaling of the fitness values, the selection operator, and the rules for creating the elite population. This section evaluates the characteristics of many of the possible combinations to identify the combinations that should be further examined in the following sub-sections using the ICEO dynamic objective thresholding and dynamic objective ordering.

4.1.2.1 Possible Combinations

Table 6 displays the components examined for each of the four categories of operations Five "raw" fitness solver functions are analyzed: the PR1, PR2, Weighted-Sum, SPEA, and SPEA II. Three fitness scaling functions are also analyzed: the transform scaler, the SPEA II density algorithm, and the fitness sharing algorithm of the Niched Pareto Genetic Algorithm. Two selection operators are investigated; fitness proportionate selection and binary tournament selection. Performance with and without elitism is also examined.

Table 6: Categories of MOEA Components Examined

Fitness	**Fitness Scaler**	**Selection**	**Elitism**
Pareto Rank 1	Transform	Fitness Proportionate	Maximum Rank
Pareto Rank 2	SPEA2 Density	Tournament	No Elitism
Weighted Sum	Fitness Sharing		
SPEA			
SPEAII			

Two transformations of “raw” fitness values using linear interpolated data are supported. Both transformations invert fitness values so that small values, which indicate the

best individuals from Pareto rank techniques, are transformed to the largest values and thus are acceptable for fitness proportionate as well as binary tournament selection. Transformation 1 contains samples of the function

$$T_1(x) = \frac{1}{2^{(x-1)}}. \tag{115}$$

Transformation 2 contains samples of the function

$$T_2(x) = \frac{1}{x}. \tag{116}$$

Table 7 gives the data used for the LID transformations. Figure 43 illustrates the two LID transformations.

Table 7: Linear Interpolated Data Transformations

Transform 1	
Input	**Output**
0.0	10.0
1.0	1.0
2.0	0.5
3.0	0.25
4.0	0.125
5.0	0.0625
6.0	0.03125
7.0	0.015625
8.0	0.0078125
21.0	9.53e-7
41.0	9.09e-13
1000.0	1e-31

Transform 2	
Input	Output
0.0	10.0
1.0	1.0
2.0	0.5
3.0	0.3333
4.0	0.25
5.0	0.2
6.0	0.16666
8.0	0.125
9.0	0.11111
10.0	0.1
20.0	0.05
100.0	0.01
1000.0	0.001

The fitness sharing algorithm allowed for thirty niches. Runs are executed without elitism and with elitism based on the maximum rank. If there are still too many individuals in the elite population after removing those with a larger rank than specified, then those individuals with the poorest fitness values are removed until the number of individuals in the elite pool meets the desired maximum number. A maximum rank of one only allows Pareto optimal individuals in the elite population. Maximum rank values from one to thirteen are examined.

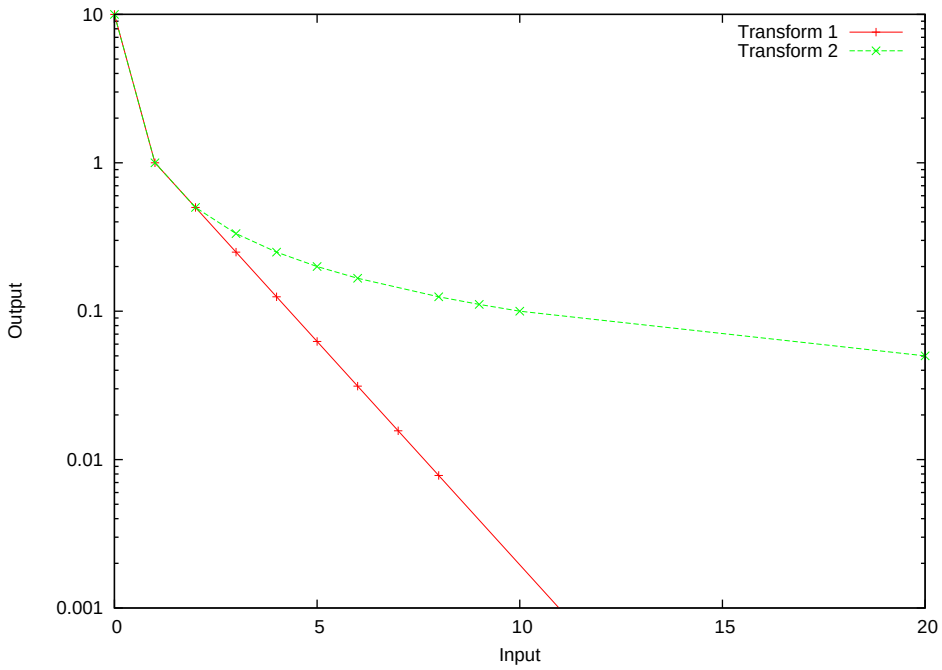

Figure 43: Linear Interpolated Data Transformations

4.1.2.2 MOEA Setup

The evaluation of the performance is made using the 0/1 Knapsack problem with two knapsacks, i.e., two objectives, and 250 items, i.e., 250 attributes in the search space. For each run 60, 000 individuals are evaluated for a total of 120, 000 objectives. Results are given for the number of objectives evaluated as opposed to the number of individuals to allow comparison with the ICEO methods in later sections.

All runs are executed with 120 individuals per generation and are initialized using 150 random individuals. For this analysis generation synchronization is enabled. A maximum size for the elite population is set at thirty individuals.

Single point crossover is utilized with a probability of eighty percent. The remaining twenty percent of individuals are created with the clone crossover, i.e., no crossover. Bitwise mutation is utilized with an independent probability of 0.25% per bit. Since each of the 250 bits corresponds to a separate item in the knapsack, a mutation or flip of the bit places an item in or out of the knapsack. The probability of mutation for an individual is

$1.0 - (1.0 - 0.0025)^{250} = 46.5\%$

A total of thirty runs are made for each MOEA combination being evaluated. By using a different specification for the random number seed, each of the thirty runs contains a different initial population.

The evaluation of performance is made by measuring the hypervolume of the resulting Pareto optimal front. The hypervolume is given after evaluation of each $10,000$ objectives. Three statistics are gathered for the results of all thirty runs: the maximum hypervolume, the minimum hypervolume, and the median hypervolume.

4.1.2.3 Pareto Rank 1 and 2 Combinations

Runs with raw fitness values from the Pareto rank 1 and Pareto rank 2 fitness methods provide the largest number of permutations. Since both of these fitness options give the best fitness to the smallest fitness values, the fitness values may or may not be scaled using the SPEAII density transformation. These fitness values are inverted using either inverting transformation. Once inverted, the fitness values may or may not be modified using the fitness sharing algorithm of the NPGA. These final fitness values are used by either of the fitness proportionate or binary tournament selection methods. If elitism is used, the fitness values are used to find the elite population of the next generation using a maximum rank of one to thirteen. These possibilities result in a total of 448 combinations. Figure 44 illustrates these combinations.

Figure 45 compares the hypervolume after the evaluation of each set of $10,000$ objectives for the fourteen different elitism possibilities. Error-bars are used to illustrate the maximum, minimum and median values at each interval. Because there are thirty runs for each combination and there are thirty-two variations for the fitness, fitness scalers and selection types for each possibility, the error-bars aggregate the results of 960 runs. The most distinct performance is that of the runs without elitism. Without elitism, the minimum and median performance is noticeably below that of the runs with elitism. Once elitism is

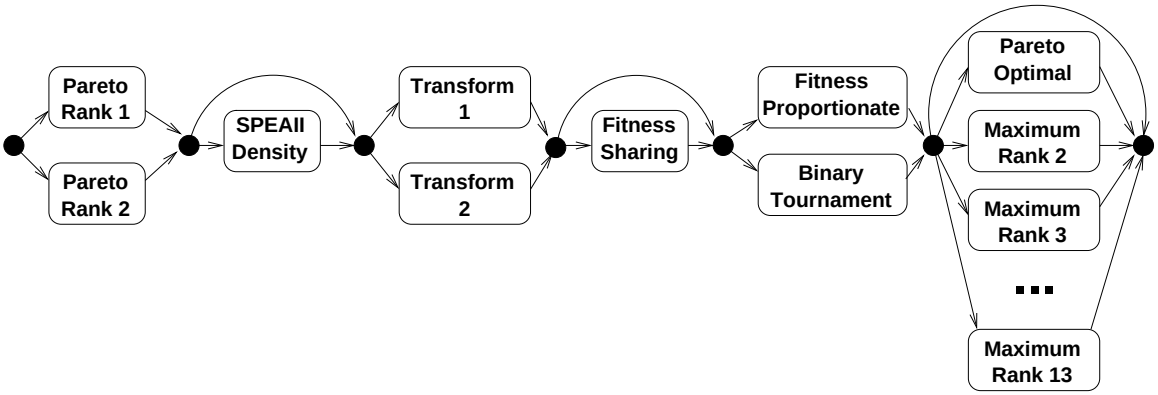

Figure 44: Pareto Rank 1 and Pareto Rank 2 Fitness Combinations

applied the results are less obvious. But, inspection shows that elitism with Pareto optimal individuals and elitism with Pareto rank 2 individuals contain a lower median value than elitism with a maximum Pareto rank of three to fourteen. Elitism with a maximum Pareto rank of three to fourteen shows very little difference in the median values, and the minimum and maximum values do not seem to be consistently better or worse as the maximum Pareto rank increases beyond three.

It is possible to use zero, one or two fitness scalers that attempt to adjust the fitness values based on the density of solutions. The scalers are the SPEA II density scaler and the NPGA fitness sharing scaler. Since each of these can be utilized separately or together there are a total of four combinations for these density scaling algorithms. Figure 46 compares the results for each of these combinations. The error-bars indicate the results of the thirty runs for each of the 112 combinations of the variations of fitness, transform scaler, selection and elitism. Thus, the error-bars indicate the results of 3360 runs. From the figure, note that without either density scaling method, the median and maximum values are consistently lower than with density scaling. The figure also reveals that the NPGA fitness sharing algorithm performs better for the maximum, minimum and average than the SPEA II density scaling algorithms for all but the first $20,000$ objectives. When the SPEA II density scaling algorithm is combined with the NPGA fitness sharing algorithm, there is no

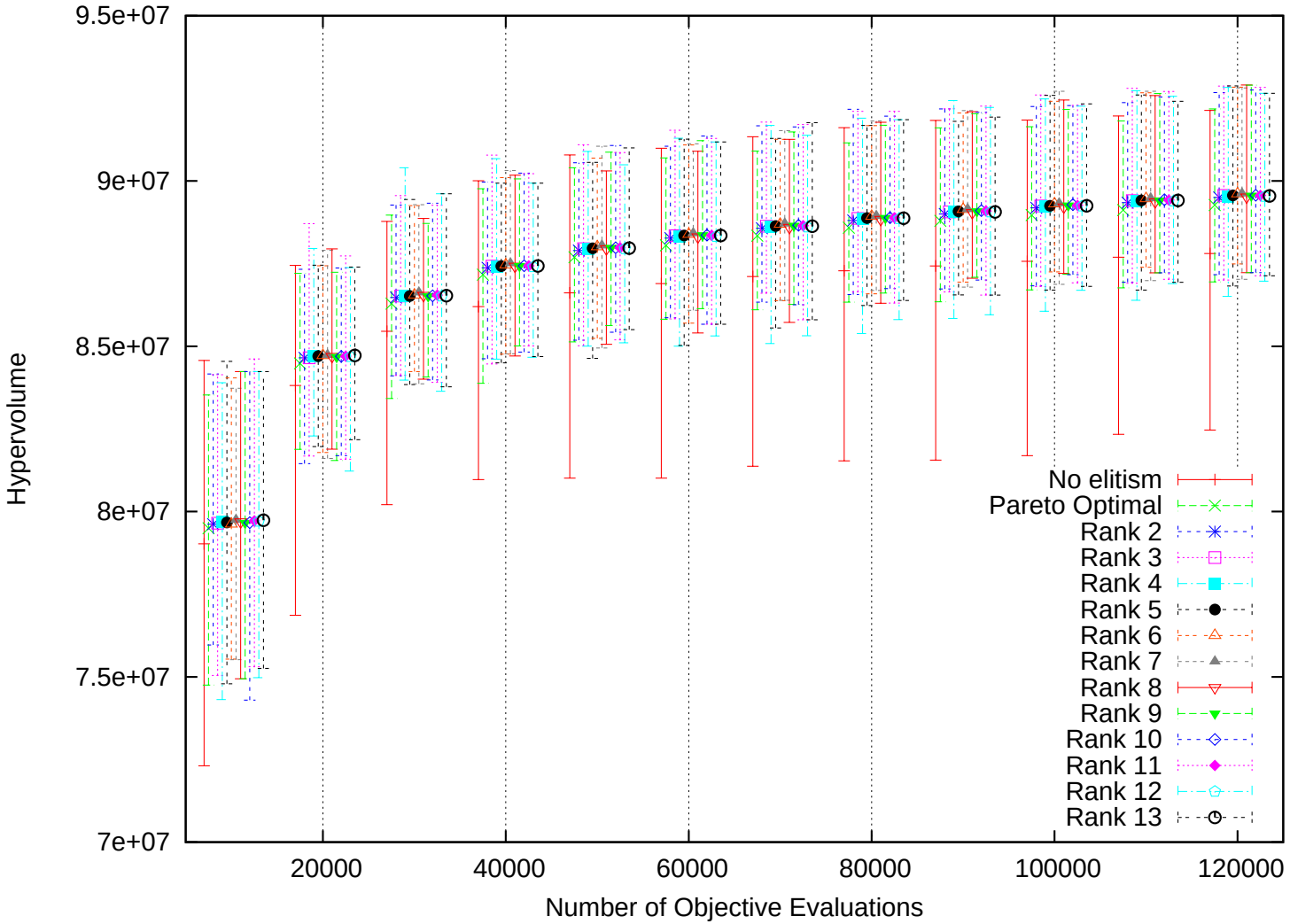

Figure 45: Comparison of Elitism Methods for Pareto Rank Combinations

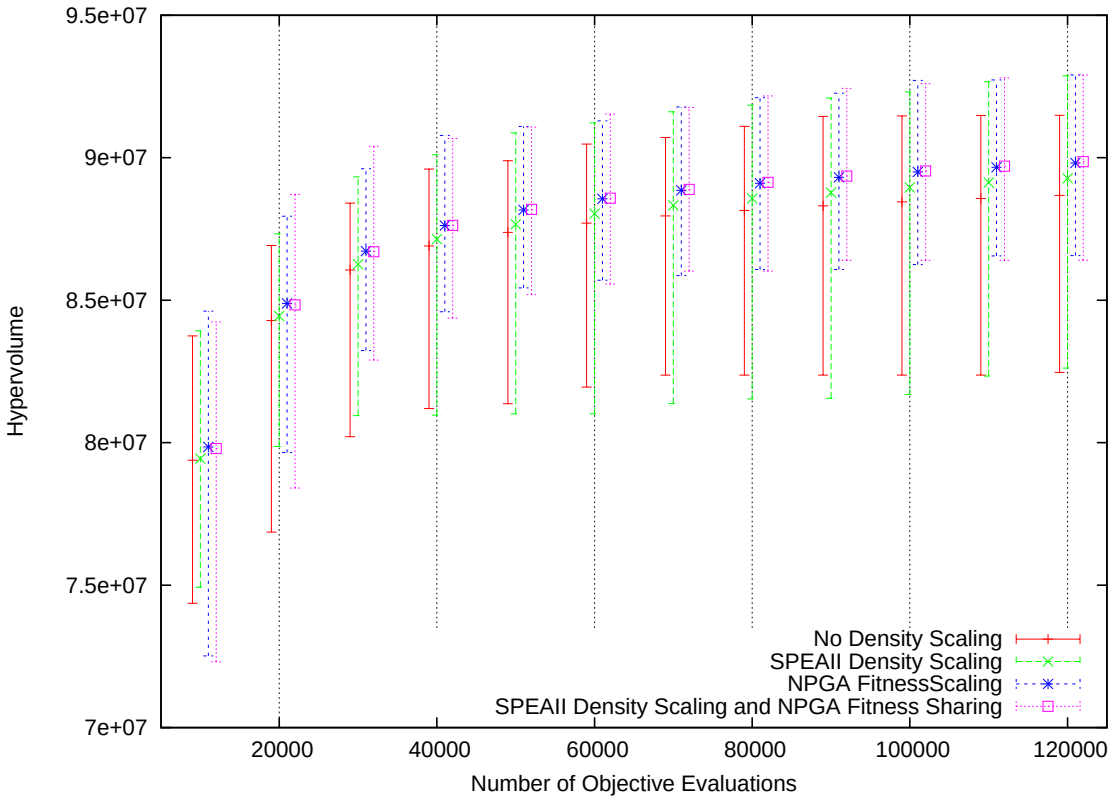

Figure 46: Comparison of Density Methods for Pareto Rank Combinations

appreciable difference in the statistics from those of the NPGA fitness sharing algorithm alone.

Figure 47 compares the results of all runs using fitness proportionate selection with those using binary tournament selection. The error-bars indicate the results of the thirty runs for each of the 224 combinations of the variations of fitness, fitness scalers, and elitism methods, for a total of 6720 runs. The figure illustrates that the fitness proportionate selection method performs better than the binary tournament selection method for the first $30,000$ objectives, but after that point the median and minimum values of the fitness proportionate selection method are consistently below that of the binary tournament selection method. But, the maximum value of the fitness proportionate selection method outperforms that of the binary tournament selection method for eleven of twelve sets of $10,000$ objectives. The minimum value of the binary tournament selection method significantly

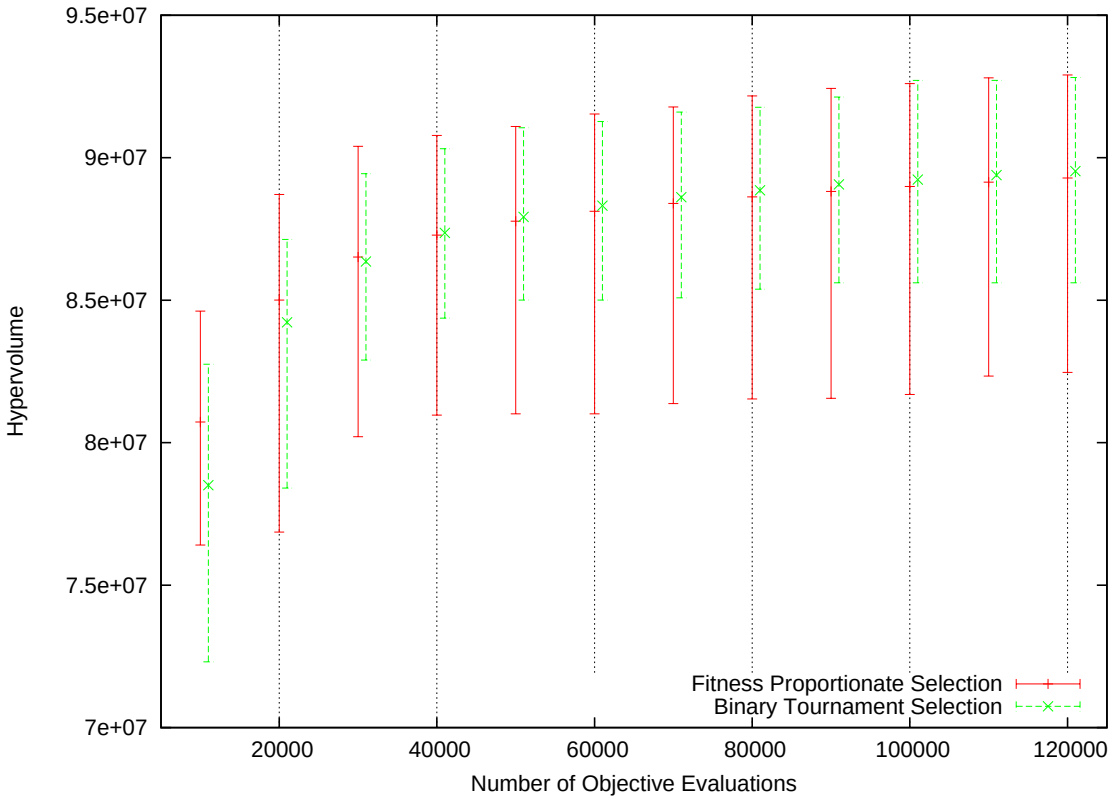

Figure 47: Comparison of Selection Methods for Pareto Rank Combinations

outperforms that of the fitness proportionate selection method for eleven of twelve sets of 10, 000 objectives, and it contains a much smaller deviation in results.

The purpose of the analysis of the 448 combinations is to determine the combinations of algorithms that should be further analyzed with ICEO techniques. Because the class of problems of interest are computationally expensive, there is rarely a chance to make multiple optimization runs. Thus, the most applicable statistic to use when comparing the results of the thirty runs is the minimum performance. To select a set of diverse algorithms, the four combinations of fitness, i.e., Pareto rank 1 and Pareto rank 2, and transformation, i.e., LID transformation 1 or transformation 2 from equations 115 and 116, are analyzed for each combination.

To further insure diversity, combinations that perform best during the beginning of optimization as well as those that perform best during the end of the optimization are desired. This diversity is accomplished by selecting the top two performers, i.e., those with the largest minimum hypervolume, for the samples collected after the first three and last three sets of $10,000$ objectives. Figures 48, 49, 50, and 51 illustrate the combinations that provide the largest minimum hypervolume for these four combinations of fitness and transformation. The key of these figures reduces the verbosity of the description with the acronyms listed in Table 8. As indicated on the four figures, for each of the combination's two representative methods are selected for further investigation with ICEO methods in Section 4.1.3. For each of the four combinations, one of the two selected methods utilizes binary tournament selection while the other utilizes fitness proportionate selection.

Table 8: List of Acronyms for Combinations of Algorithms

Acronym	Description
PR1	Pareto Rank 1 Fitness Method
PR2	Pareto Rank 2 Fitness Method
WS	Weighted-Sum Fitness method
SPEA	Strength Pareto Evolutionary Algorithm Fitness Method
SPEAII	Strength Pareto Evolutionary Algorithm II Fitness Method
SD	SPEAII Density Fitness scaler
T1	LID Transform scaler 1 based on Equation 115
T2	LID Transform scaler 2 based on Equation 116
FS	NPGA Fitness Sharing
FPS	Fitness Proportionate Selection
BTS	Binary Tournament Selection
MR1	Pareto optimal Elitism
MR2-MR14	Maximum Rank 2 through Maximum Rank 14 Elitism

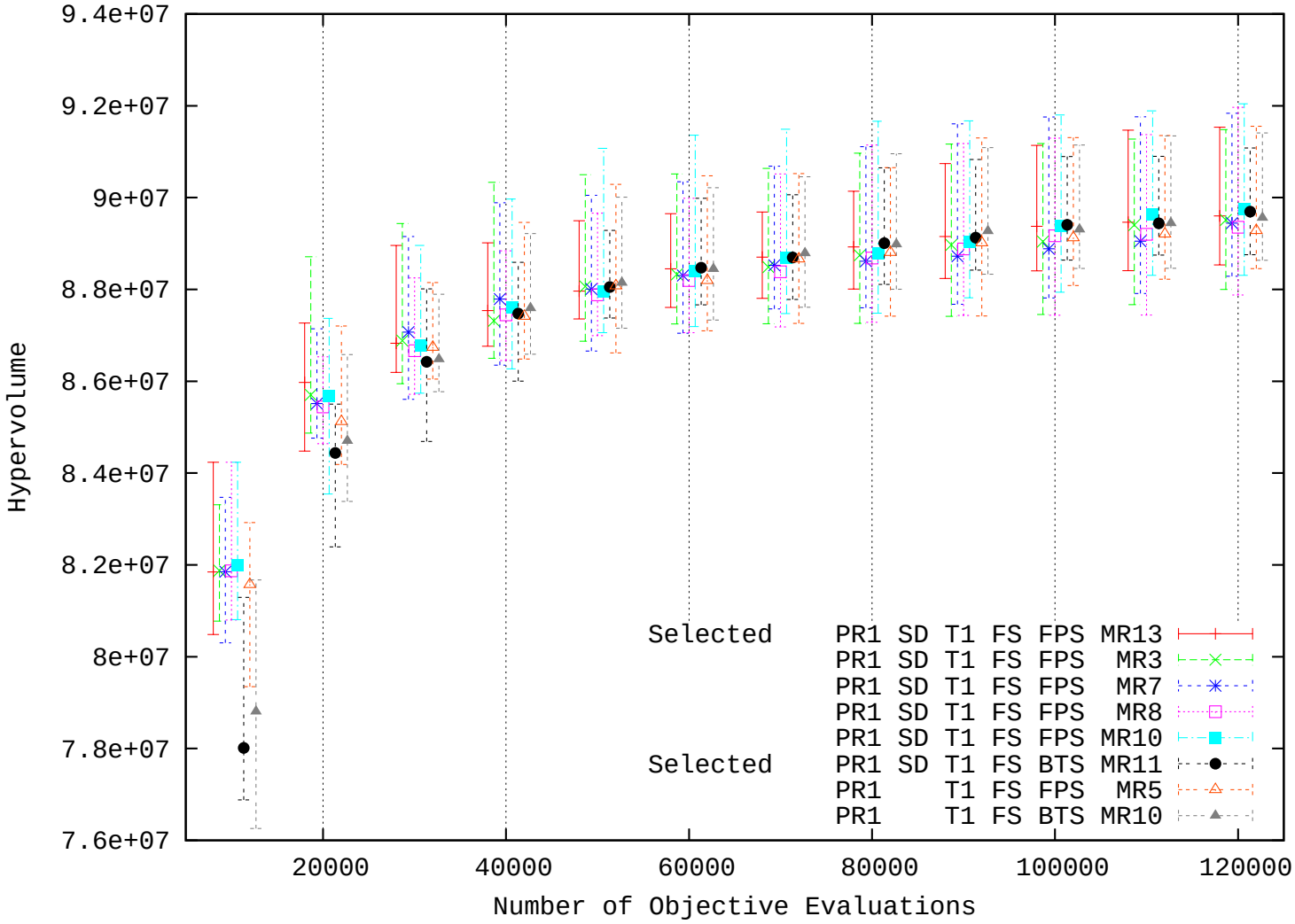

Figure 48: Hypervolume for Pareto Rank 1, Transformation 1 Methods that Provide Largest Minimum Hypervolume

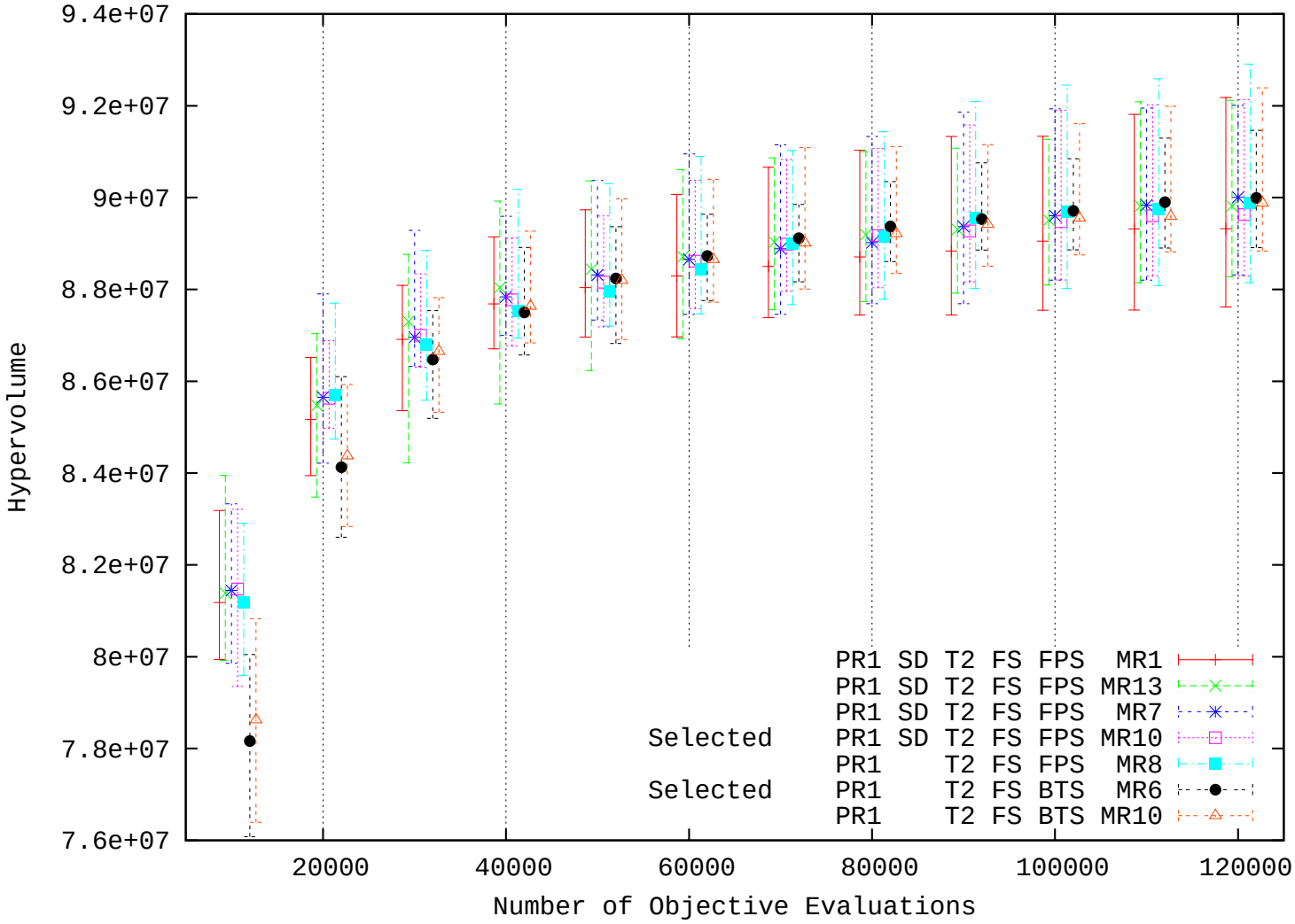

Figure 49: Hypervolume for Pareto Rank 1, Transformation 2 Methods that Provide Largest Minimum Hypervolume

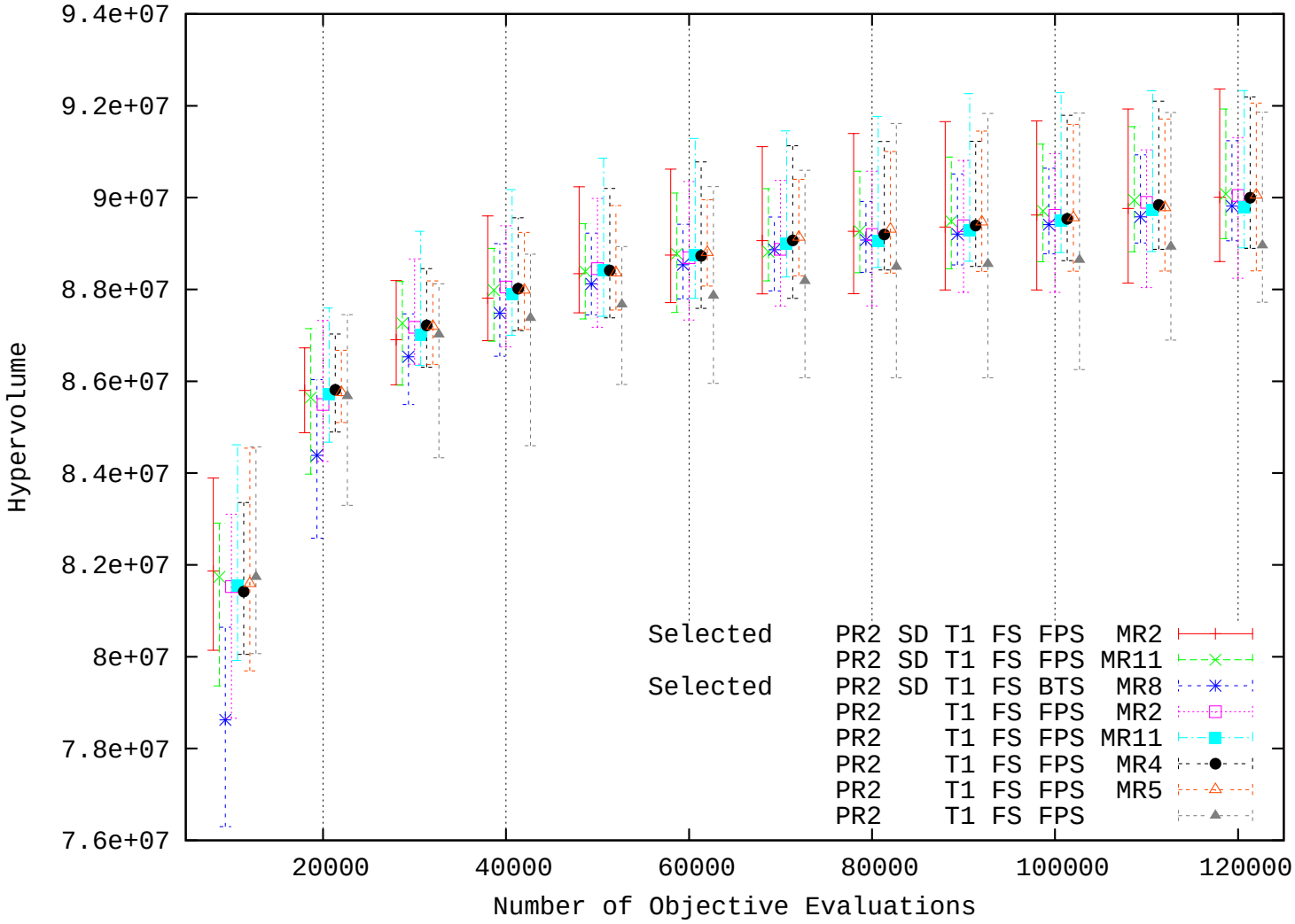

Figure 50: Hypervolume for Pareto Rank 2, Transformation 1 Methods that Provide Largest Minimum Hypervolume

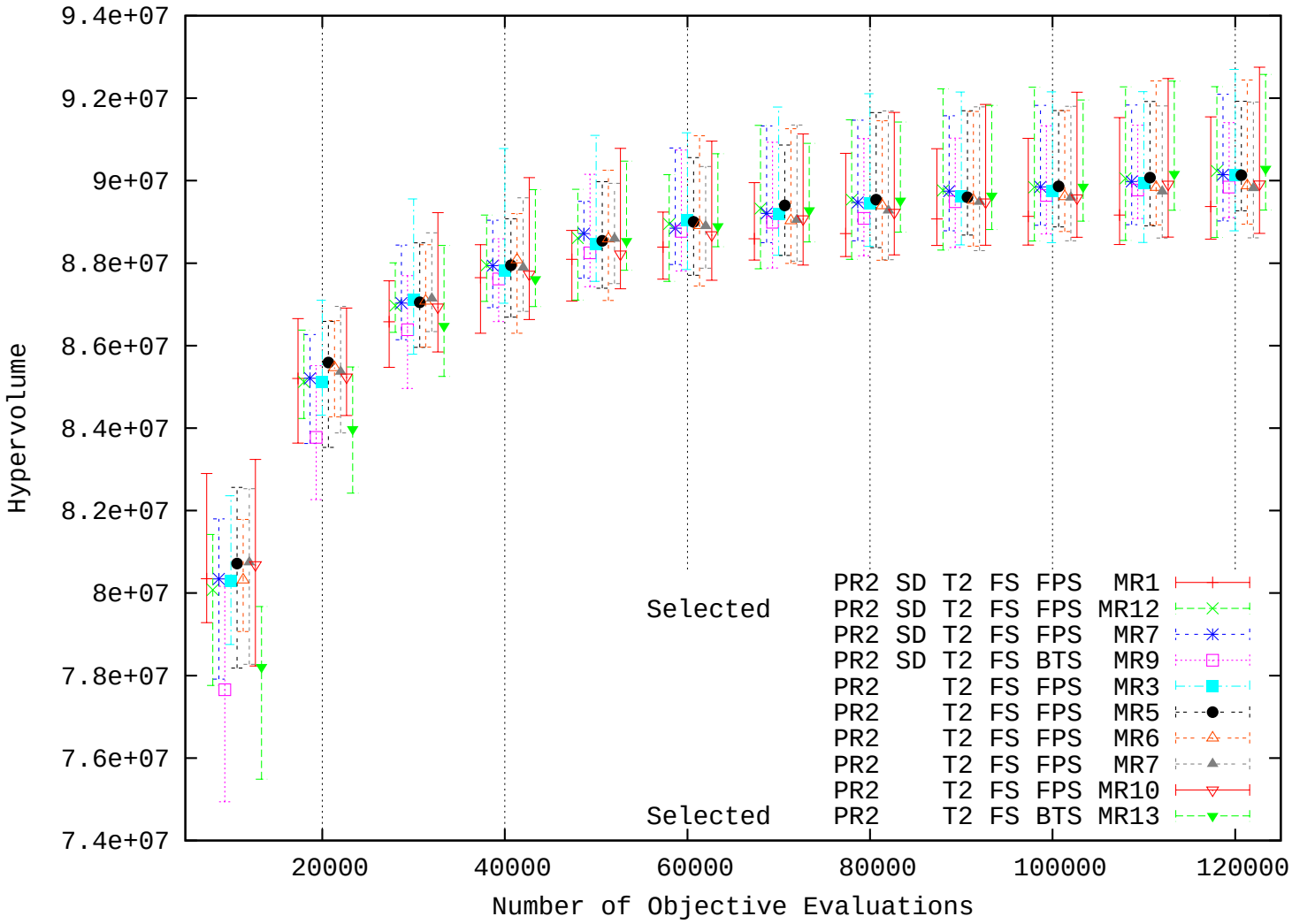

Figure 51: Hypervolume for Pareto Rank 2, Transformation 2 Methods that Provide Largest Minimum Hypervolume

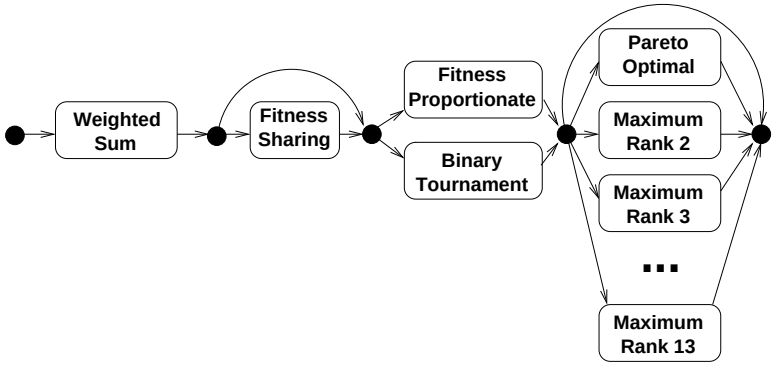

Figure 52: Weighted-Sum Fitness Combinations

4.1.2.4 Weighted-Sum Combinations

The weighted-sum method is utilized with the weight for each objective set to 1.0. Unlike the Pareto rank fitness measure that assigns small fitness values to the preferred individuals, the weighted-sum method assigns large fitness values to the best individuals. The magnitude of the raw fitness values from the weighted-sum are also dependent on the magnitude of the objective values. Therefore, any inversion function that would translate the values for the SPEA II density scaler would be dependent on the function being evaluated. The remaining components, i.e., NPGA fitness sharing, selection and elitism options, are the same as those of the Pareto rank combinations. This results in fifty-six combinations as illustrated in Figure 52.

Figure 53 illustrates the effects of elitism on the hypervolume after the evaluation of each $10,000$ objectives. Because there are thirty runs for each combination and there are eight variations for the fitness, fitness scalers and selection types for each possibility, the error-bars aggregate the results of 240 runs. The most noticeable effect is the abysmal minimum and median performance of the runs without elitism. Once elitism is applied the results show that elitism with up to maximum rank of four further improves the performance. But, after $50,000$ objectives there is a slight decrease in performance for those runs with a maximum rank above four. A maximum rank of four, resulting in improved performance, is larger than the maximum rank of two found for the Pareto rank methods in

Figure 53. These differences must be due to the inherent difference in the raw fitness values. The Pareto rank method raw fitness values are set to the Pareto rank resulting in a large deviation between Pareto optimal and non-Pareto optimal individuals. Weighted-sum raw fitness values are based on the sum of objectives, i.e., the distance from a hyperplane perpendicular to the specified weight vector, resulting in very little difference in fitness values between Pareto optimal and non-Pareto optimal individuals. Therefore, Pareto optimal and near Pareto optimal individuals can only increase their probability of mating dramatically through elitism, which *does* include the concept of Pareto optimality.

There are two options for density scaling: with and without NPGA fitness scaling. There are a total of twenty-eight combinations of the selection and elitism operators. With thirty runs for each combination, the error-bars represent a total of 840 runs. Figure 54 displays the results with and without NPGA scaling. In contrast to the Pareto rank methods, the effect of the NPGA fitness sharing algorithm results in poorer performance for all three statistics in all but the final four error-bars where the maximum is better with fitness sharing. Again, this contrasting result must be attributable to the relatively small variations in the raw-fitness values. With fitness sharing, a cluster of the best two individuals residing farthest from the perpendicular hyperplane will be reduced to half fitness of their raw fitness values, and thus halve the probability of mating. The probability of mating for these two individuals is the same as solutions that are only half as far from the perpendicular hyperplane. With Pareto rank fitness methods, the raw fitness values of those leading individuals would be reduced to be equivalent to Pareto rank two individuals, which are much closer to the Pareto optimal front than those half the distance from the hyperplane.

There are two options for selection: fitness proportionate selection and binary tournament selection. There are a total of twenty-eight combinations of the density scaling and elitism operators. With thirty runs for each combination, the error-bars represent a total of 840 runs. Figure 55 displays the results for the two selection options. These results are also different from the Pareto rank methods. The resulting statistics are much closer, with the

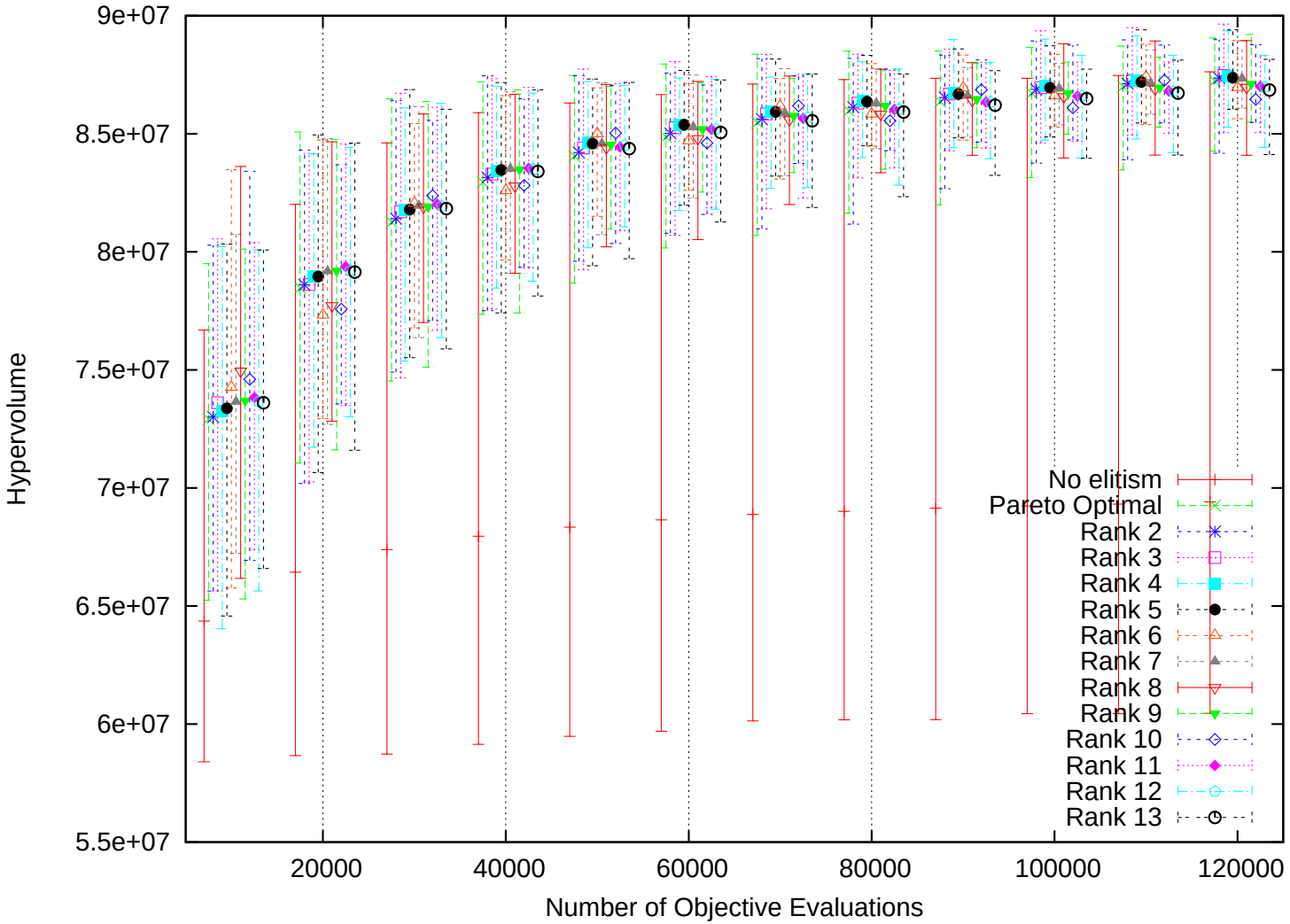

Figure 53: Comparison of Elitism Methods for Weighted-Sum Combinations

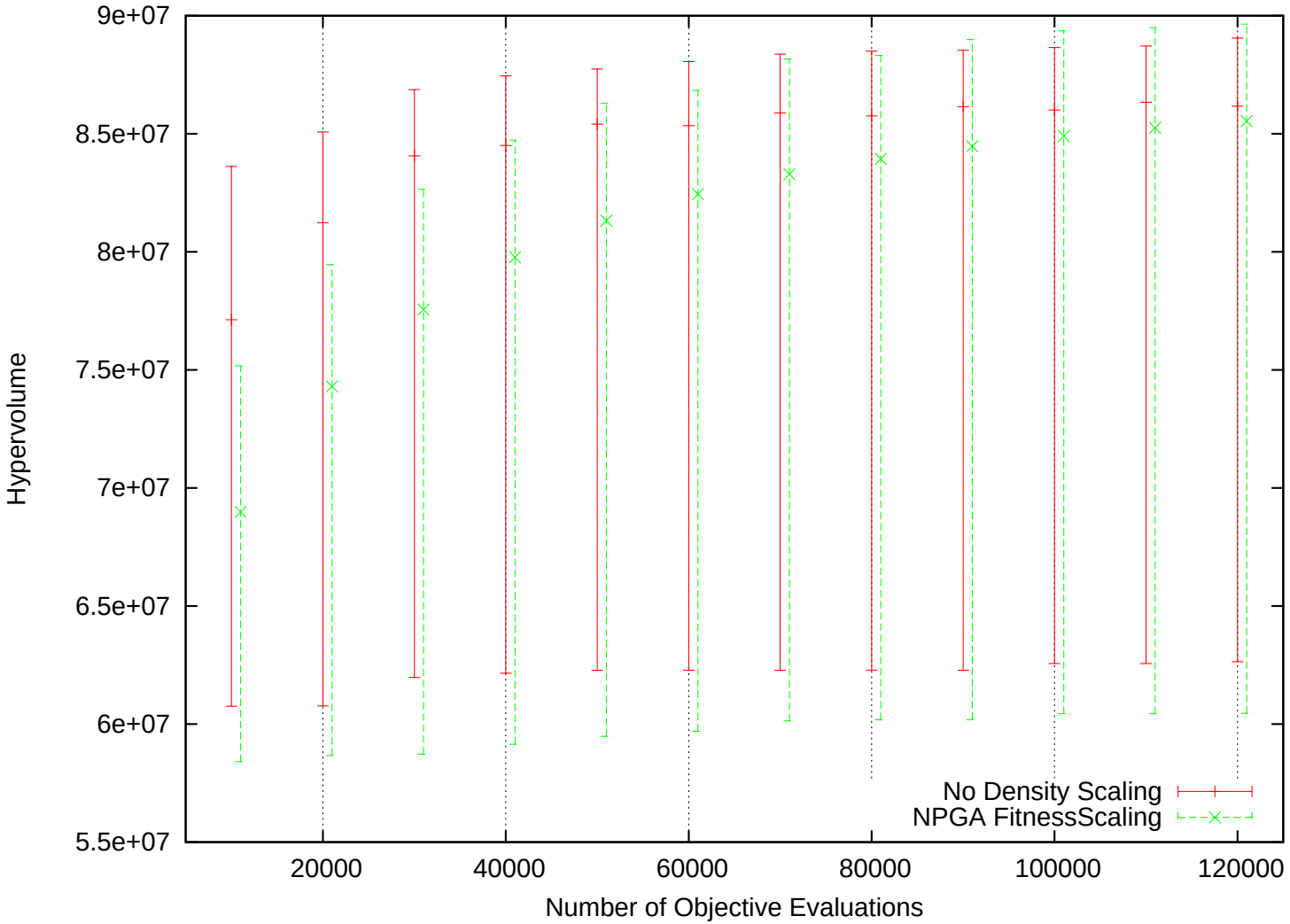

Figure 54: Comparison of Density Methods for Weighted-Sum Combinations

median value being slightly higher for the fitness proportionate selection than the binary tournament selection method, for all but the first $10,000$ objectives.

Using the same methods outlined in the previous sub-section for selection of the best combinations, Figure 56 displays the results of the best two weighted-sum combinations for the minimum performance of the first and last $30,000$ objectives. Three versions of the weighted-sum method are selected for further inspection. Two of the methods selected use contrasting selection methods. The third method selected is the only combination found that utilizes fitness sharing.

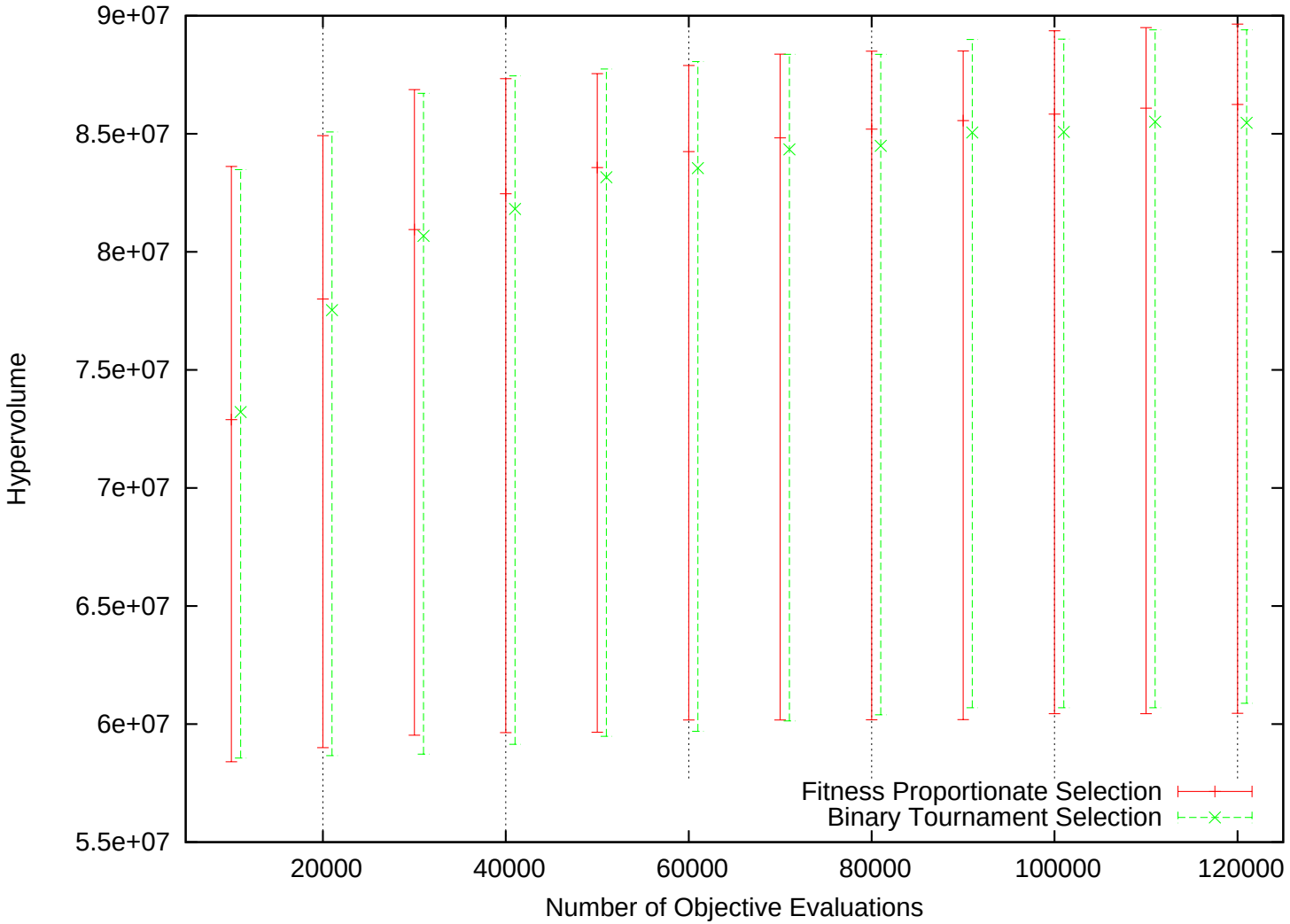

Figure 55: Comparison of Selection Methods for Weighted-Sum Combinations

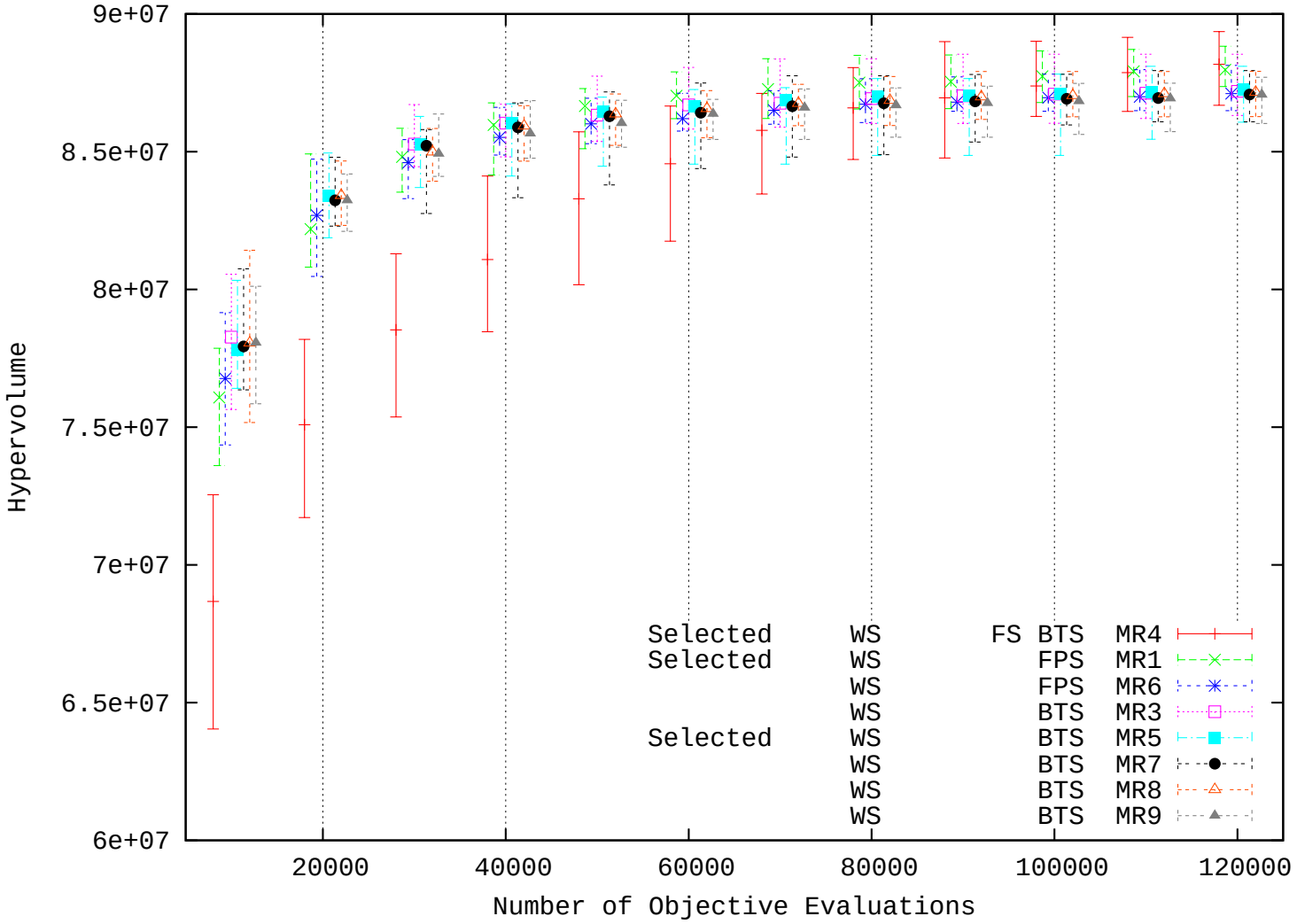

Figure 56: Hypervolume for Weighted-Sum Methods that Provide Largest Minimum Hypervolume

4.1.2.5 SPEA Combinations

For the SPEA method, two sets of variations are provided: thirteen possibilities of elitism, and the two possibilities for selection. Therefore, there are only a total of twenty-six combinations.

The thirteen elitism possibilities provide Pareto optimal elitism and maximum rank two to thirteen. Note that it is not possible to use the SPEA algorithm without elitism. Also note that selection with Pareto optimal solutions is the SPEA algorithm setup utilized by Zitzler [61]. In addition to the maximum rank methods, the SPEA cluster algorithm is also used for elitism. Figure 57, which compares the results of elitism, shows that median performance is increased beyond that of the Pareto optimal elitism when the elitism is expanded to include individuals with larger Pareto rank values. Maximum ranks of two and three show improvement for all $120,000$ objectives. A maximum rank of four shows improvement for all but the first $20,000$ objectives. Use of maximum rank above four shows no appreciable improvement. Note that the use of individuals that are not Pareto optimal is not included in Zitzler's description of the SPEA algorithm (see Section 2.2.5), but is included in his description of the SPEA II algorithm [63] (see Section 2.2.6). There are a total of two combinations for each selection method examined. Thus, there is a total of sixty runs for each error-bar.

The second comparison is for the selection methods, which contains the thirteen possibilities of elitism for a total of 390 runs for each error-bar illustrated in Figure 58. This comparison illustrates that the binary tournament selection method outperforms that of the fitness proportionate selection methods in all three statistics for all $120,000$ objectives. Note that binary tournament selection is the selection method used by Zitzler in the default algorithm and is the best choice for selection methods.

Using the same methods outlined in Section 4.1.2.3 of the best combinations, Figure 59 displays the results of the best two SPEA combinations for the minimum performance for the first and last $30,000$ objectives. From this figure note that the only variations in the

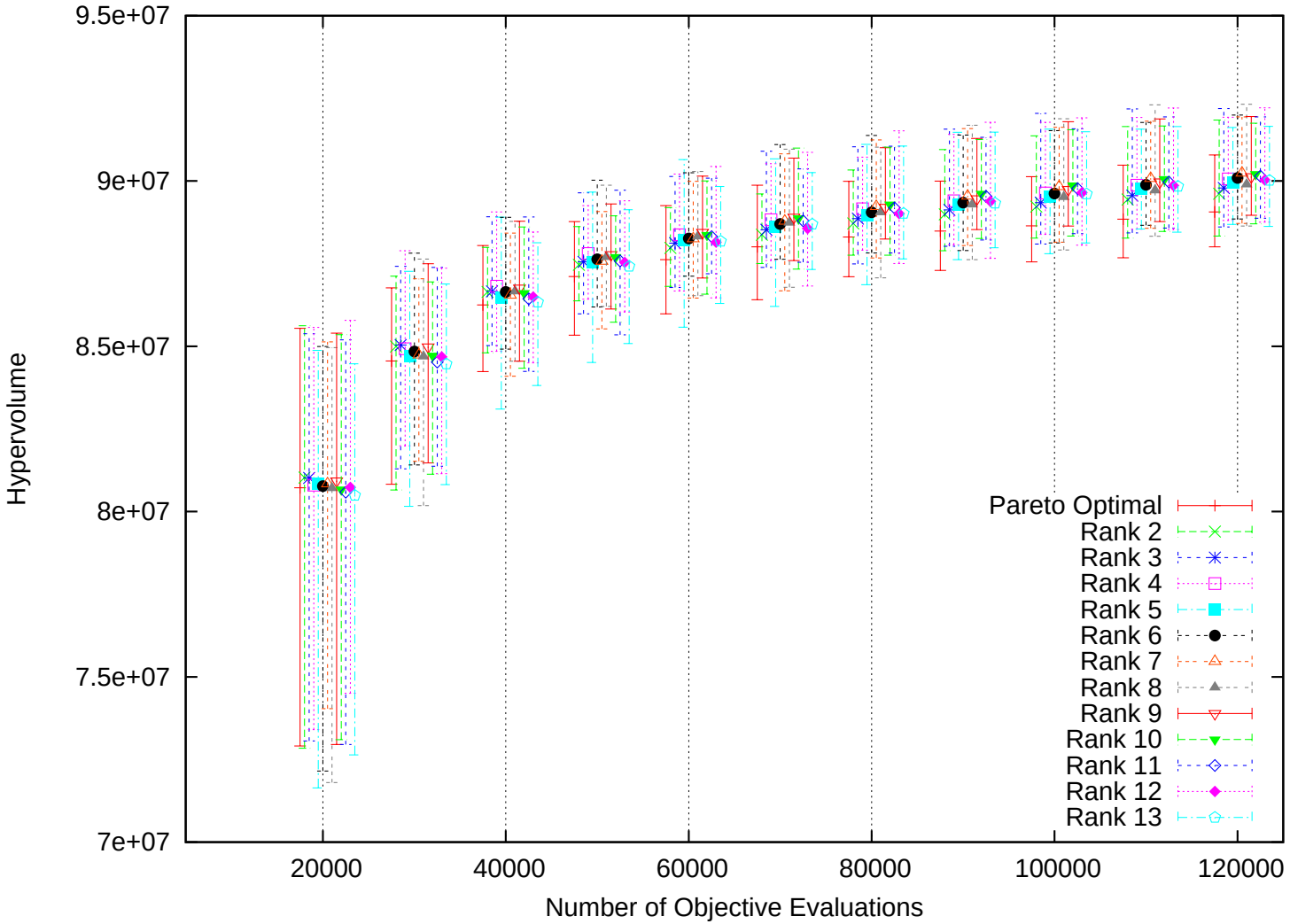

Figure 57: Comparison of Elitism Methods for SPEA Combinations

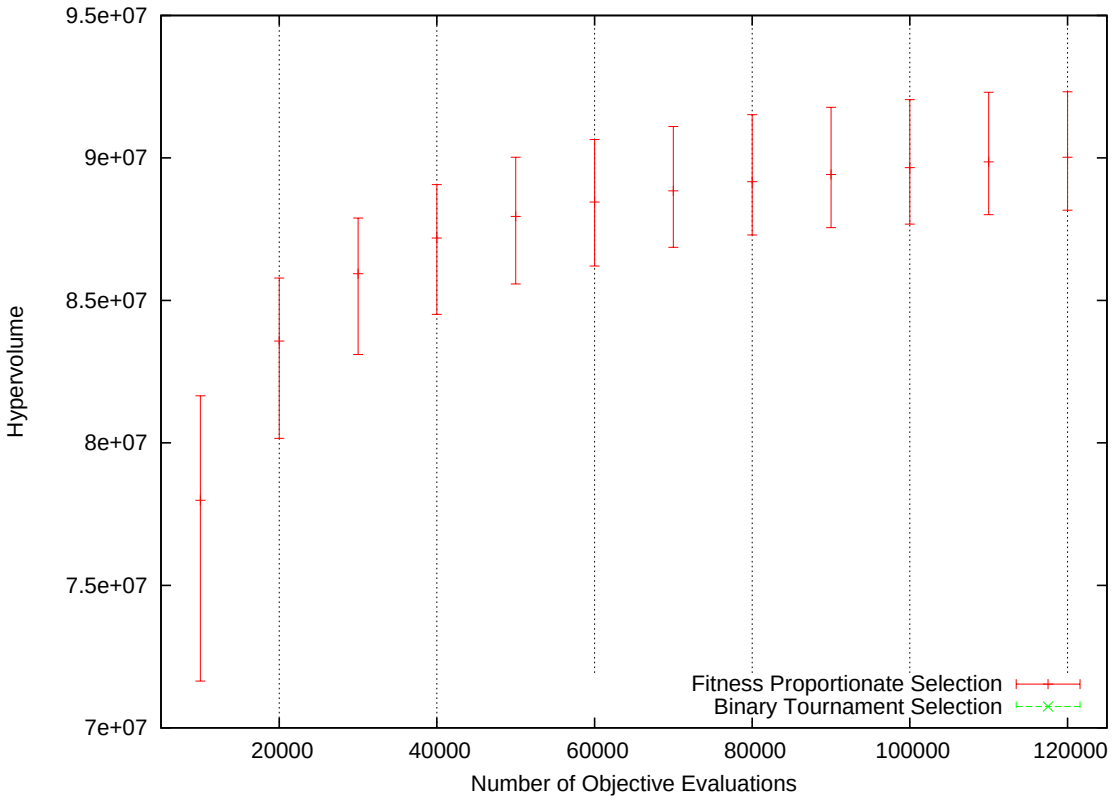

Figure 58: Comparison of Selection Methods for SPEA Combinations

maximum Pareto rank provide the variation. The maximum ranks of three and eight are selected for further investigation. In addition, Pareto optimal elitism is also kept, allowing comparison to the base SPEA algorithm.

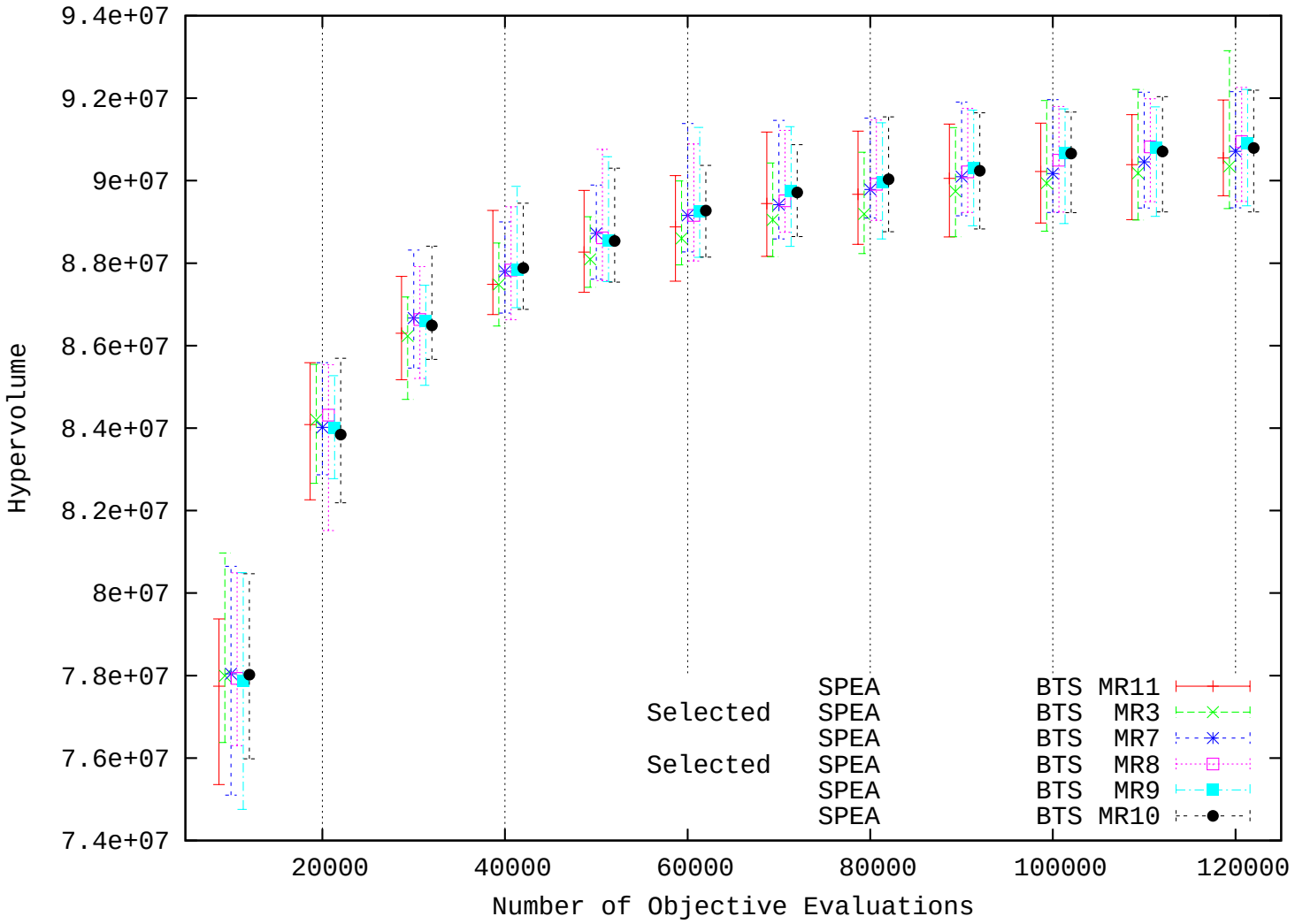

Figure 59: Hypervolume for SPEA Methods that Provide Largest Minimum Hypervolume

4.1.2.6 SPEA II Combinations

Like the SPEA analysis of the previous subsection, two sets of variations are provided for the SPEA II combinations: the thirteen possibilities of elitism, and the two possibilities for selection. Again, there are only a total of twenty-six combinations.

Figure 60 compares the results for elitism, where each error-bar presents the results for sixty runs. Interestingly, the SPEA algorithm proposed by Zitzler does not limit the population based on the Pareto optimal, but rather based on fitness values. Thus the highest rank of thirteen most closely matches that of Zitzler's algorithm. There are no clear winners when comparing minimum and median values, but the Pareto optimal only solutions do provide a noticeable difference for the maximum hypervolume found.

The second comparison is for the selection methods, which contain the thirteen possibilities of elitism for a total of 390 runs for each error-bar illustrated in Figure 61. Like the SPEA results of the previous section, the binary tournament selection method outperforms that of the fitness proportionate selection methods in all three statistics for all $120,000$ objectives. Like the SPEA method, the binary tournament selection is the selection method used by Zitzler in the default SPEA II algorithm. Also like the SPEA method, the binary tournament selection provides the best choice for the selection method.

Using the same methods outlined in Section 4.1.2.3 of the best combinations, Figure 62 displays the results of the best two SPEA combinations for the minimum performance for the first and last $30,000$ objectives. Two combinations are selected for further analysis: one with fitness proportionate selection, and one with binary tournament selection. Because the combination selected with binary tournament selection also contains a large maximum rank (eleven), it is not considered necessary to run the default SPEA II algorithm, which limits elite individuals only based on the fitness values.

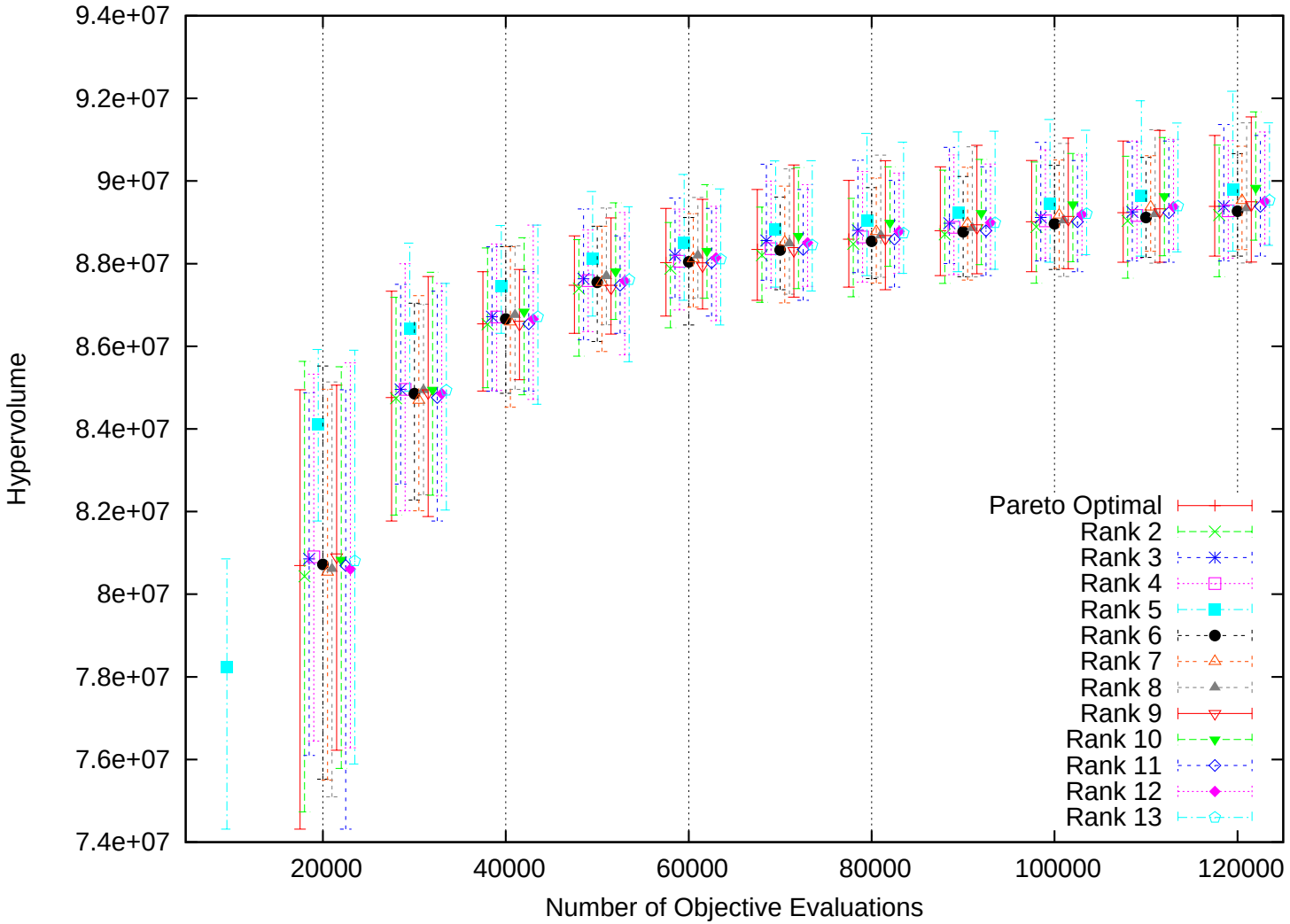

Figure 60: Comparison of Elitism Methods for SPEA II Combinations

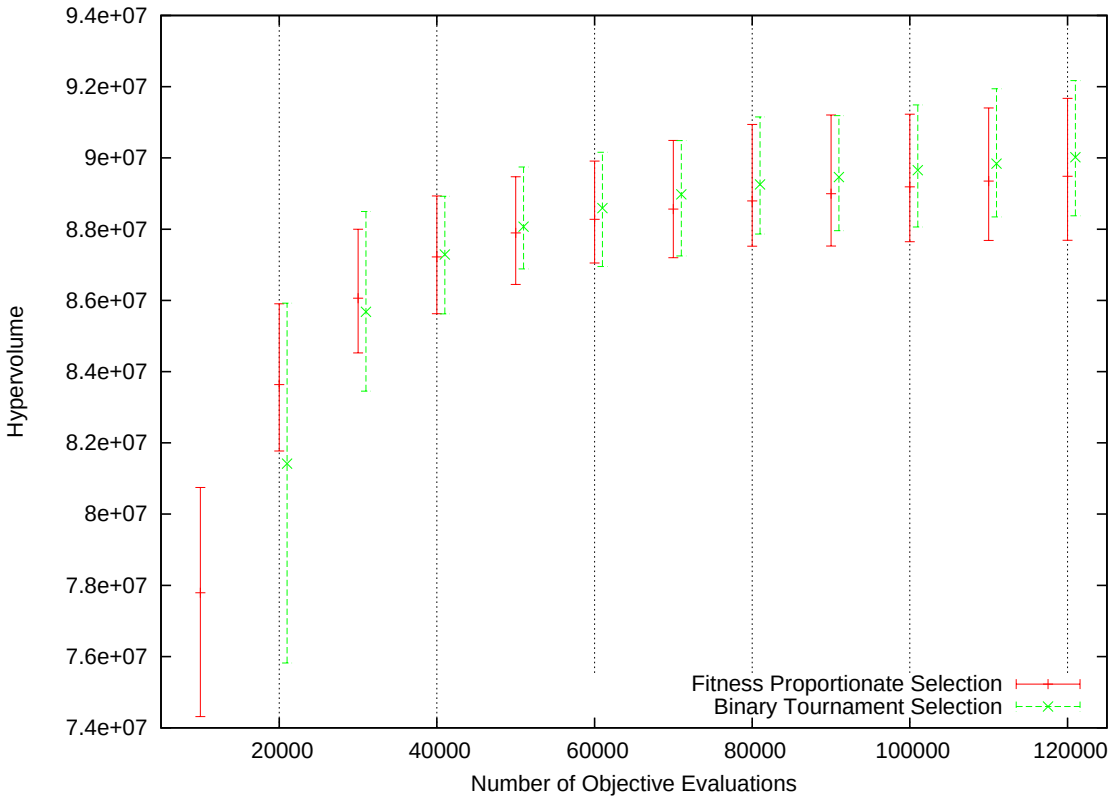

Figure 61: Comparison of Selection Methods for SPEA II Combinations

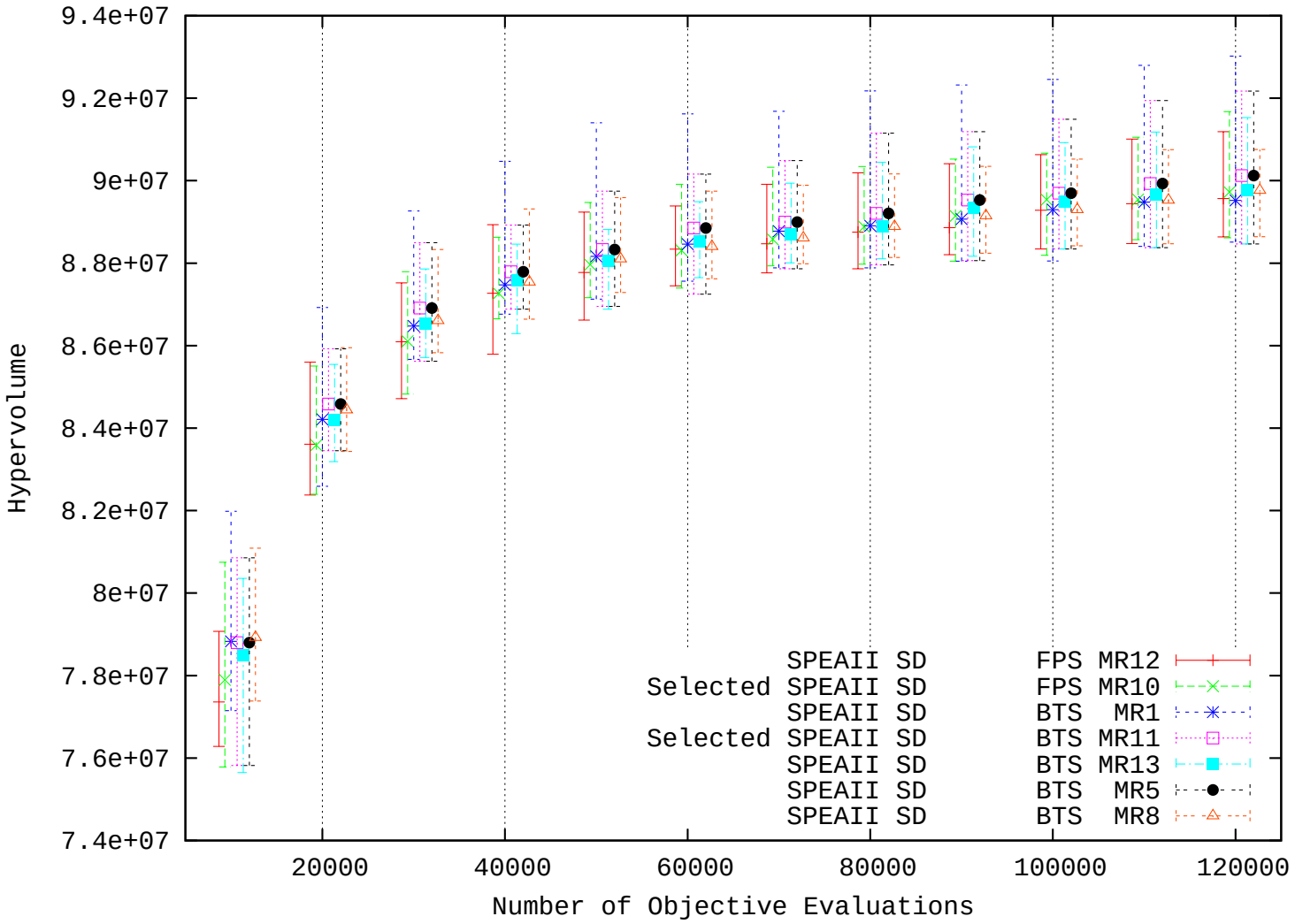

Figure 62: Hypervolume for SPEA II Methods that Provide Best Minimum Hypervolume

4.1.2.7 Summary for Combinations of MOEA Components

Table 9 presents the sixteen combinations of methods that are selected for further analysis with the ICEO methods of the next sub-section. These methods are selected based on the results of the methods that provide the best minimum hypervolume as presented in the previous sub-sections as well as selection of methods that provide a diversity of algorithm combinations. Figure 63 displays the hypervolumes for those combinations selected. Note that the weighted-sum methods perform poorer than any of the other methods.

Fitness	SPEA II Density Scaler	Transform Scaler	Fitness Sharing	Selection	Elitism
PR1	SD	T1	FS	FPS	MR13
PR1	SD	T1	FS	BTS	MR11
PR1	SD	T2	FS	FPS	MR10
PR1		T2	FS	BTS	MR6
PR2	SD	T1	FS	FPS	MR2
PR2	SD	T1	FS	BTS	MR8
PR2	SD	T2	FS	FPS	MR12
PR2		T2	FS	BTS	MR13
WS			FS	BTS	MR4
WS				FPS	MR1
WS				BTS	MR5
SPEA				BTS	MR1
SPEA				BTS	MR3
SPEA				BTS	MR8
SPEA II	SD			FPS	MR10
SPEA II	SD			BTS	MR11

Table 9: Table of Combinations Selected for Further Analysis

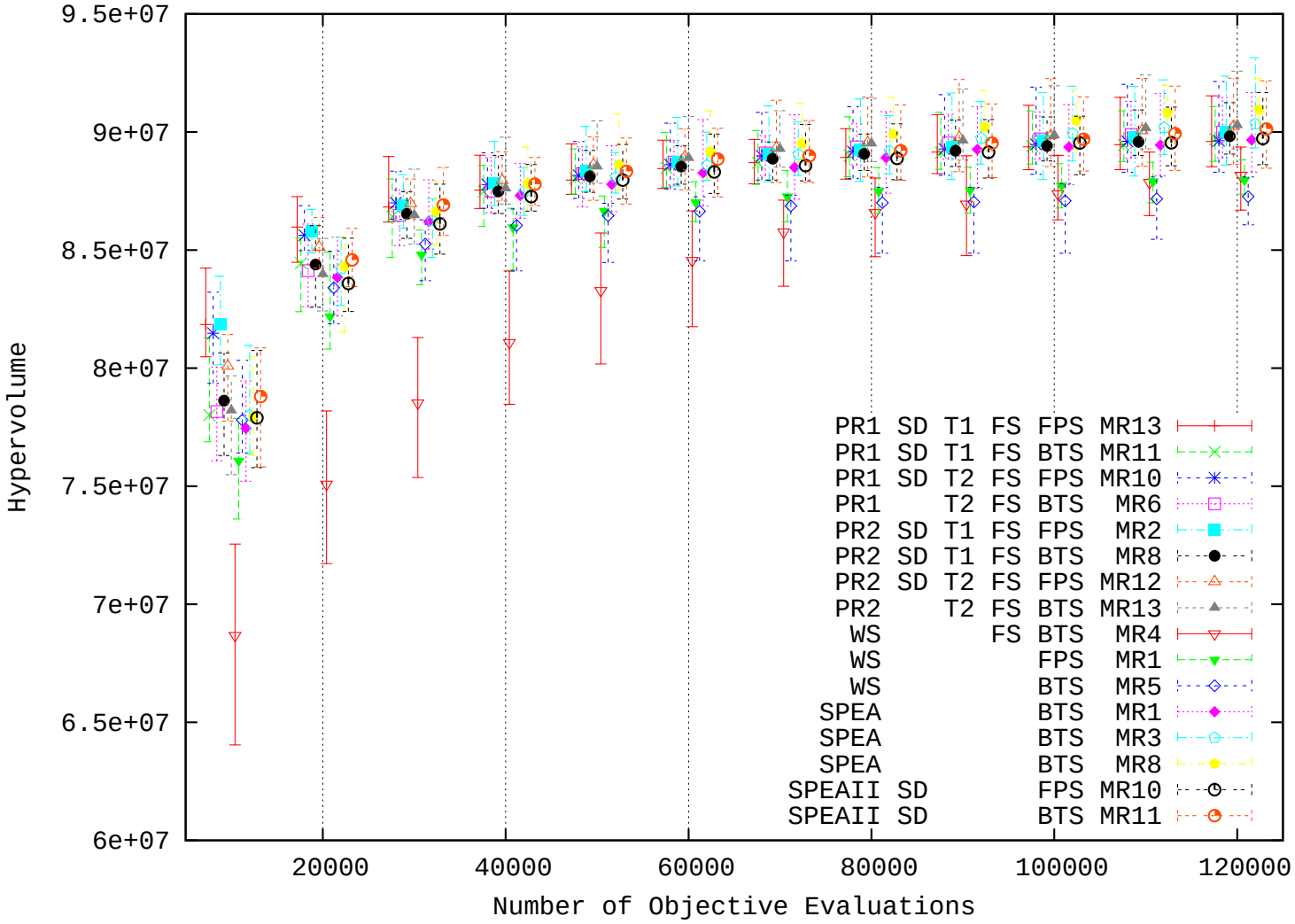

Figure 63: Comparison of Combinations Selected For Further Analysis

4.1.3 Analysis of ICEO Methods for 0/1 Knapsack Problem

The following sub-sections detail the performance of the 0/1 Knapsack problem using ICEO techniques. The sections investigate the use of dynamic objective thresholding, hypercube-distance scaler, and the hypercube distance dynamic objective ordering method against three 0/1 Knapsack problems of increasing complexity.

4.1.3.1 Dynamic Thresholding With Two Objectives

To test the new dynamic objective thresholding algorithm, two reachable regions of interest are selected from the 250 items and two knapsacks problems of the previous section. As indicated in Figure 64, the first region of interest is defined by $\boldsymbol{L_1} = \{8500, 9200\}$, and the second region of interest is defined by $\boldsymbol{L_2} = \{9200, 8500\}$. Although the regions of interest are symmetrical due to the selection of $\boldsymbol{L_1}$ and $\boldsymbol{L_2}$, the results are not symmetrical due to the profits, weights and capacities, which are all dependent on the knapsack. For this investigation the lower bounds are set to zero, $\boldsymbol{S} = \{0, 0\}$. For the cases without dynamic objective thresholding, thresholding is disabled by setting the weights to zero, $\boldsymbol{W} = \{0, 0\}$. For the cases with dynamic objective thresholding, the weights are $\boldsymbol{W} = \{0.99, 0.99\}$. The size of the historic pool for dynamic objective thresholding is set to the same size as the number current generation, $|H_T| = |C| = 120$. The number of individuals between updates is also set to 120 individuals, $D_T = 120$.

The goal of ICEO problems is to reach the region of interest as fast as possible. Because the 0/1 Knapsack problem is not computationally expensive nor are the objectives independent, instead of using time, the measure of performance is the number of objectives required to reach the region of interest. To measure the effectiveness of each algorithm, thirty runs are made where the run is truncated as soon as the region of interest is reached. A maximum of $120,000$ objectives is allowed to be evaluated. Horizontal error-bars are used to display the minimum, maximum and median number of objectives required. Figure 65 presents the

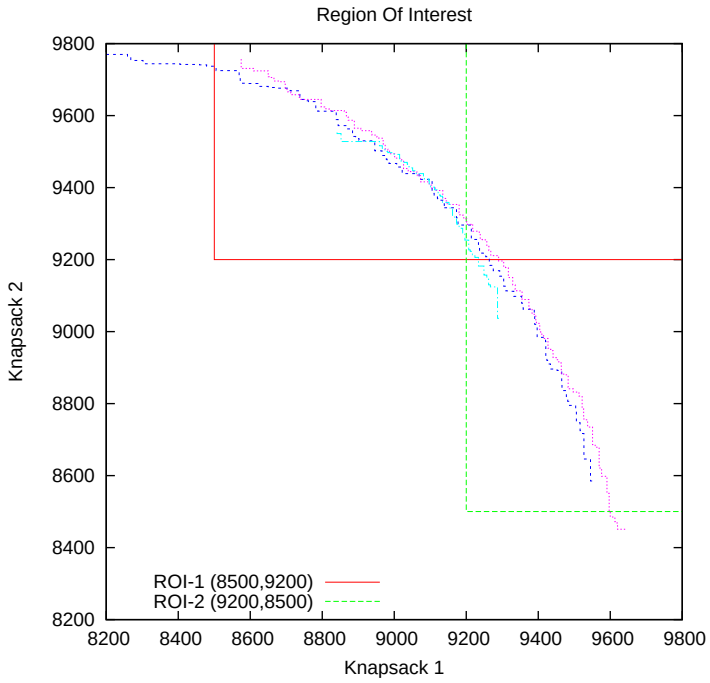

Figure 64: Regions of Interest For Two Objective Problems

results of reaching the two regions of interest with and without dynamic thresholding enabled. The performance of reaching the first region of interest, $\boldsymbol{L} = \{8500, 9200\}$, is worse with dynamic objective thresholding than without it. The performance of reaching the second region of interest, $\boldsymbol{L} = \{9200, 8500\}$, is better with dynamic objective thresholding than without it for all combinations except those including the SPEA algorithm.

The lack of improvement for the SPEA algorithm is likely due to the inability of the SPEA algorithm to function properly when reduced to a single objective. The raw fitness value of the Pareto optimal individual, i.e., the individual with the best performance when reduced to single objective, is given a fitness value as:

$$S(i) = \frac{|\{j | j \in C \wedge f(i) \geq f(j)\}|}{|C| + 1} = \frac{|C|}{|C| + 1} \approx 1. \tag{117}$$

With only a single objective, there is only on Pareto optimal individual, $|E_t| = 1$. Thus, the fitness of non-Pareto individuals is:

$$F(j) = 1 + \sum_{i \in E_t, f(i) \geq f(j)} S(i) = 1 + \frac{|C|}{|C| + 1} \approx 2. \tag{118}$$

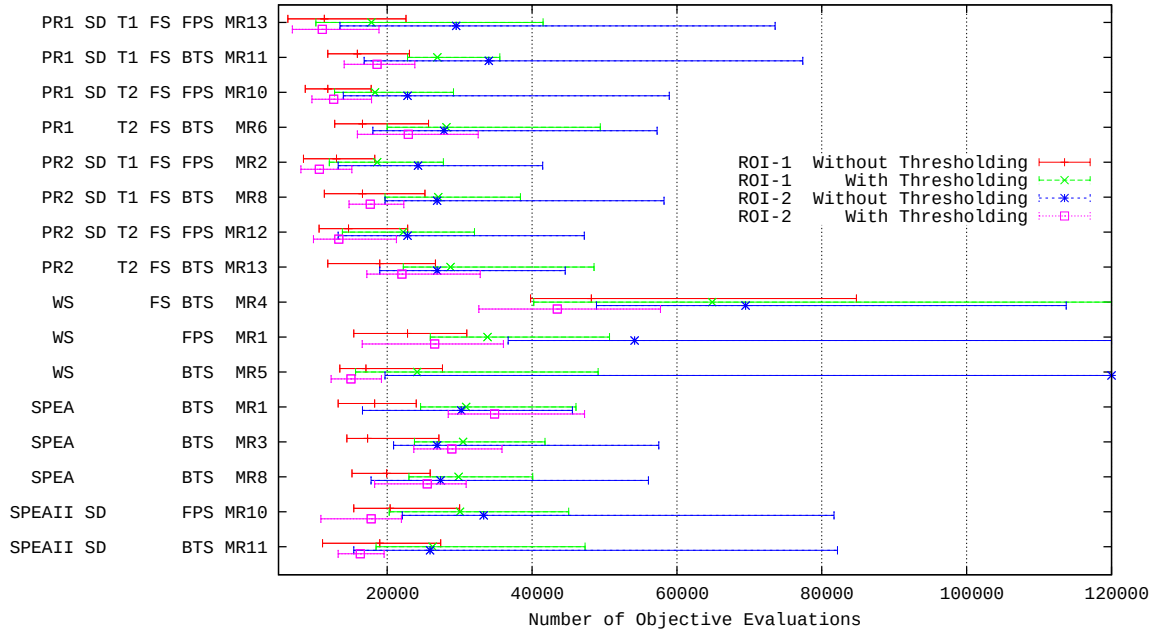

Figure 65: Performance in Reaching Regions of Interest With and Without Dynamic Objective Thresholding for Selected Algorithm Combinations Using 2 Knapsacks and 250 Items

Therefore, there is no difference in the probability of mating for non-Pareto optimal individuals when the problem is reduced to a single objective dimension.

Figure 65 shows that dynamic objective thresholding performance can be detrimental when the first objective is not very difficult to reach. Also note that without dynamic objective thresholding, the first region of interest requires fewer objectives to be evaluated than does the second. Using thresholding, the decrease in the number of objectives required to reach the second objective, and also a decrease in the deviation of results, indicates that dynamic objective thresholding can be very effective if the proper objective is selected first. Thus, there is a need to exploit the order of objectives, such as the hypercube distance dynamic objective ordering, which will be analyzed in Section 4.1.3.3.

4.1.3.2 Hypercube Distance Scaler With Two Objectives

The hypercube distance scaler requires the definition of a region of interest in objective space, but does not require dynamic objective thresholding to be utilized, (see Section 3.3 for details of the HCD scaling algorithm). The HCD scaler can be applied anywhere in the chain of fitness scalers that are applied after the raw fitness value is calculated. For this analysis, the HCD scaler is applied as the last operator in the chain of fitness scalers and contains an offset of zero, $\alpha = 0.0$. Figure 66 illustrates the improved performance with HCD scaling. *Note that when dynamic objective thresholding is not applied, all algorithm combinations dramatically improve their performance with the use of HCD scaling. This implies HCD scaling has broad applicability to MOEA techniques.* When dynamic objective thresholding is applied to the second region of interest, there are several cases, e.g., the PR1 with T1 cases, where the minimum performance is increased with HCD scaling. But, even with these cases, the overall deviation in performance of runs is reduced. All of the remaining selected algorithm combinations show improvement in performance especially with the first region of interest. *Therefore, the remaining analysis uses the HCD scaling algorithm.*

4.1.3.3 Dynamic Objective Ordering With Two Objectives

As illustrated in Figure 65, dynamic thresholding is most effective when the objectives are in a desired order. To investigate this relationship, this and the following sections present results where the combinations of algorithms are evaluated in four ways:

1. Without any dynamic objective thresholding, i.e., all objectives are evaluated for every individual.

2. Dynamic objective thresholding is applied in the order the objectives are specified, i.e., objective 1 evaluated first, then objective 2 evaluated second.

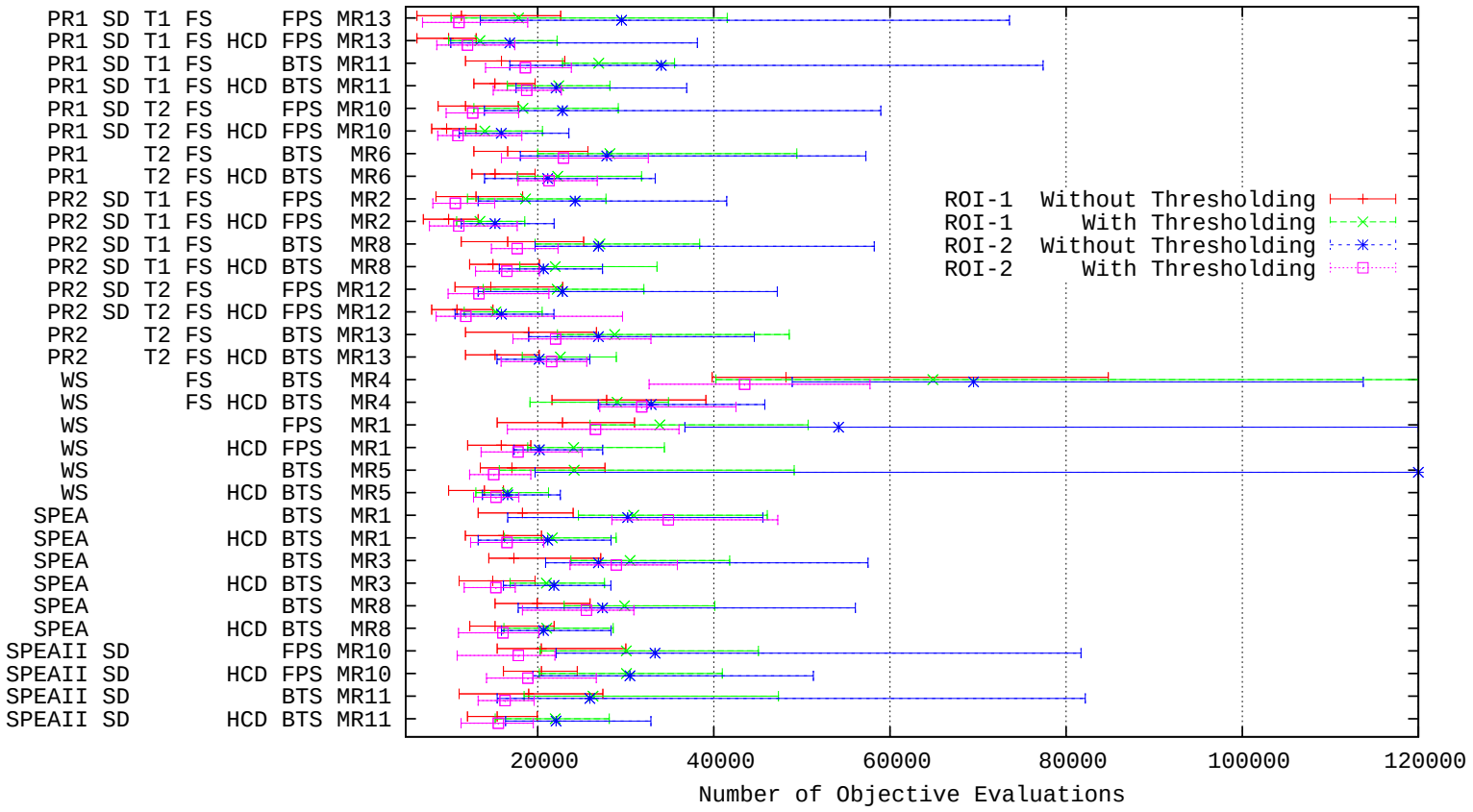

Figure 66: Performance Enhancement of Hypercube Distance Scaler in Reaching Regions of Interest With and Without Dynamic Objective Thresholding for Selected Algorithm Combinations Using 2 Knapsacks and 250 Items

3. Dynamic objective thresholding is applied in the reverse of the order in which the objectives are specified, i.e., objective 2 evaluated first, then objective 1 evaluated second.

4. Dynamic objective thresholding is applied in the order specified by hypercube distance dynamic objective ordering.

Again, thirty runs are evaluated for each possibility. For a full description of hypercube distance dynamic objective ordering see Section 3.4.2. The initial order of objectives when utilizing hypercube distance dynamic objective ordering is set randomly to remove any bias from the initial objective ordering. The number of individuals in the historic pool $|H_O|$ is set to 500. The number of individuals evaluated between ordering updates D_O is set to 100.

Figure 67 illustrates the performance in reaching region of interest 1 for the four methods of dynamic objective ordering and thresholding applied to each of the selected combinations. Figure 68 illustrates the performance in reaching region of interest 2. Remembering that the desire is to optimize results for ICEO problems where the ability to make multiple runs is not likely, the most important statistic for comparison is the maximum number of objectives evaluated. Thus, method 1 outperforms method 2 if the maximum number of objectives required for method 1 is less than the maximum number of objectives required for method 2.

Note that for the simpler case, i.e., region of interest 1, running without any dynamic objective thresholding or ordering still outperforms all other dynamic objective thresholding or ordering cases. It is only for the more difficult case, i.e., region of interest 2, that the dynamic objective thresholding and ordering begins to outperform the runs without any dynamic objective thresholding or ordering.

For twenty-five out of the thirty-two total combinations presented, the runs with HCD dynamic objective ordering perform better than the worst possible objective ordering. This implies that for a majority of the combinations, dynamic objective ordering is able to improve the performance beyond that of the worst possible ordering. It is also important

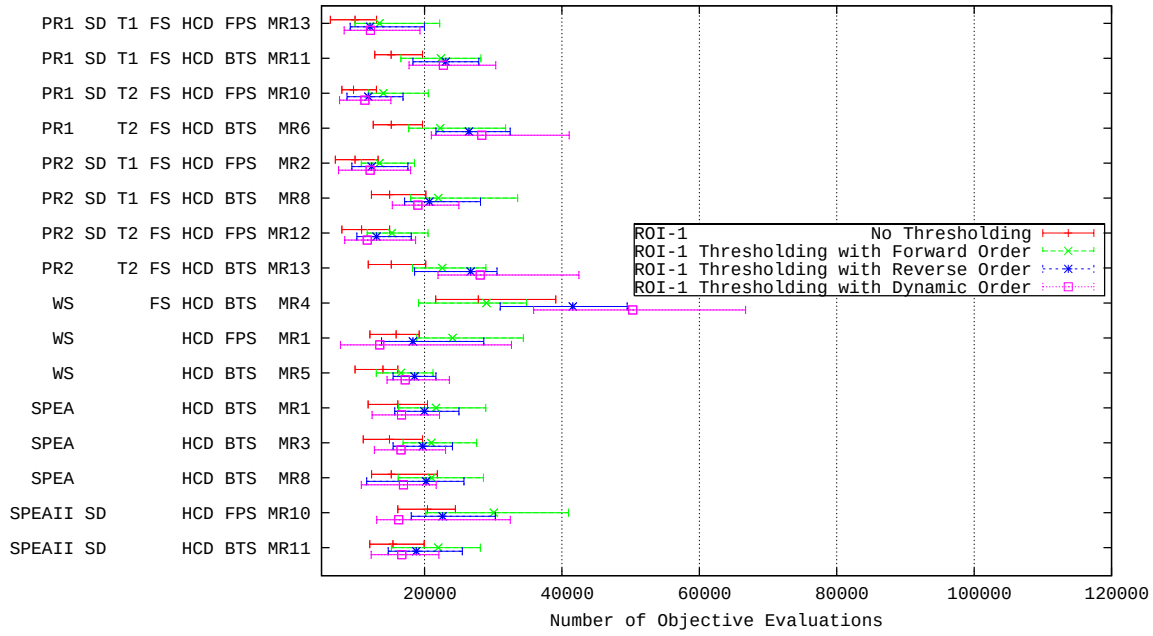

Figure 67: Performance Enhancement of Ordering Methods in Reaching Regions of Interest 1 for Selected Algorithm Combinations Using 2 Knapsacks and 250 Items

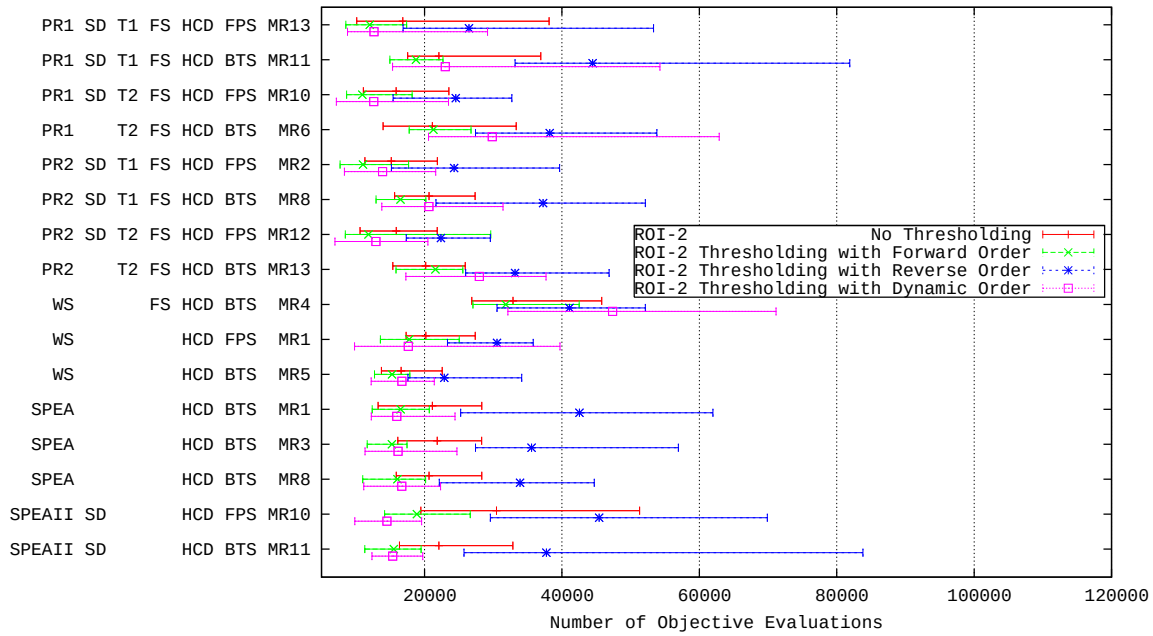

Figure 68: Performance Enhancement of Ordering Methods in Reaching Regions of Interest 2 for Selected Algorithm Combinations Using 2 Knapsacks and 250 Items

to note that six out of seven of the cases, where the HCD dynamic objective ordering performed worse than any other method, are cases where binary tournament selection is utilized.

The poor performance of dynamic objective ordering and binary tournament selection is likely due to an exacerbation of the problem of the combinations of dynamic objective thresholding and binary tournament selection in general. With dynamic objective thresholding a number of individuals are assigned a fitness based on a single objective. For Pareto rank with the inversion transformations and the weighted-sum fitness measures, the single objective value results in a dramatic decrease in the fitness values. When using fitness proportionate selection, this dramatic decrease in fitness results in a corresponding decrease in the probability of selection for the poorly performing individuals and increases the probability of selection for the better performing individuals based on the following equation for the probability of selection of an individual as:

$$P_{s,FPS}(i) = \frac{F(i) - F_{Min}}{\sum_{j \in P}(F(j) - F_{Min})}, \tag{119}$$

where $F_{Min} = \min_{i \in P} F(i)$. With the binary tournament selection method, these dramatic changes in fitness values may only change the probability of selection by reordering the fitness values. The probability of the k^{th} individual in the sorted list of fitness values is still:

$$P_{s,BTSNR}(k, 1) = \frac{2(k-1)}{n(n-1)}. \tag{120}$$

Another note is that for some cases, (specifically PR2-SD-T2-FS-HCD-FPS-MR12 and SPEAII-SD-HCD-FPS-MR10), the performance with HCD dynamic objective ordering outperforms either static objective ordering. This implies that the optimal objective ordering is not static throughout the run and the HCD dynamic objective ordering is able to modify the objective ordering through the evolutionary epoch to produce results that are better than any other possible static ordering.

4.1.3.4 Dynamic Objective Ordering With Four Objectives

The next experiment with the selected combinations is with an increased search space size of 750 items and an increased objectives space of four objectives. Three regions of interest are selected. The first region of interest, $\boldsymbol{L_1} = \{24000, 21000, 21000, 21000\}$, is expected to work well when objectives are solved starting with the first objective. The second region of interest, $\boldsymbol{L_2} = \{21000, 21000, 21000, 24000\}$, is expected to work well when objectives are solved starting with the fourth objective solved first. The third region of interest, $\boldsymbol{L_3} = \{21000, 24000, 24000, 21000\}$, is expected to remove the advantage of solving the objectives in forward or reverse order. For these investigations, the weights for the dynamic objective thresholds are set to ninety-five percent, $\boldsymbol{W} = \{0.95, 0.95, 0.95, 0.95\}$.

Because of the increased difficulty of this problem, the maximum number of objectives allowed is increased to 300, 000 objective evaluations, the number of individuals per generation is increased to 280 individuals, $|C| = 280$, the maximum number of individuals in the elite population is increased to seventy, $|E| = 70$, and the number of individuals in the initial random population is increased to 350. The settings for the dynamic objective thresholding, dynamic objective ordering, crossover and mutation are identical to that of the two knapsack 750 item investigation described in Sections 4.1.2.2 and 4.1.3.3.

Figure 69 displays the results when evaluated for region of interest 1 defined by $\boldsymbol{L_1} = \{24000, 21000, 21000, 21000\}$, where the first objective is the most difficult. Note that dynamic objective thresholding, whether evaluated in the forward or reverse order, improves the performance beyond that achieved without dynamic objective thresholding. Dynamic objective ordering is able to achieve performance similar to that with the anticipated optimal ordering. A slight reduction in performance from the optimal order is expected because with the randomly initialized order, the process must "learn" the optimal order from the evaluation of previous individuals.

Figure 70 displays the results when evaluated for region of interest 2 defined by $\boldsymbol{L_2} =$

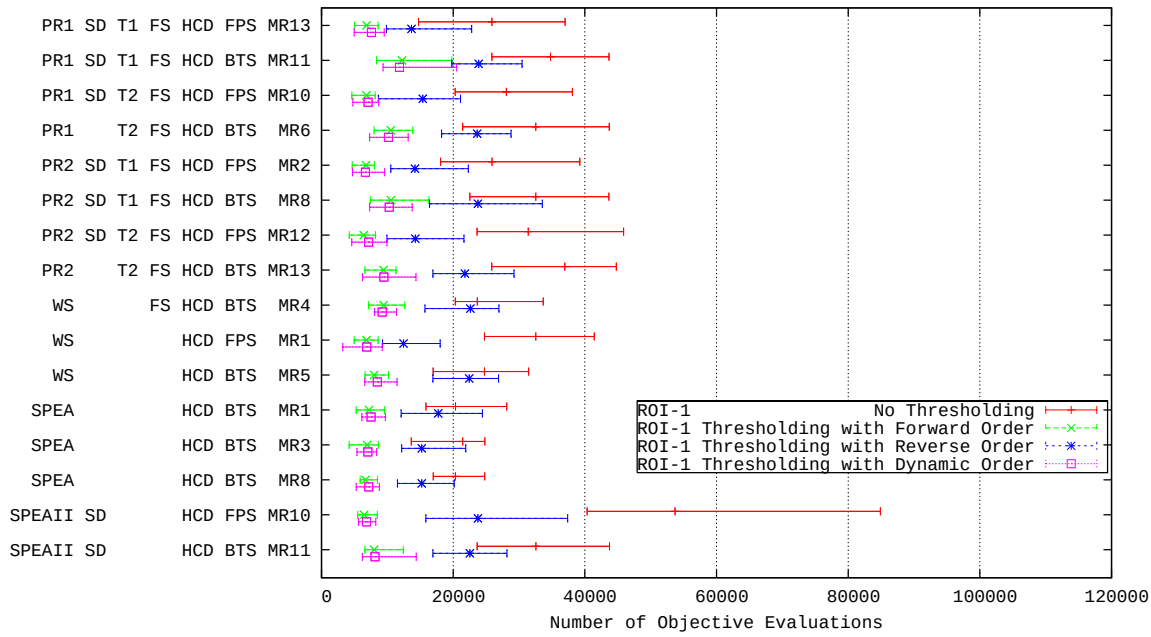

Figure 69: Performance Enhancement of Ordering Methods in Reaching Regions of Interest 1 for Selected Algorithm Combinations Using 4 Knapsacks and 750 Items

{21000, 21000, 21000, 24000}, where the fourth objective is the most difficult. Unlike region of interest 1, dynamic objective thresholding does not always outperform the worst possible forward ordering case. But, like region of interest 1, dynamic objective ordering is able find orders that are only slightly worse than the anticipated optimal reverse objective ordering.

Figure 71 displays the results when evaluated for region of interest 3 defined by $\boldsymbol{L_3} = \{21000, 24000, 24000, 21000\}$, where neither forward nor reverse objective ordering is expected to be optimal. Despite these anticipated results, the results without dynamic objective thresholding are able to outperform the results with dynamic objective thresholding in seven out of the sixteen results, where outperform implies the smallest maximum number of objective evaluations required. But, these seven results are all associated with the binary tournament selection method, and thus the arguments of Section 4.1.3.3 discussing the problems of the combination of binary tournament selection and dynamic objective

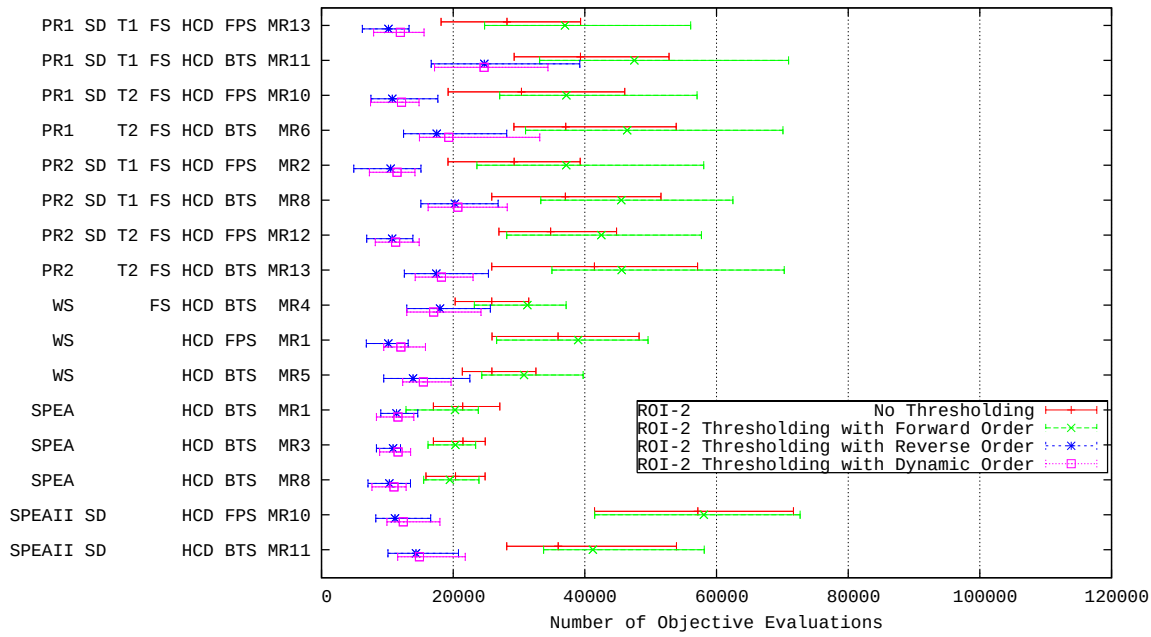

Figure 70: Performance Enhancement of Ordering Methods in Reaching Regions of Interest 2 for Selected Algorithm Combinations Using 4 Knapsacks and 750 Items

thresholding are applicable. Dynamic objective ordering again illustrates good results at finding an order that performs with close to the best performance. But, it must be noted that only two of the sixteen possible objective orderings are investigated.

An unanticipated result of this investigation is the good performance of the SPEA algorithm combinations without dynamic objective thresholding or dynamic objective ordering for all three regions of interest. These results are not duplicated for the two objective or ten objective knapsack problems. This may show a superiority of the SPEA algorithm combinations for problems where relatively few objective evaluations are required.

4.1.3.5 Dynamic Objective Ordering With Ten Objectives

The final experiment with the 0/1 Knapsack problem increases the search space size to 1000 items, and increased objectives space of ten objectives. A total of five regions of interest are selected for investigation. The first region of interest,

$$\boldsymbol{L_1} = \{29000, 29000, 27000, 27000, 27000, 27000, 27000, 27000, 27000, 27000\},$$

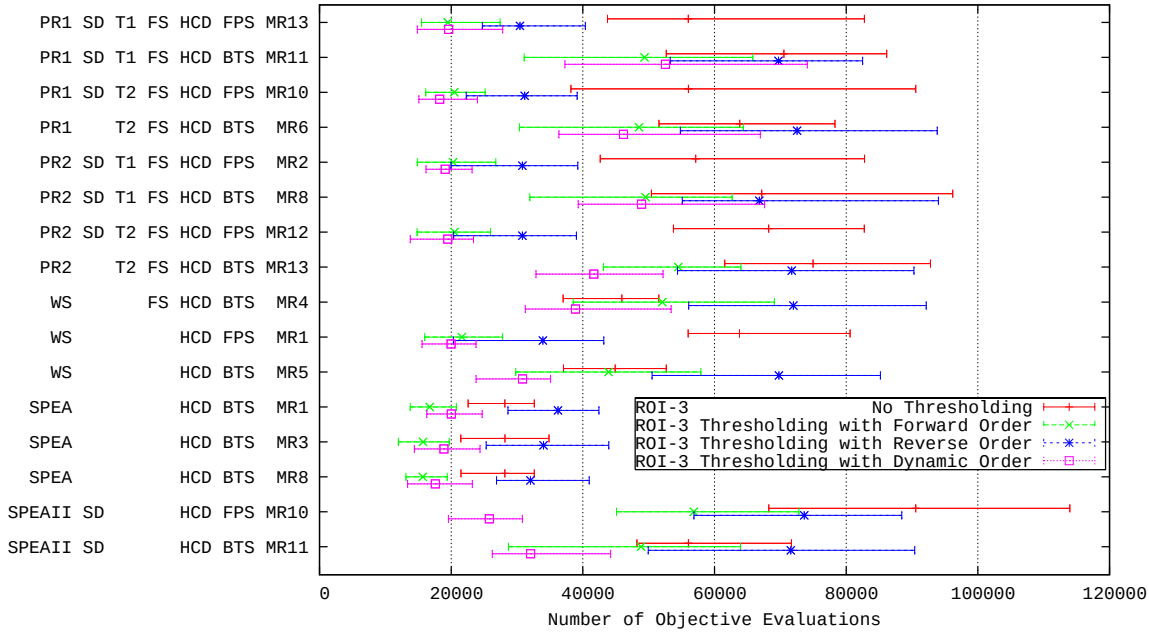

Figure 71: Performance Enhancement of Ordering Methods in Reaching Regions of Interest 3 for Selected Algorithm Combinations Using 4 Knapsacks and 750 Items

is expected to work well with dynamic objective ordering when objectives are solved in the order specified. The second region of interest,

$$\boldsymbol{L_2} = \{27000, 27000, 27000, 27000, 27000, 27000, 27000, 27000, 29000, 29000\},$$

is expected to work well with dynamic objective ordering when objectives are solved in a reverse order. The third region of interest,

$$\boldsymbol{L_3} = \{27000, 29000, 27000, 29000, 27000, 29000, 27000, 29000, 27000, 29000\},$$

is expected not to be favored by either forward or reverse objective ordering. The fourth and fifth regions of interest,

$$\boldsymbol{L_4} = \{25000, 25000, 25000, 25000, 30000, 30000, 25000, 25000, 25000, 25000\}$$

$$\boldsymbol{L_5} = \{25000, 25000, 25000, 25000, 32000, 32000, 25000, 25000, 25000, 25000\}$$

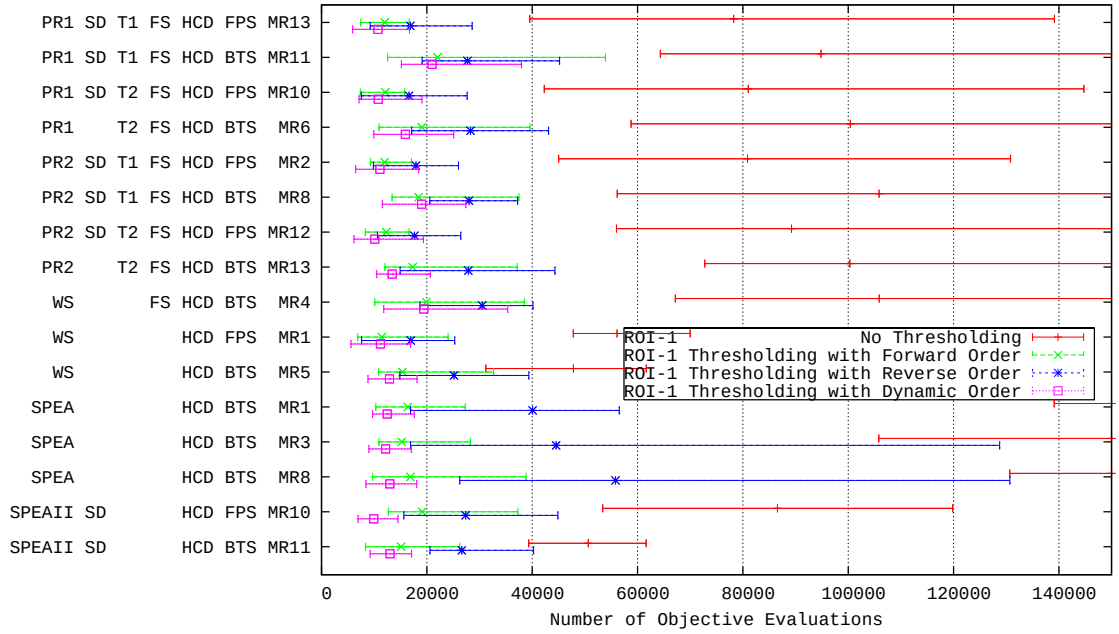

Figure 72: Performance Enhancement of Ordering Methods in Reaching Regions of Interest 1 for Selected Algorithm Combinations Using 10 Knapsacks and 1000 Items

specify regions that are successively harder to reach with either objective ordering. The weights for the all dynamic objective thresholding runs are:

$$\boldsymbol{W} = \{0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95\}.$$

The settings for the dynamic objective thresholding, dynamic objective ordering, crossover and mutation are identical to that of the four knapsack 750 item investigation described in Sections 4.1.2.2 and 4.1.3.4.

Figure 72 displays the results for the first region of interest, $\boldsymbol{L_1}$. Note that dynamic objective thresholding consistently reaches the region of interest with fewer objectives than without objective thresholding, regardless of the order of the objectives or the algorithm combination. Runs with dynamic objective ordering outperform even the static objective ordering in the forward direction for thirteen out of sixteen algorithm combinations.

Figure 73 displays the results for the second region of interest, $\boldsymbol{L_2}$. With this region of interest, the dynamic objective ordering displays superiority to both objectives orderings

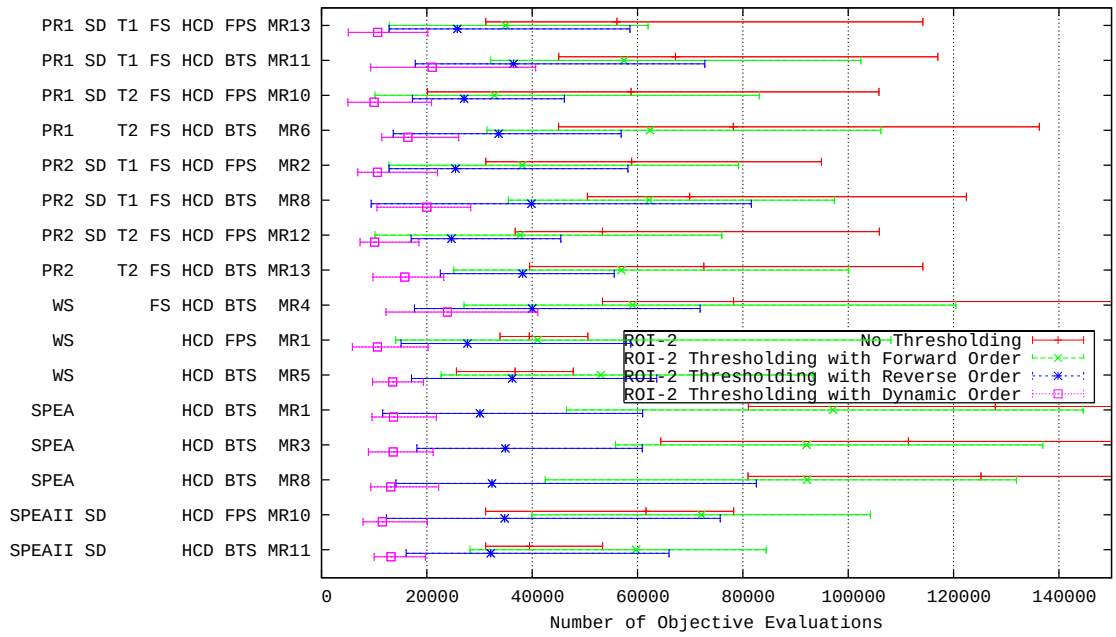

Figure 73: Performance Enhancement of Ordering Methods in Reaching Regions of Interest 2 for Selected Algorithm Combinations Using 10 Knapsacks and 1000 Items

for all algorithm combinations, including the reverse order which was anticipated to be a good performer. The only surprise with this run is the rather good performance of the WS-HCD-FPS-MR1, WS-HCD-BTS-MR5, and SPEAII-SD-HCD-BTS-MR11 combinations without dynamic objective thresholding. Similar results for these combinations are also found for region of interest 1, implying that these regions of interest are for some reason easily found by these algorithm combinations.

Figure 74 displays the results for the third region of interest, $\boldsymbol{L_3}$. With this difficult region of interest, many runs without dynamic objective thresholding and several runs with dynamic objective thresholding, but without dynamic objective ordering, are unable to reach the region of interest within $150,000$ objectives. As anticipated, results show the clear superiority of the dynamic objective ordering over the other orders.

Figure 75 displays the results for the fourth region of interest, $\boldsymbol{L_4}$. This chart clearly displays several results that have been hinted at in previous charts. First, there is a clear

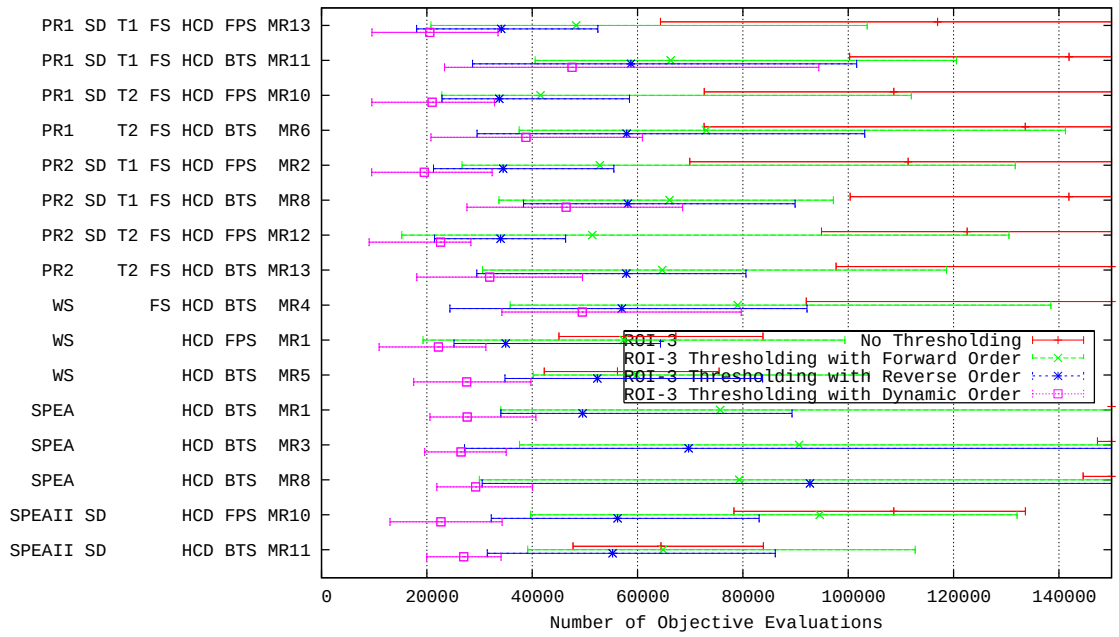

Figure 74: Performance Enhancement of Ordering Methods in Reaching Regions of Interest 3 for Selected Algorithm Combinations Using 10 Knapsacks and 1000 Items

advantage of dynamic objective thresholding with fitness proportionate selection for all Pareto rank and weighted-sum combinations. In fact, without this combination of fitness proportionate selection and dynamic objective thresholding, none of the combinations can reach the region of interest within $150,000$ objectives without dynamic objective ordering. Second, there is a clear advantage of dynamic objective ordering for all algorithm combinations. Third, five algorithm combinations, which have hinted at superiority in the past, are becoming clear winners with these difficult problems: PR1-SD-T1-FS-HCD-FPS-MR13, PR1-SD-T2-FS-HCD-FPS-MR10, PR2-SD-T1-FS-HCD-FPS-MR2, PR2-SD-T2-FS-HCD-FPS-MR12, and WS-HCD-FPS-MR1.

Figure 76 displays the results for the fifth region of interest, $\boldsymbol{L_5}$, which is the hardest region of interest examined. In this region of interest, the five algorithmic combinations itemized in the previous paragraph again display their superiority.

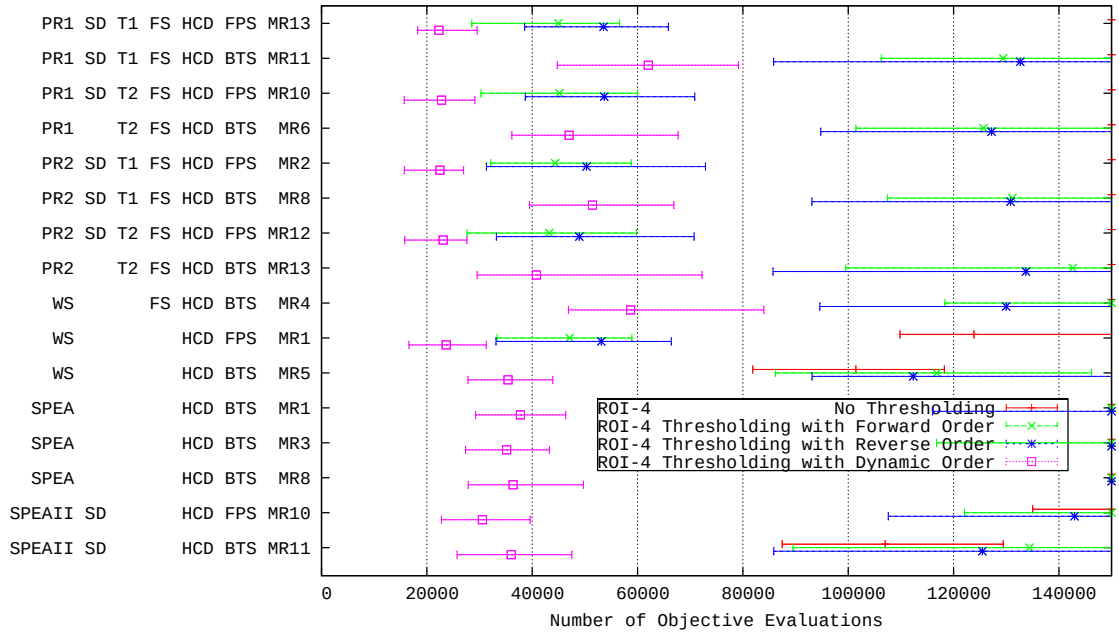

Figure 75: Performance Enhancement of Ordering Methods in Reaching Regions of Interest 4 for Selected Algorithm Combinations Using 10 Knapsacks and 1000 Items

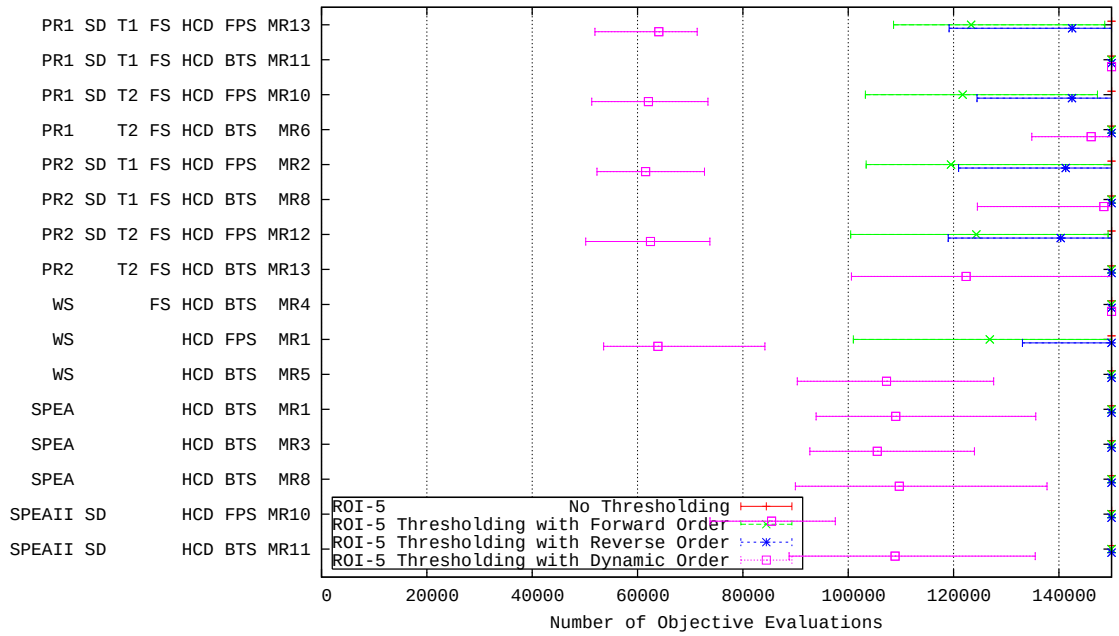

Figure 76: Performance Enhancement of Ordering Methods in Reaching Regions of Interest 5 for Selected Algorithm Combinations Using 10 Knapsacks and 1000 Items

4.1.4 0/1 Knapsack Conclusions

In this section the number of algorithmic combinations examined has been down-selected from 556 to 16 based on the hypervolume of the resulting Pareto fronts. The remaining sixteen combinations were then down-selected to five combinations based on their performance with ICEO problems. The following are the important conclusions reached during this section.

Five algorithm combinations are suggested for further investigation: PR1-SD-T1-FS-HCD-FPS-MR13, PR1-SD-T2-FS-HCD-FPS-MR10, PR2-SD-T1-FS-HCD-FPS-MR2, PR2-SD-T2-FS-HCD-FPS-MR12, and WS-HCD-FPS-MR1. In order to get a relative view of the performance of these algorithms, Table 10 displays the performance rank of each of the algorithms when run with and without dynamic objective thresholding. Again, because the emphasis is on ICEO problems, where multiple optimization runs may not be possible, the measure of performance is the maximum number of objectives required to reach the region of interest for the set of thirty runs. These measures are then normalized by assigning the rank of one to the algorithm combination reaching the region of interest with the fewest objective evaluations, and a rank of ten to the algorithm combination reaching the region of interest with the most objective evaluations. A total of the ranks indicates a total score for the algorithms. Those algorithms with consistently good performance should have the lowest total score. This table indicates the dramatic improvement of those algorithms run with dynamic objective thresholding over those run without dynamic objective thresholding.

Hypercube distance scaling reduces the number of objectives required to reach the region of interest for a wide variety of algorithm combinations. This simple-to-implement component can be added to many of the existing components to help reach the region of interest faster.

Dynamic objective thresholding can reduce the total number of objectives required. This should be caveated that it is most important for problems with many objectives to be solved, i.e., more than two. Also related is that dynamic objective thresholding is best

when the hardest objectives are solved first. To place objectives in the order of increasing difficulty requires either *a priori* knowledge of the problem domain or the ability to learn the difficulty of objectives.

Hypercube distance dynamic objective ordering can reduce the total number of objectives required. Many times dynamic objective ordering can not surpass even the poorest static ordering of objectives. But in several cases, dynamic objective ordering is able to surpass the optimal static objective ordering.

Dynamic objective thresholding is best when combined with fitness proportionate selection. Discussions presented in Section 4.1.3.3 show that the fitness proportionate selection is able to dramatically decrease the probability of mating of individuals without removing them from the population. Binary tournament selection can only reorder the individuals and is thus unable to dramatically reduce the probability of mating for the poor performers.

Table 10: Performance Rank for Five Selected Algorithms With and Without Dynamic Objective Thresholding for Each of the 0/1 Knapsack Problems and Regions of Interest. Lower Scores Are Better.

With Dynamic Objective Thresholding											
Number of Objectives	**2**		**4**			**10**					
Region of Interest	**1**	**2**	**1**	**2**	**3**	**1**	**2**	**3**	**4**	**5**	**Total**
PR1-SD-T1-FS-HCD-FPS-MR13	8	8	3	4	5	1	2	5	4	1	41
PR1-SD-T2-FS-HCD-FPS-MR10	5	5	1	2	4	4	4	4	3	3	35
PR2-SD-T1-FS-HCD-FPS-MR2	6	2	4	1	1	3	5	3	1	2	28
PR2-SD-T2-FS-HCD-FPS-MR12	7	1	5	3	2	5	1	1	2	4	31
WS-HCD-FPS-MR1	9	7	2	5	3	2	3	2	5	5	43
Without Dynamic Objective Thresholding											
PR1-SD-T1-FS-HCD-FPS-MR13	2	8	6	7	8	8	10	7	7	7	70
PR1-SD-T2-FS-HCD-FPS-MR10	1	6	7	9	10	9	8	8	8	8	74
PR2-SD-T1-FS-HCD-FPS-MR2	3	4	8	6	9	7	7	9	9	9	71
PR2-SD-T2-FS-HCD-FPS-MR12	4	3	10	8	7	10	9	10	10	10	81
WS-HCD-FPS-MR1	10	10	9	10	6	6	6	6	6	6	75

4.2 Six Problems of Deb

In 1999 Deb [17] created a set of generic problems for testing MOEAs. Then in 2000 Zitzler, Deb and Thiele [62] created instances of these functions with empirical results for a variety of EAs. Each function is defined using the following relations:

$$\begin{aligned} \text{Find the Pareto front for minimal values} \quad & T_1(x) = (f_1(x_1),\, f_2(x)) \\ \text{subject to} \quad & f_2(x) = g(x_2, \cdots, x_n) \cdot h(f_1(x_1), g(x_2, \cdots, x_n)) \\ \text{where} \quad & x = (x_1, x_2, \cdots, x_n) \end{aligned}$$

This section examines the performance of the five selected algorithm combinations against these algorithms to ensure the wide applicability of the methods. In addition to the five selected algorithm combinations, the SPEA algorithm is included to provide comparison with a standard algorithm. As with the 0/1 Knapsack problem with only two objectives, the ICEO methods applied to these two objective Deb's T functions are also not expected to produce significant improvement. Instead, the Deb's T functions are used to search for significant decreases in performance.

For each evaluation, $40,000$ individuals, or $80,000$ objectives, are evaluated. Each generation contains twenty individuals, and twenty individuals are included in the elite pool. The HCD scaler is given an offset value of 0.2, resulting in a maximum gain of 6, (see section 3.3 for detail of the HCD scaler). Bitwise mutation is included with a bit rate of 0.25 percent. Single point crossover is applied to create eighty percent of children, with clone crossover applied to create the remaining twenty percent of children. The initial population contains 100 individuals created using a random distribution of individuals. Again, thirty runs are made to create a diversity of solutions.

For runs involving dynamic objective thresholding, the weights were set to eighty percent, $\boldsymbol{W} = \{0.8, 0.8\}$. The number of individuals in the historic pool was set to 120, $|H_T| = 120$. The number of individuals evaluated between updates of the thresholding was set to 120, $D_T = 120$. For runs involving dynamic objective ordering, the number

of individuals in the historic pool was set to 500, $|H_T| = 500$. The number of individuals evaluated between updates of the objective order was set to 100, $D_O = 100$. The following sub-sections detail each of the six T functions.

4.2.1 *Deb's T_1 Function*

The first function, T_1, results in a convex Pareto-optimal front. The search space is defined by $n = 30$ and $x_i \in [0, 1]$. The function is defined as follows:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \cdots, x_n) &= 1 + 9 \cdot \frac{\sum_{i=2}^{n} x_i}{n-1} \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} \end{aligned} \tag{121}$$

For this case, the Pareto front occurs when $g(x) = 1$. This results in a Pareto front with $f_1 = x_1$ and $f_2 = 1 - \sqrt{x_1}$, as illustrated in Figure 77. This figure also displays best Pareto front found from the thirty runs as well as the worst Pareto front. The best Pareto front is defined as the Pareto optimal individuals from the set of all individuals found in all thirty runs, i.e., the Pareto optimal individuals of the union of the sets of Pareto optimal solutions found for each of the thirty runs. These are the best solutions that could be found using all thirty runs. The worst Pareto front is defined as the poorest performing individuals from the union of the sets of Pareto optimal solutions found for each of the thirty runs, i.e., those solutions of the union of Pareto optimal solutions that do not dominate any other solutions. Therefore, the worst Pareto front represents the worst performance that can be expected from any of the thirty runs, or the best performance that can be guaranteed from any single run.

The desire of Deb's T functions is to *minimize* the objectives. This causes some problems when using the hypervolume measure directly. It can not be guaranteed that the Pareto optimal front will result in the minimum hypervolume. For example, a single Pareto point

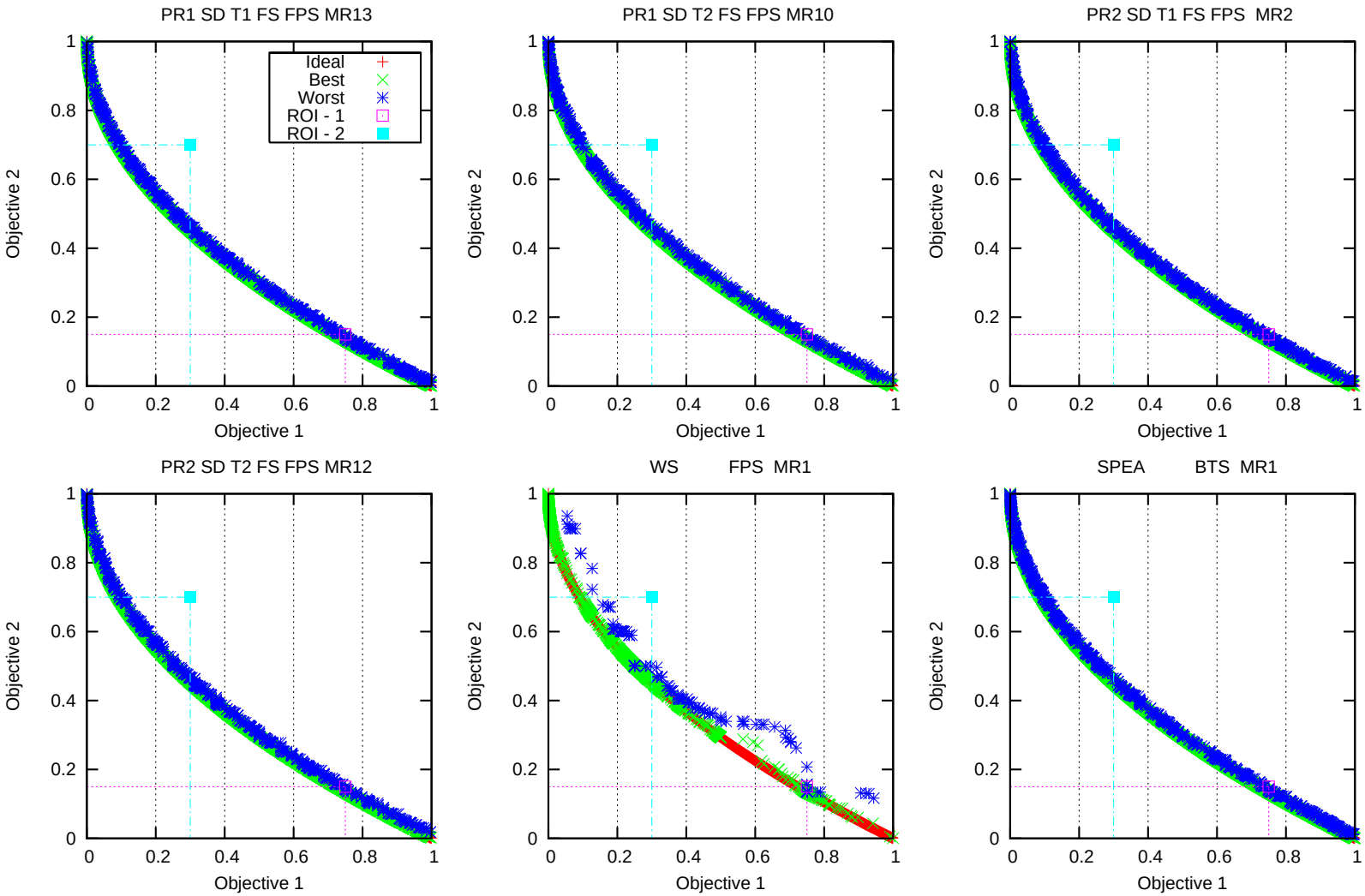

Figure 77: Ideal, Best and Worst Pareto Optimal Fronts for Deb's T_1 Function for Six Selected Algorithm Combinations

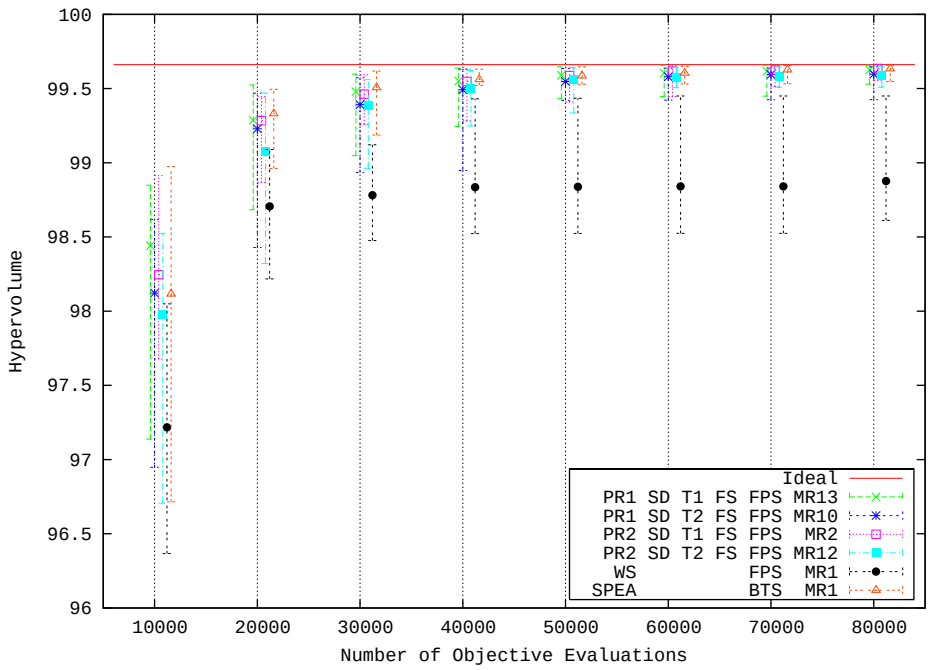

Figure 78: Resulting Hypervolume of T_1 Function for Six Selected Algorithm Combinations

along the front would have a smaller hypervolume than the entire Pareto front. Therefore to calculate the hypervolume, objectives are recast into an increasing function with values greater than zero.

For the T_1 function the hypervolume is calculated by transforming the objective values using:

$$f_{1,mod} = 10 - f_1$$
$$f_{2,mod} = 10 - f_2.$$

Using the ideal curve, the hypervolume approaches a value of 99.66. Figure 78 displays the error-bars after the evaluation for each set of 10, 000 objectives, or 5, 000 individuals. From this figure note that all Pareto rank algorithms perform well with the weighted-sum method producing much less acceptable solutions. None of the combinations have the performance of the SPEA algorithm for the T_1 function.

Two regions of interest are selected for the problem: $\boldsymbol{L_1} = \{0.75, 0.15\}$ and $\boldsymbol{L_2} =$

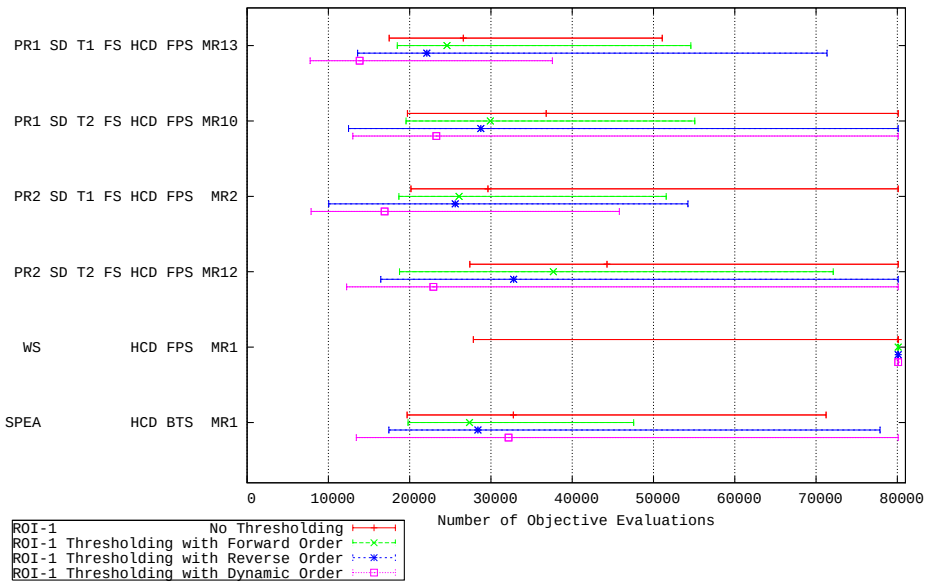

Figure 79: Performance Enhancement of Ordering Methods in Reaching Region of Interest 1 for Six Selected Algorithm Combinations Using Deb's T_1 Function

$\{0.30, 0.70\}$. Figure 77 displayed the locations of these ROIs in objective space along with the resulting solutions found using the algorithms without HCD scaling or dynamic or static objective thresholding. Figures 79 and 80 display the results for the four ordering methods for these two regions of interest. Based on the position of the ROIs in objective space, and the number of objectives required for evaluation, region of interest 1 appears to be more difficult. The use of dynamic objective thresholding with HCD dynamic objective ordering is able to outperform methods without these ICEO methods for a majority of the Pareto rank methods. Use of the ICEO methods with the weighted sum and SPEA methods is detrimental.

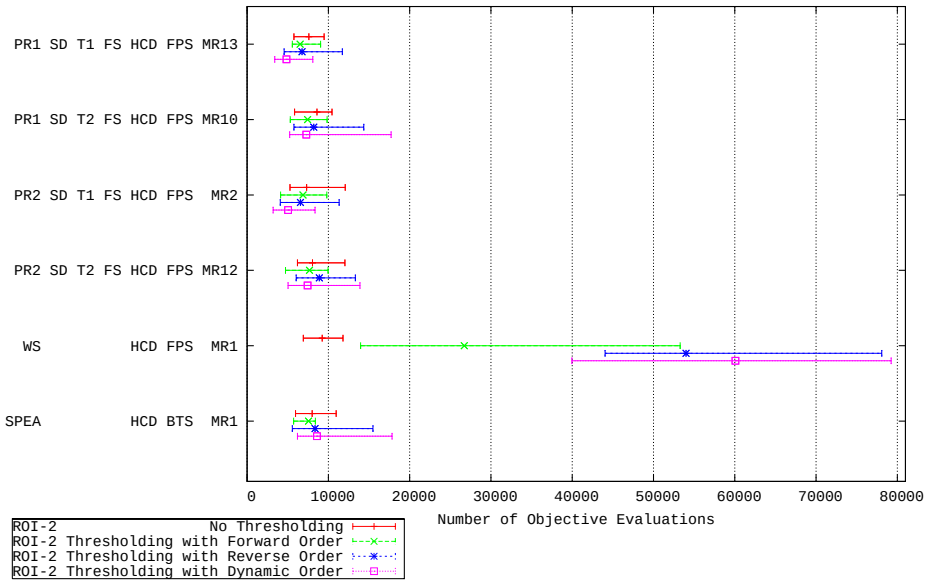

Figure 80: Performance Enhancement of Ordering Methods in Reaching Region of Interest 2 for Six Selected Algorithm Combinations Using Deb's T_1 Function

4.2.2 *Deb's T_2 Function*

The second function, T_2, results in a concave Pareto-optimal front. The search space is defined by $n = 30$ and $x_i \in [0, 1]$. The function is defined as follows:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \cdots, x_n) &= 1 + 9 \cdot \frac{\sum_{i=2}^{n} x_i}{n-1} \\ h(f_1, g) &= 1 - \left(\frac{f_1}{g}\right)^2 \end{aligned} \tag{122}$$

Again, the Pareto front occurs when $g(x) = 1$. This results in a Pareto front with $f_1 = x_1$ and $f_2 = 1 - x_1^2$, as illustrated in Figure 81. From this figure, note that the weighted-sum method displays the largest disparity between the best and worst Pareto solutions found.

For the T_2 function the hypervolume is calculated by transforming the objective values using the same calculation as the T_1 function:

$$\begin{aligned} f_{1,mod} &= 10 - f_1 \\ f_{2,mod} &= 10 - f_2. \end{aligned}$$

Using the ideal curve, the hypervolume approaches a value of 99.31. Figure 82 displays the error-bars after the evaluation for each set of $10,000$ objectives, or $5,000$ individuals. From this figure note that all Pareto rank algorithms perform well with the weighted-sum method producing much less acceptable solutions.

Two regions of interest are selected for the problem: $\boldsymbol{L_1} = \{0.35, 1.10\}$ and $\boldsymbol{L_2} = \{0.75, 0.70\}$. Figures 83 and 84 display the results for the four ordering methods for these two regions of interest. With the first region of interest, the Pareto rank methods are able to realize improvements with dynamic objective thresholding when the objectives are evaluated in the forward order, but not when evaluated in the reverse order. Dynamic objective ordering is able to moderate the results to always improve the performance beyond that of no objective thresholding regardless of the initial objective order. Unfortunately, none of

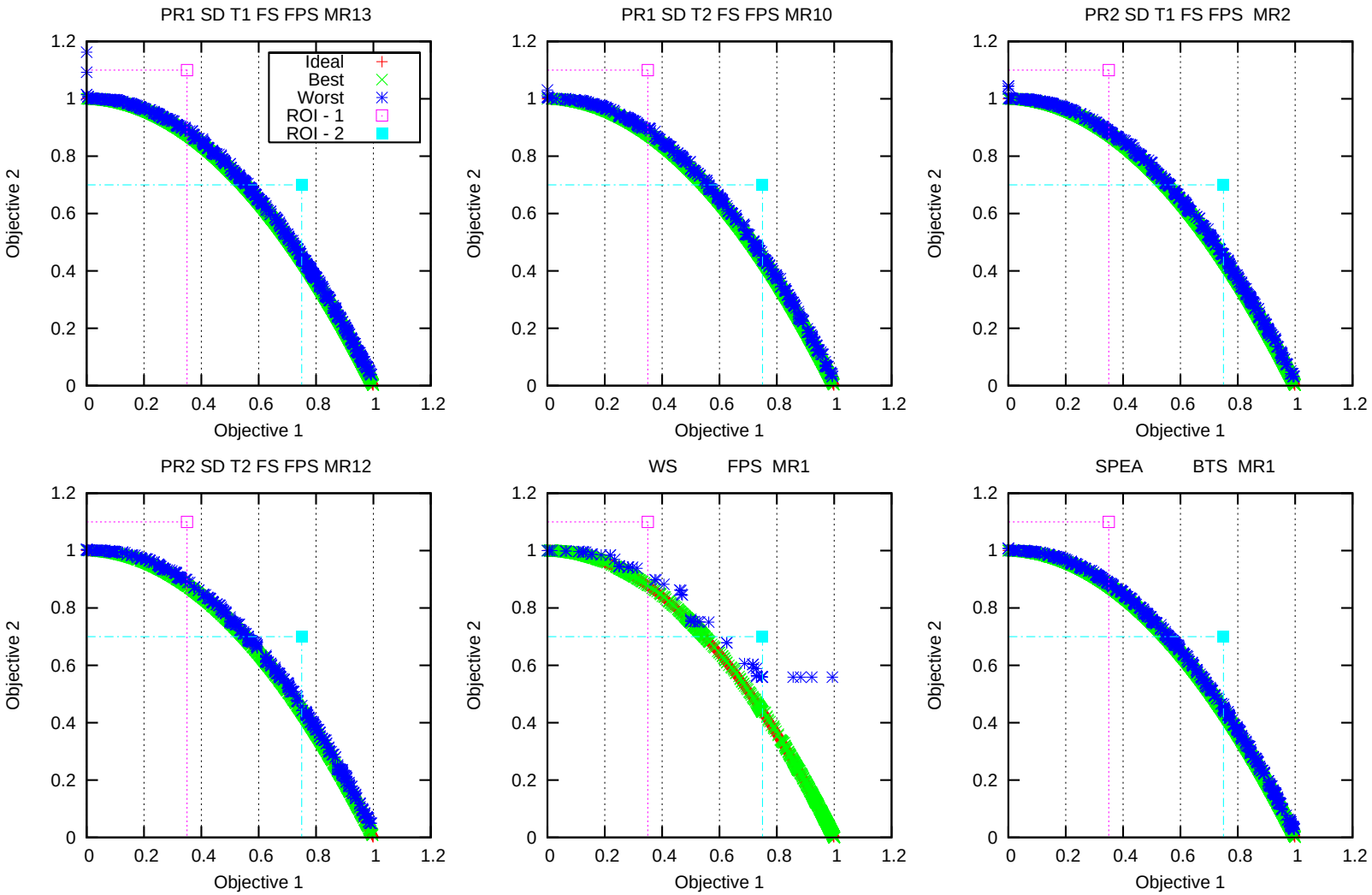

Figure 81: Ideal, Best and Worst Pareto Optimal Fronts for Deb's T_2 Function for Six Selected Algorithm Combinations

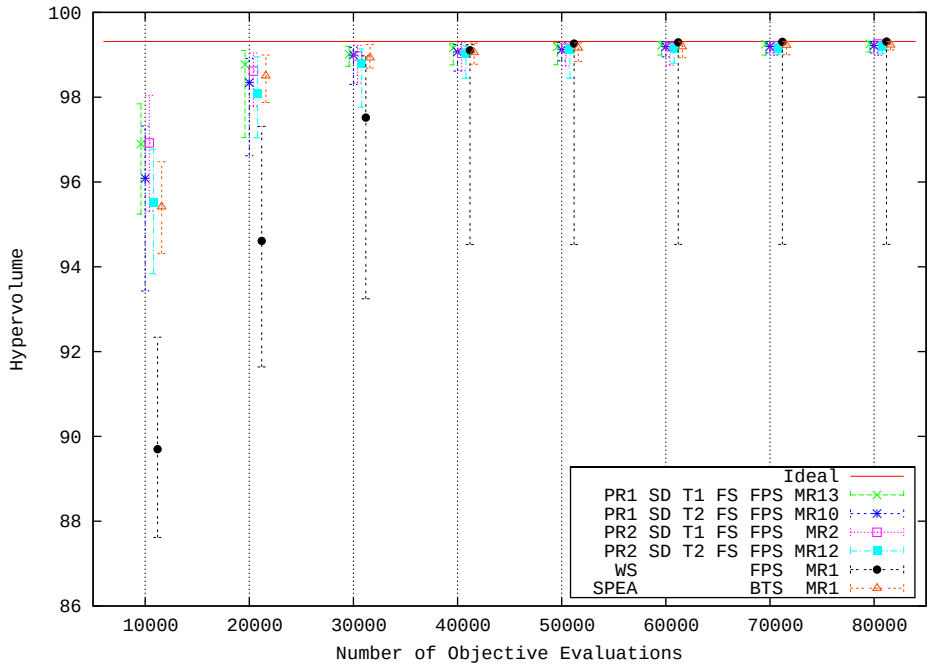

Figure 82: Resulting Hypervolume of T_2 Function for Six Selected Algorithm Combinations

the ICEO methods of hypercube distance scaling, dynamic objective thresholding, or dynamic objective ordering are helpful with the weighted-sum or SPEA algorithms. For the second region of interest, the ICEO techniques were only helpful with the PR1-SD-T1-FS-HCD-FPS-MR13 algorithm.

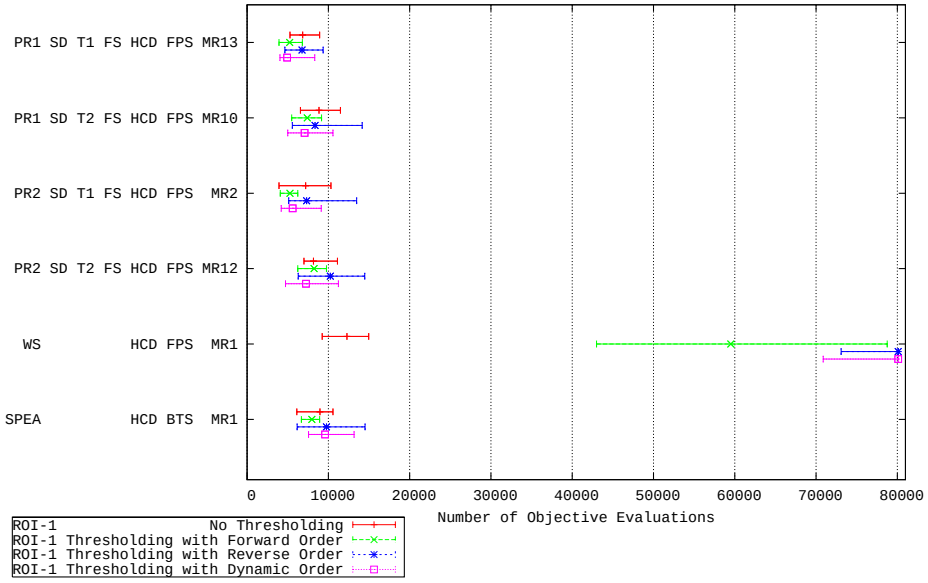

Figure 83: Performance Enhancement of Ordering Methods in Reaching Region of Interest 1 for Six Selected Algorithm Combinations Using Deb's T_2 Function

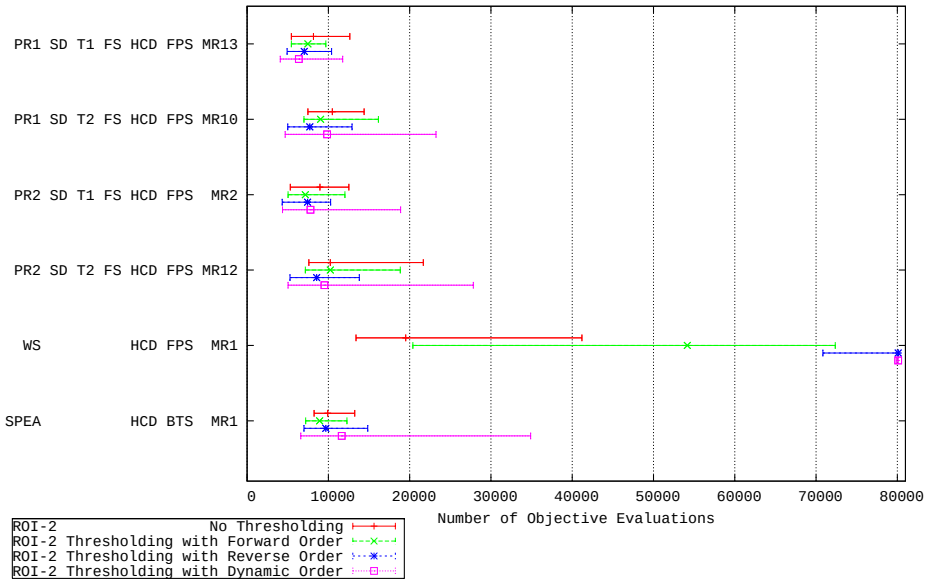

Figure 84: Performance Enhancement of Ordering Methods in Reaching Region of Interest 2 for Six Selected Algorithm Combinations Using Deb's T_2 Function

4.2.3 *Deb's T_3 Function*

The third function, T_3, is designed to test an EA's ability to evolve for a Pareto-optimal front that is not contiguous. The search space is defined by $n = 30$ and $x_i \in [0, 1]$. The function is defined as follows:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \cdots, x_n) &= 1 + 9 \cdot \frac{\sum_{i=2}^{n} x_i}{n-1} \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} - f_1 g \sin(10\pi f_1) \end{aligned} \tag{123}$$

The Pareto front again occurs when $g(x) = 1$. This results in a Pareto front with $f_1 = x_1$ and $f_2 = 1 - \sqrt{x_1} - x_1 \sin(10\pi x_1)$, as illustrated in Figure 85. Note the addition of the sine function in h, which causes the discontinuity with period of $1/5$. The difficulty of this problem is illustrated in Figure 85 by the large disparity between the best and worst solutions for all of the optimization techniques. These results imply that regions of the Pareto optimal front are not found in all thirty runs, and some regions of the Pareto front are not found in any of the thirty-runs.

For the T_3 function the hypervolume is calculated by transforming the objective values using the same calculation as the T_1 function:

$$\begin{aligned} f_{1,mod} &= 10 - f_1 \\ f_{2,mod} &= 10 - f_2. \end{aligned}$$

Using the ideal curve, the hypervolume approaches a value of 107.00. Figure 86 displays the error-bars after the evaluation for each set of $10,000$ objectives, or $5,000$ individuals. From this figure note the much larger deviation in results for all methods implying the difficulty of assuring convergence.

Two regions of interest are selected for the problem: $\boldsymbol{L_1} = \{0.2, 0.6\}$ and $\boldsymbol{L_2} = \{0.7, 0.0\}$. Figures 87 and 88 display the results for the four ordering methods for these two regions of

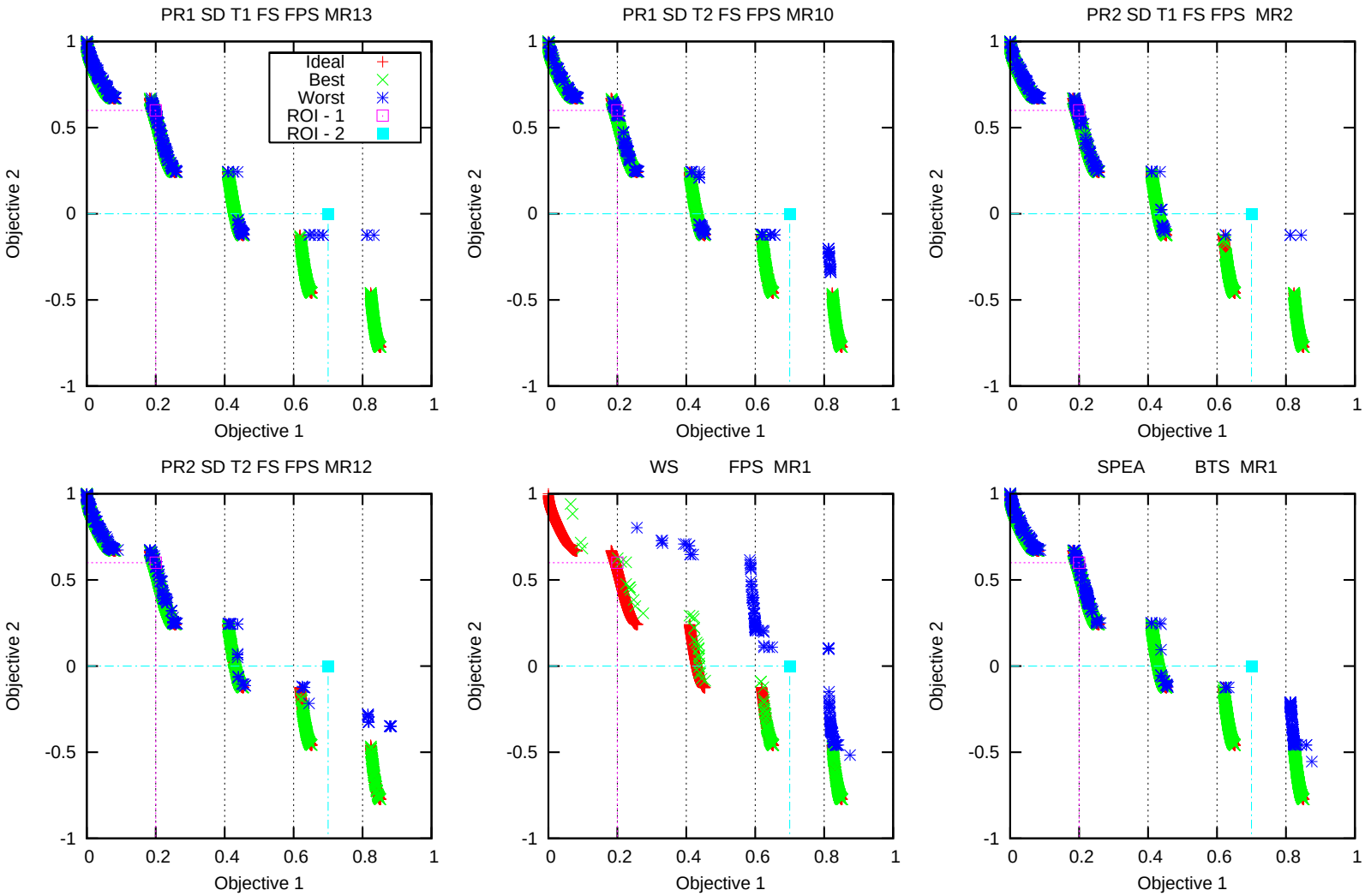

Figure 85: Ideal, Best and Worst Pareto Optimal Fronts for Deb's T_3 Function for Six Selected Algorithm Combinations

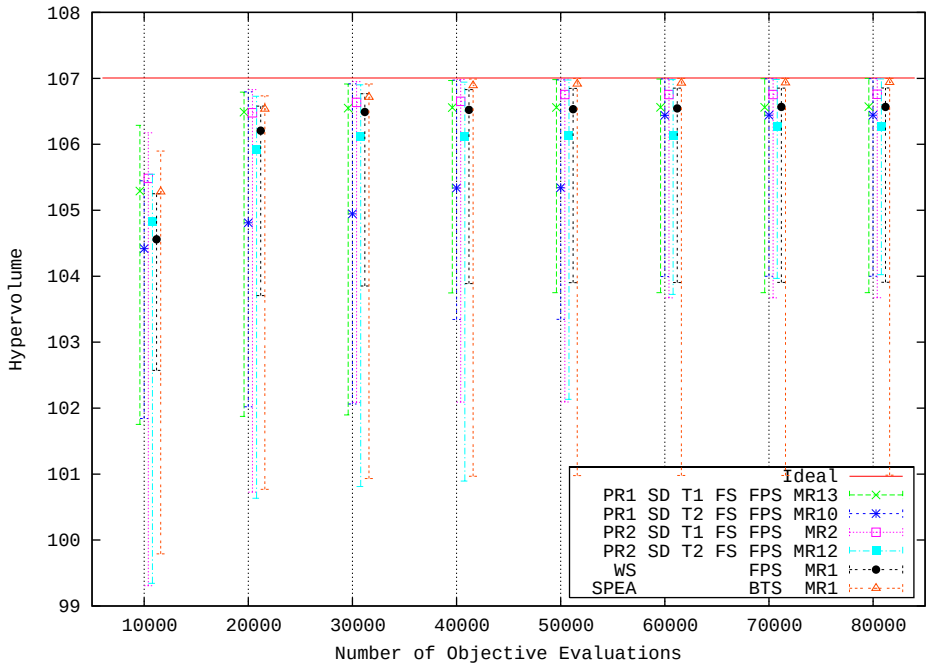

Figure 86: Resulting Hypervolume of T_3 Function for Six Selected Algorithm Combinations

interest. Note that neither region of interest displays better results with dynamic objective ordering.

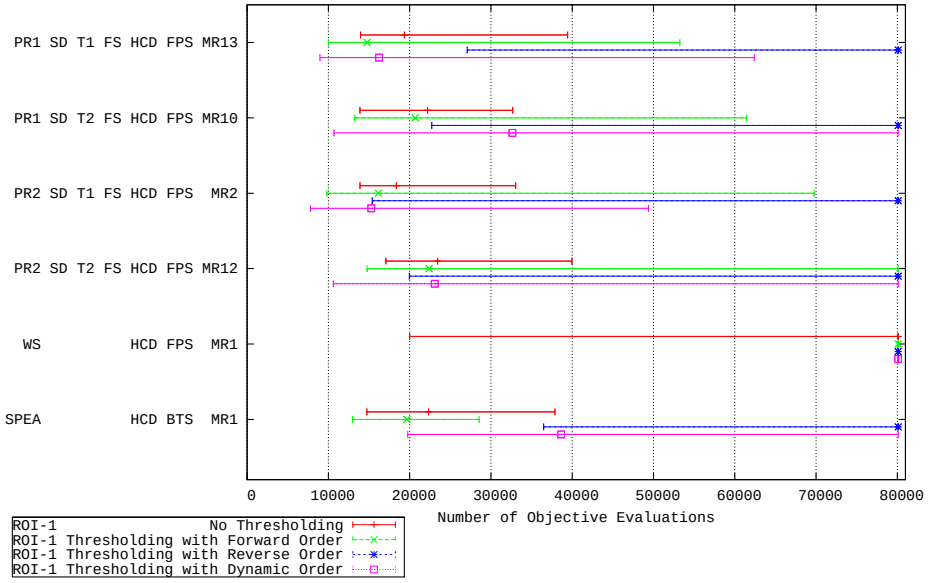

Figure 87: Performance Enhancement of Ordering Methods in Reaching Region of Interest 1 for Six Selected Algorithm Combinations Using Deb's T_3 Function

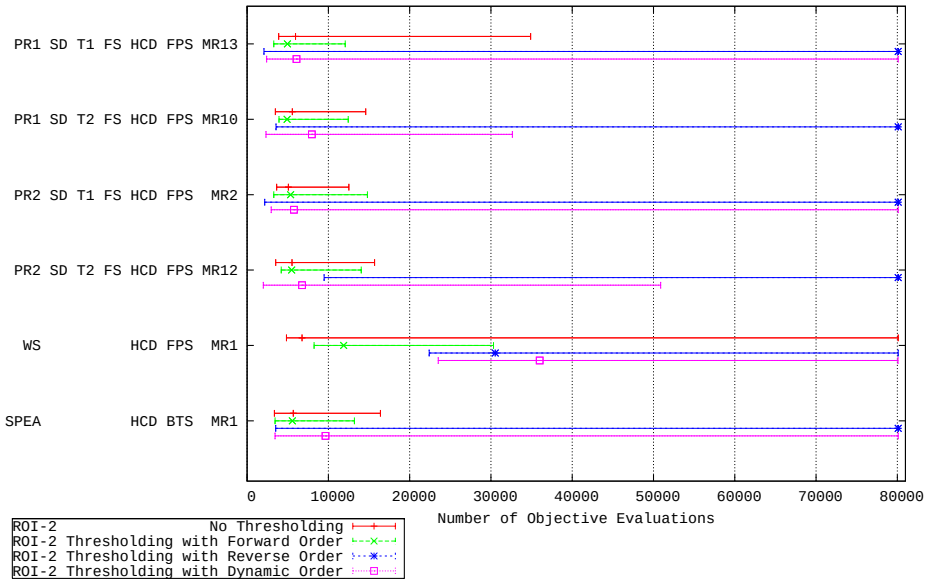

Figure 88: Performance Enhancement of Ordering Methods in Reaching Region of Interest 2 for Six Selected Algorithm Combinations Using Deb's T_3 Function

4.2.4 *Deb's T_4 Function*

The fourth function, T_4, is designed to test an EA's ability to evolve for a multi-modal system. The function contains 21^9 local Pareto-optimal fronts. The search space is defined by $n = 10$, $x_1 \in [0, 1]$ and $x_2, \cdots, x_n \in [-5, 5]$. The function is defined as follows:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \cdots, x_n) &= 1 + 10(n-1) + \sum_{i=2}^{n} \left(x_i^2 - 10\cos(4\pi x_i) \right) \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} \end{aligned} \tag{124}$$

The Pareto front occurs when $g(x) = 1.25$. This results in a Pareto front with $f_1 = x_1$ and $f_2 = 1 - \sqrt{x_1/1.25}$. Figure 89 displays the largest discrepancy between the best and worst Pareto optimal fronts of any of the T functions.

For the T_4 function the hypervolume is calculated by transforming the objective values using:

$$\begin{aligned} f_{1,mod} &= 2 - f_1 \\ f_{2,mod} &= 35 - f_2. \end{aligned}$$

Using the ideal curve, the hypervolume approaches a value of 69.49. Figure 90 displays the error-bars after the evaluation for each set of $10,000$ objectives, or $5,000$ individuals. From Figure 90, note the much larger disparity between the best and worst hypervolume. Close examination of Figures 89 and 90 reveals the superiority of the PR2-SD-T2-FS-FPS-MR12 algorithm for the best, worst and average performance.

Two regions of interest are selected for the problem: $\boldsymbol{L_1} = \{0.2, 12.0\}$ and $\boldsymbol{L_2} = \{0.8, 12.0\}$. Figures 91 and 92 display the results for the four ordering methods for these two regions of interest. Neither of the regions of interest is predictably found using any method.

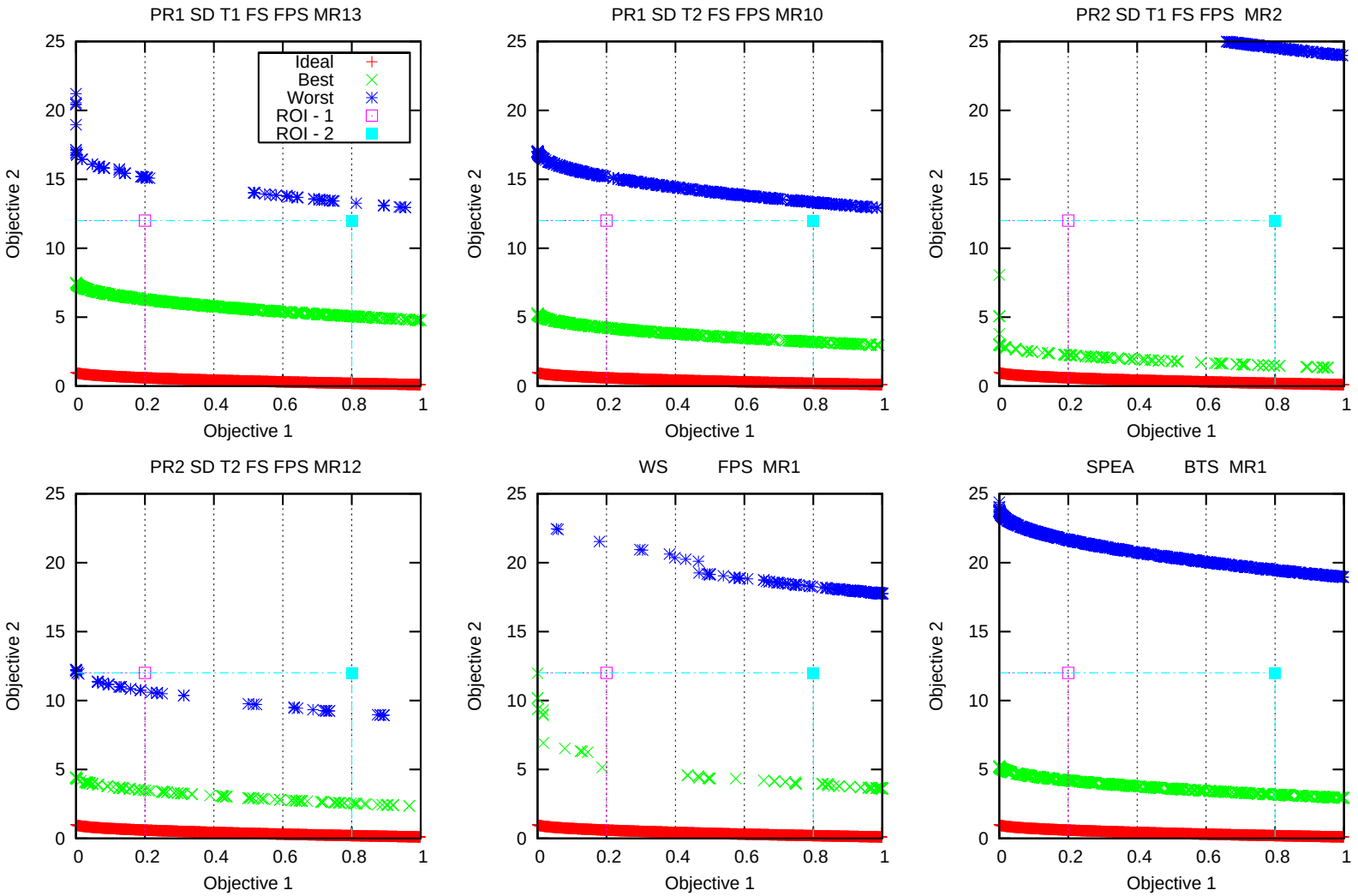

Figure 89: Ideal, Best and Worst Pareto Optimal Fronts for Deb's T_4 Function for Six Selected Algorithm Combinations

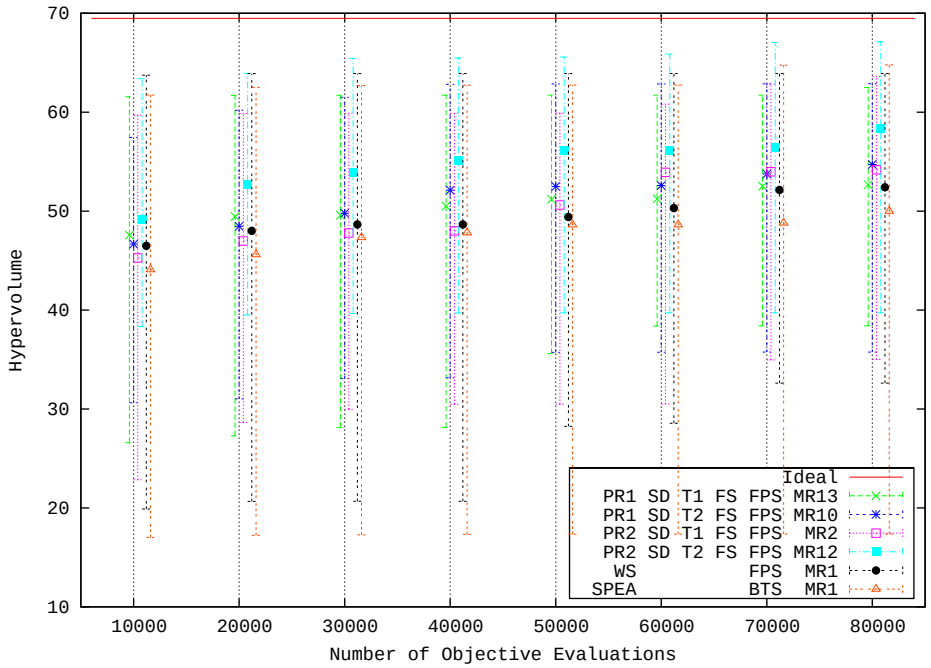

Figure 90: Resulting Hypervolume of T_4 Function for Six Selected Algorithm Combinations

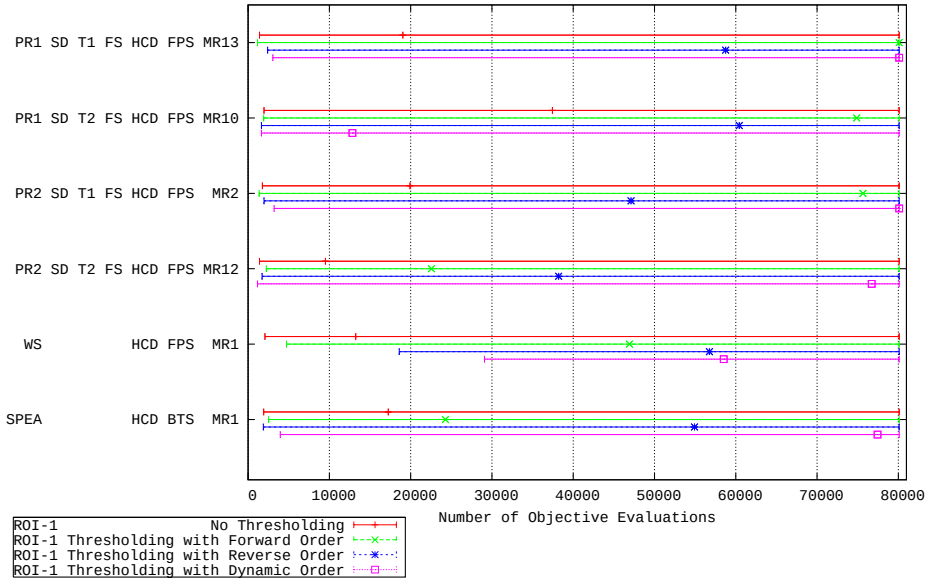

Figure 91: Performance Enhancement of Ordering Methods in Reaching Region of Interest 1 for Six Selected Algorithm Combinations Using Deb's T_4 Function

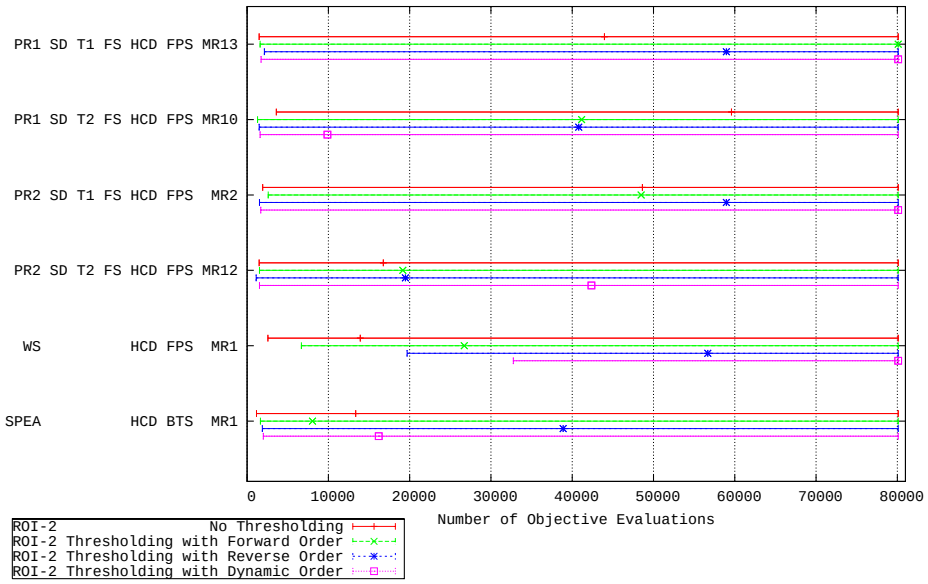

Figure 92: Performance Enhancement of Ordering Methods in Reaching Region of Interest 2 for Six Selected Algorithm Combinations Using Deb's T_4 Function

4.2.5 *Deb's T_5 Function*

The fifth function, T_5, is designed to be deceptive. The search space is defined as a binary string with $n = 11$, x_1 is a 30 bit string, $x_1 \in \{0,1\}^{30}$ and x_2 through x_n are 5 bit strings, $x_2, \cdots, x_n \in \{0,1\}^5$. The function is defined as follows:

$$\begin{aligned} f_1(x_1) &= 1 + u(x_1) \\ g(x_2, \cdots, x_n) &= \sum_{i=2}^{n} v(u(x_i)) \\ h(f_1, g) &= \frac{1}{f_1} \end{aligned} \tag{125}$$

The function $u(x_i)$ is the unitation, or number of ones in the binary string x_i. The $v(u(x_i))$ function, defined as

$$v(u(x_i)) = \begin{cases} 2 + u(x_i) & \text{if } u(x_i) < 5 \\ 1 & \text{if } u(x_i) = 5 \end{cases} \tag{126}$$

results in the deceptive nature. Note that $v(u(x_i))$ is minimal with a value $v(u(x_i)) = 1$ when x_i is all ones, and $v(u(x_i))$ is maximum with a value $v(u(x_i)) = 6$ when only one zero is included in x_i. Figure 93 displays the performance of each of the algorithms in finding the region of interest. Also note that all methods miss sections of the Pareto front sometimes, but that the PR2-SD-T2-FS-FPS-MR12, weighted-sum and SPEA algorithms always miss sections of the Pareto front.

For the T_5 function the hypervolume is calculated by transforming the objective values using:

$$\begin{aligned} f_{1,mod} &= 35 - f_1 \\ f_{2,mod} &= 11 - f_2. \end{aligned}$$

Using the ideal curve, the hypervolume approaches a value of 332.72. Figure 94 displays the error-bars after the evaluation for each set of $10,000$ objectives, or $5,000$ individuals.

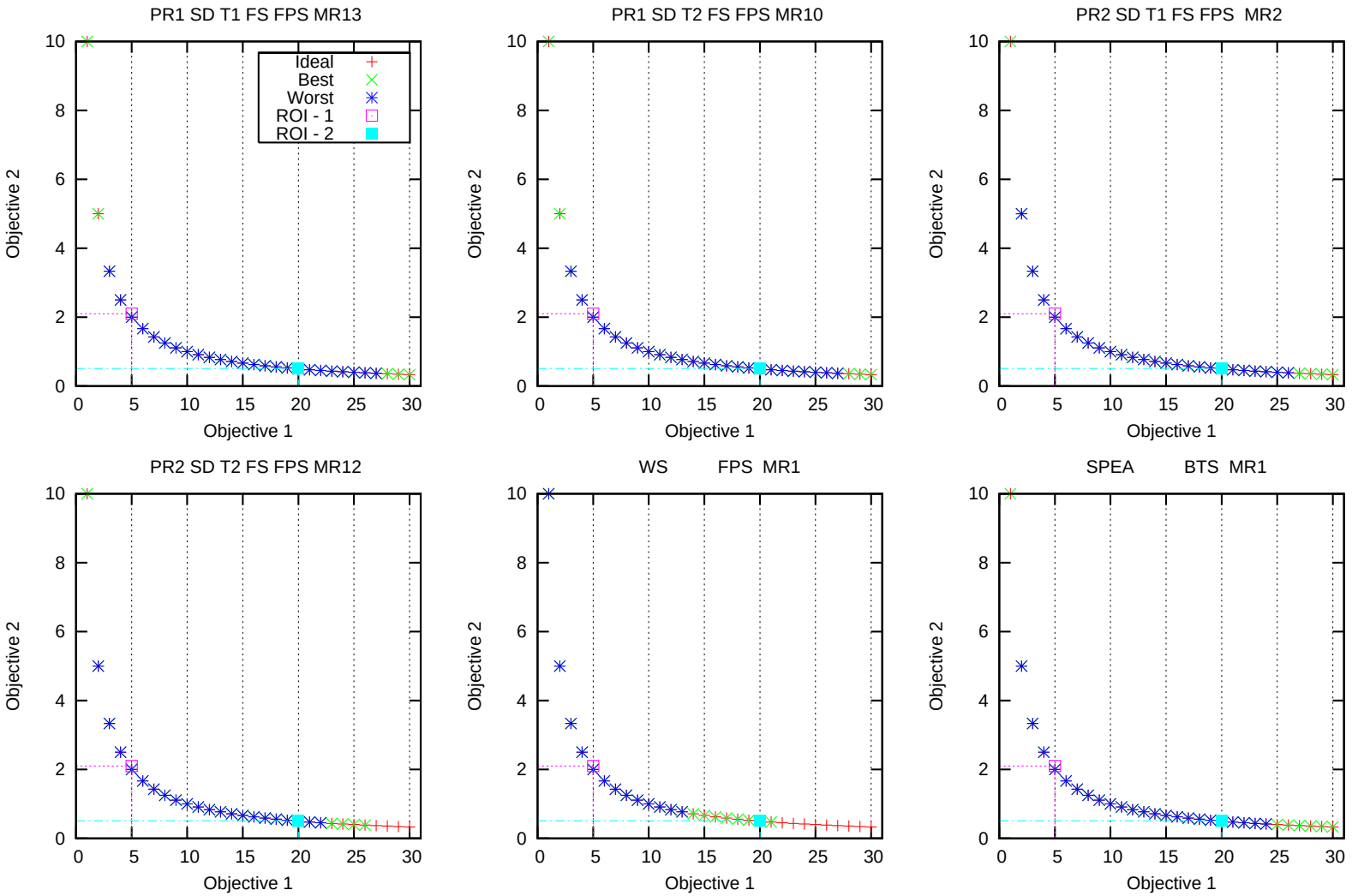

Figure 93: Ideal, Best and Worst Pareto Optimal Fronts for Deb's T_5 Function for Six Selected Algorithm Combinations

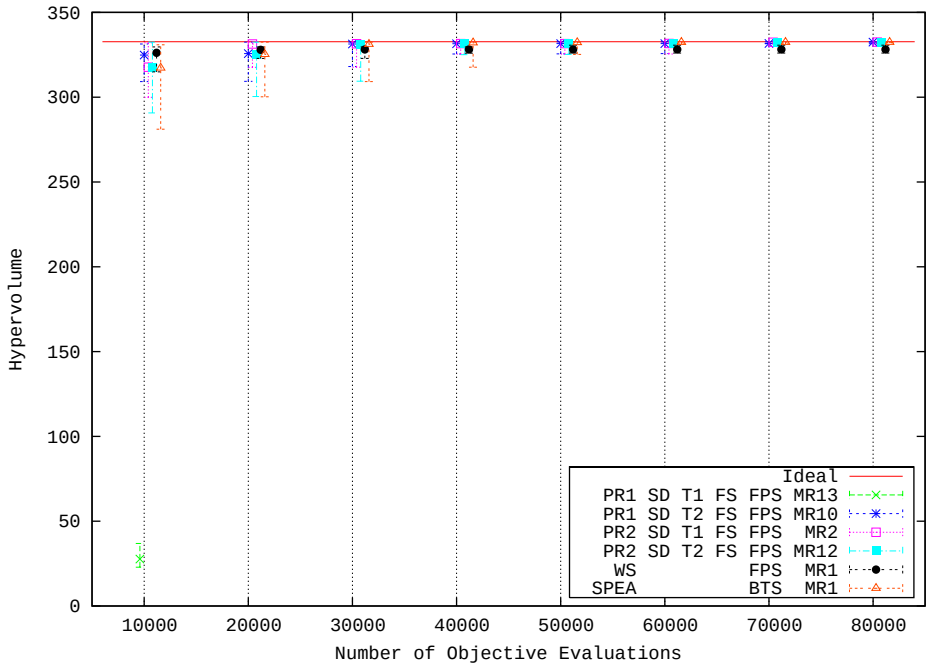

Figure 94: Resulting Hypervolume of T_5 Function for Six Selected Algorithm Combinations

From this figure note that all Pareto rank algorithms perform well with the weighted-sum method producing much less acceptable solutions. None of the combinations have the performance of the SPEA algorithm for the T_5 function.

Two regions of interest are selected for the problem: $\boldsymbol{L_1} = \{5.0, 2.1\}$ and $\boldsymbol{L_2} = \{20.0, 0.51\}$. Figures 95 and 96 display the results for the four ordering methods for these two regions of interest. Both regions of interest display an improved performance with dynamic objective ordering for all algorithm combinations except the weighted-sum.

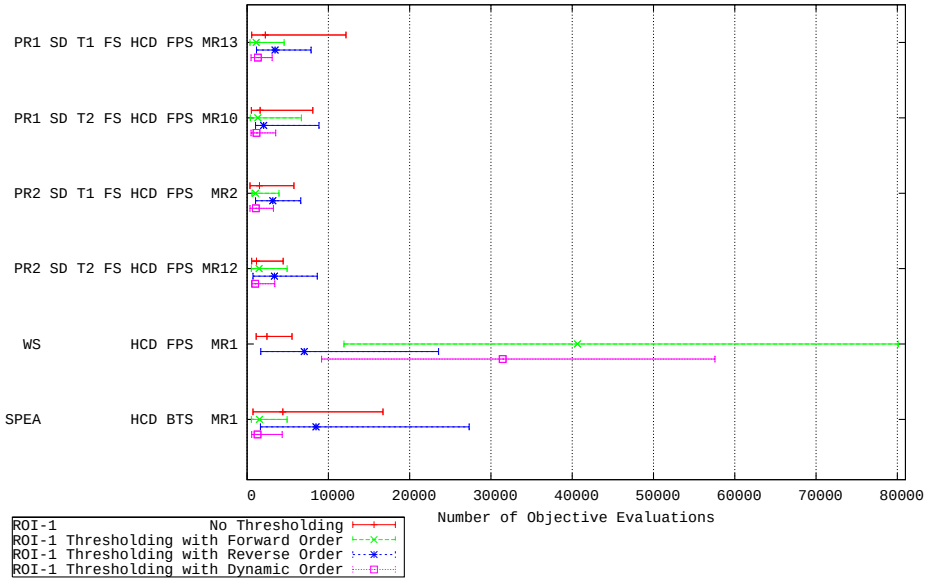

Figure 95: Performance Enhancement of Ordering Methods in Reaching Region of Interest 1 for Six Selected Algorithm Combinations Using Deb's T_5 Function

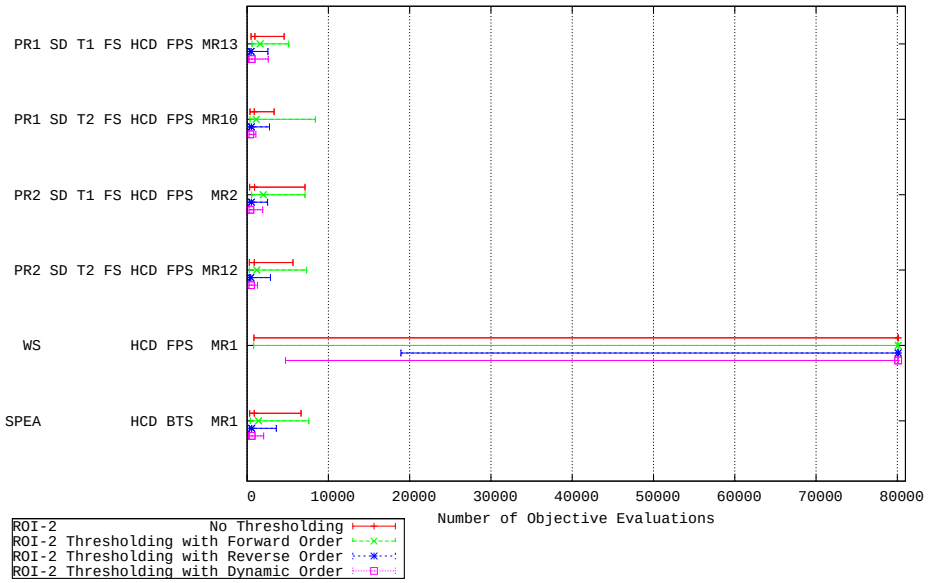

Figure 96: Performance Enhancement of Ordering Methods in Reaching Region of Interest 2 for Six Selected Algorithm Combinations Using Deb's T_5 Function

4.2.6 *Deb's T_6 Function*

The sixth function, T_6, is designed to test two difficulties. First, solutions are not distributed evenly along the Pareto-optimal front. Second, the density of solutions is lowest near the front and highest away from the front. The search space is defined as $n = 10$, and $x_i \in [0, 1]$, and $x_2, \cdots, x_n \in \{0, 1\}^5$. The function is defined as follows:

$$\begin{aligned} f_1(x_1) &= 1 - \frac{e^{-4x_i}}{sin^6(6\pi x_1)} \\ g(x_2, \cdots, x_n) &= 1 + 9 \cdot \frac{\sum_{i=2}^{n} x_i}{n-1} \\ h(f_1, g) &= 1 - \left(\frac{f_1}{g}\right)^2 \end{aligned} \tag{127}$$

The Pareto front occurs when $g(x) = 1$. Figure 97 displays the results for each of the optimization methods. From this figure note that a large section of the Pareto front is never found. Also note the difference between the best and worst Pareto fronts for several of the methods.

For the T_6 function the hypervolume is calculated by transforming the objective values using:

$$\begin{aligned} f_{1,mod} &= 10 - f_1 \\ f_{2,mod} &= 10 - f_2. \end{aligned}$$

Using the ideal curve, the hypervolume approaches a value of 69.49. Figure 98 displays the error-bars after the evaluation for each set of $10,000$ objectives, or $5,000$ individuals. From this figure note that all Pareto rank algorithms perform well with the weighted-sum method producing much less predictable solutions.

Two regions of interest are selected for the problem: $\boldsymbol{L_1} = \{0.2, 2.0\}$ and $\boldsymbol{L_2} = \{0.8, 1.0\}$. Figures 99 and 100 display the results for the four ordering methods for these two regions of interest. Region of interest two is readily found, but objective thresholding and dynamic

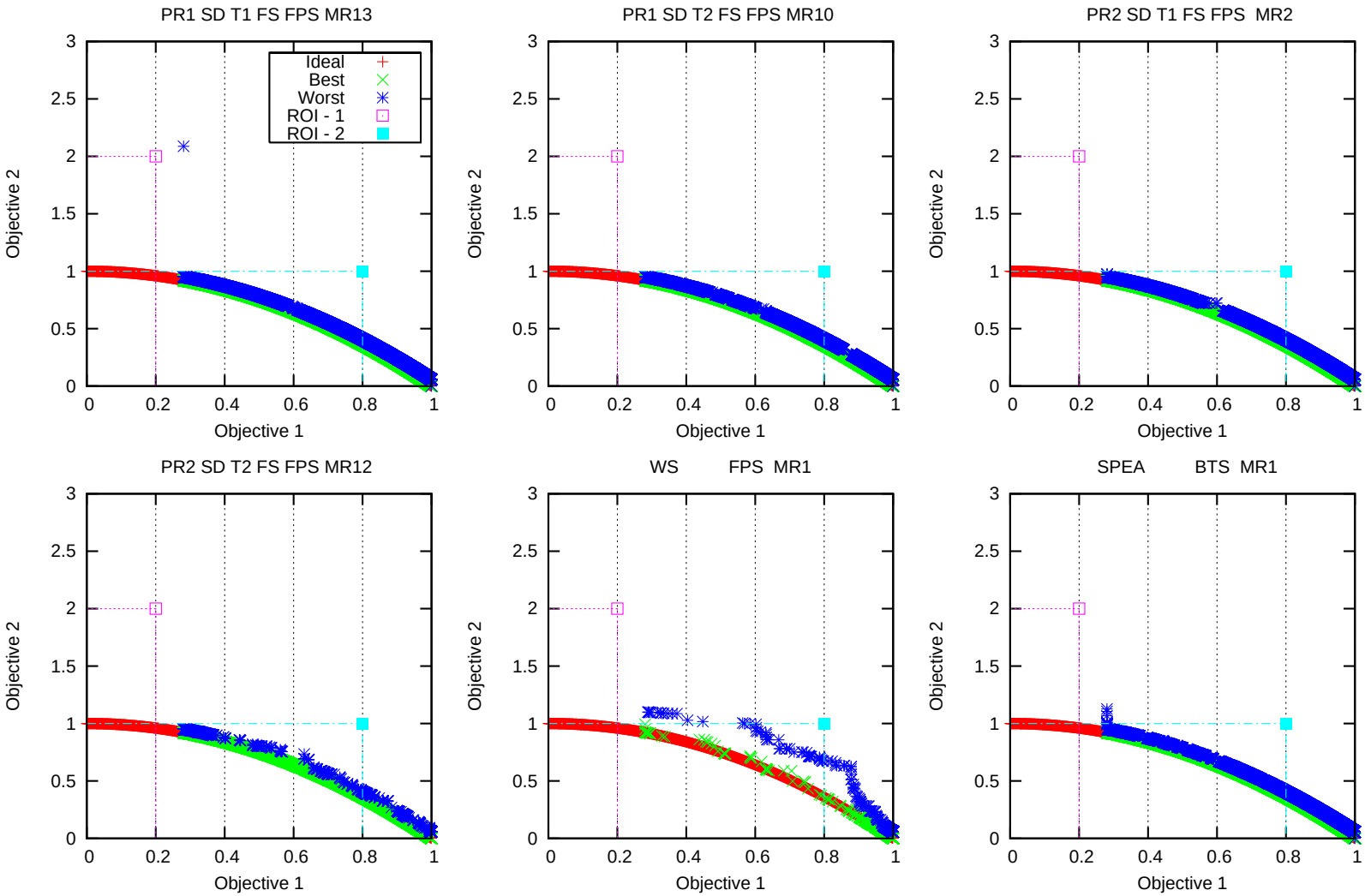

Figure 97: Ideal and Derived Pareto Optimal Fronts for Deb's T_6 Function for Six Selected Algorithm Combinations

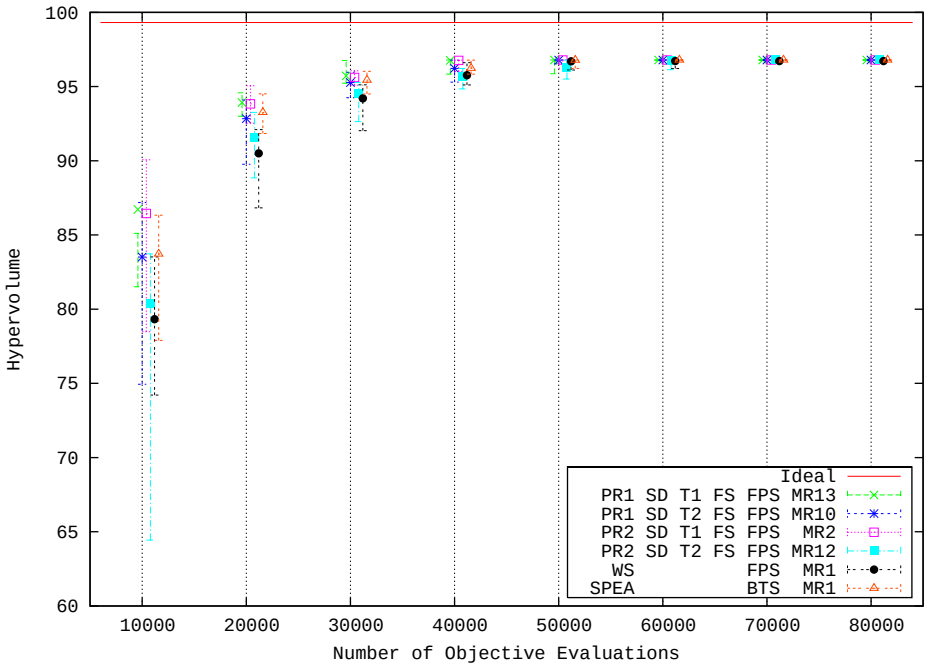

Figure 98: Resulting Hypervolume of T_6 Function for Six Selected Algorithm Combinations

objective ordering only improve the performance for the PR2-SD-T2-FS-FPS-MR12 algorithm combination.

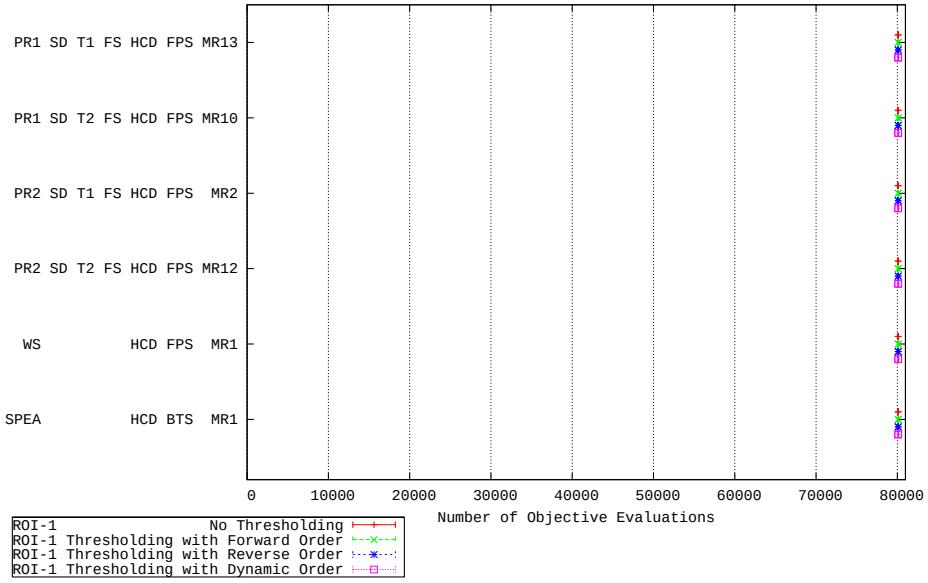

Figure 99: Performance Enhancement of Ordering Methods in Reaching Region of Interest 1 for Six Selected Algorithm Combinations Using Deb's T_6 Function

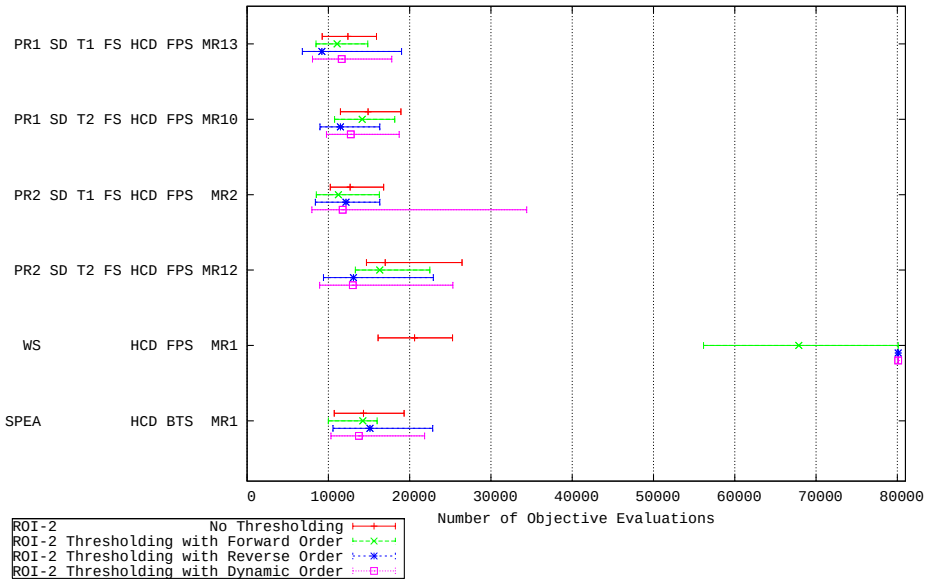

Figure 100: Performance Enhancement of Ordering Methods in Reaching Region of Interest 2 for Six Selected Algorithm Combinations Using Deb's T_6 Function

4.2.7 *Results of Deb's Functions*

The use of the Deb's T functions was twofold. The first purpose was to verify that the base optimization algorithms proposed, without ICEO enhancements, perform well against a variety of optimization problems. The second purpose was to verify the ICEO enhancements perform as expected with a variety of optimization problems.

To measure the success of the first purpose, a modified hypervolume was used. Table 11 ranks the poorest performance from each of the thirty runs for each of the six algorithm combinations against each of the six Deb's T functions at the end of the evaluation of $80,000$ objectives. From this table note that the SPEA algorithm, although the best at the T_1, T_2 and T_5 problems, performed the poorest on T_3, and T_4. The other surprising result was that the PR1-SD-T1-FS-HCD-FPS-MR13 algorithm, which performed well for most problems, performed the poorest on the deceptive T_5 problem. *All Pareto rank base algorithm components, without the ICEO enhancements, performed well against the diverse Deb's T functions*. This implied the weighted-sum algorithm should not be included.

Table 11: Rank of the Performance for Each of the Thirty Runs for Each of the Six Algorithm Combinations the Six Deb's T Functions (1 has the best performance, 6 has the poorest performance)

Algorithm	T_1	T_2	T_3	T_4	T_5	T_6	**Total**
PR1-SD-T1-FS-FPS-MR13	3	2	4	2	6	2	19
PR1-SD-T2-FS-FPS-MR10	5	3	2	3	2	4	19
PR2-SD-T1-FS-FPS-MR2	2	5	5	4	3	1	20
PR2-SD-T2-FS-FPS-MR12	4	4	1	1	4	5	19
WS-FPS-MR1	6	6	3	5	5	6	31
SPEA-BTS-MR1	1	1	6	6	1	3	18

To analyze the second purpose of verifying the ICEO enhancements performed as expected with a variety of optimization problems, Table 12 displays the ranked results of the evolution with and without dynamic objective thresholding. Those solutions that did not reach the region of interest in the $80,000$ objective evaluations were given the largest rank

value; i.e., a value of 12. The 0/1 Knapsack results given in Table 10 indicated improvements in performance when using dynamic objective thresholding with only two objectives were not expected. But, the results of PR1-SD-T1-FS-HCD-FPS-MR13 and PR2-SD-T1-FS-HCD-FPS-MR2 show an improvement in reaching the region of interest with dynamic objective thresholding. Overall, the only dramatic decreases in performance with ICEO methods were with the WS-HCD-FPS-MR1 and SPEA-HCD-BTS-MR1 algorithms. In conclusion, *all Pareto rank methods are applicable to ICEO improvements.* Therefore, all Pareto rank methods have met the two purposes of the Deb's T functions and are candidates for use in ICEO optimization problems.

Table 12: Rank of the Poorest Performance for Each of the Thirty Runs for Each of the Six Algorithm Combinations of the Six Deb's T Functions (1 has the best performance, 6 has poorest performance)

With Dynamic Objective Thresholding													
Algorithm	T_1		T_2		T_3		T_4		T_5		T_6		
Region of Interest	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**Total**
PR1-SD-T1-FS-HCD-FPS-MR13	1	1	1	1	7	9	12	12	1	5	12	3	65
PR1-SD-T2-FS-HCD-FPS-MR10	12	10	5	8	6	10	12	12	4	1	12	4	96
PR2-SD-T1-FS-HCD-FPS-MR2	2	2	3	6	8	5	12	12	2	3	12	11	78
PR2-SD-T2-FS-HCD-FPS-MR12	12	8	8	9	9	7	12	12	3	2	12	9	103
WS-HCD-FPS-MR1	12	12	12	12	11	8	12	12	12	11	12	12	138
SPEA-HCD-BTS-MR1	12	11	10	10	10	11	12	12	5	4	12	7	116
Without Dynamic Objective Thresholding													
PR1-SD-T1-FS-HCD-FPS-MR13	3	3	2	3	4	6	12	12	10	7	12	1	75
PR1-SD-T2-FS-HCD-FPS-MR10	12	4	9	5	1	2	12	12	9	6	12	5	89
PR2-SD-T1-FS-HCD-FPS-MR2	12	8	4	2	2	1	12	12	6	10	12	2	83
PR2-SD-T2-FS-HCD-FPS-MR12	12	7	7	7	5	3	12	12	8	8	12	10	103
WS-HCD-FPS-MR1	12	6	11	11	12	12	12	12	7	12	12	8	127
SPEA-HCD-BTS-MR1	4	5	6	4	3	4	12	12	11	9	12	6	88

4.3 Conclusions

Section 4.1 utilized the 0/1 Knapsack problem to reduce the number of algorithmic combinations examined from 556 to 16 based on the hypervolume of the resulting Pareto fronts. The 0/1 Knapsack problem then down-selected the remaining sixteen combinations to five combinations based on their performance with ICEO problems, resulting in the following five functions to be evaluated: PR1-SD-T1-FS-HCD-FPS-MR13, PR1-SD-T2-FS-HCD-FPS-MR10, PR2-SD-T1-FS-HCD-FPS-MR2, PR2-SD-T2-FS-HCD-FPS-MR12, and WS-HCD-FPS-MR1. Table 10 detailed the ranking of these functions for the various 0/1 Knapsack problems and regions of interest.

Section 4.2 then utilized the six Deb's T function problems to test the robustness of the five selected algorithms to ICEO and non-ICEO optimization. Tables 11 and 12 presented a summary of these results, which found the weighted-sum to not be a robust algorithm.

Table 13 ranks the performance of the remaining four Pareto combinations. Again, the rank is based on the largest number of objective evaluations required to reach the region of interest for the set of thirty runs executed. From this table, there are no clear winners or losers. Each algorithm performs best and worst on different regions of interest. This indicates a non-dependence of the fundamental underlying fitness methods. The analysis of the 0/1 Knapsack problem indicated the advantages of the SPEA II density methods and NPGA fitness sharing algorithms with all Pareto rank combinations. The robustness analysis has shown there is little difference between any combinations of the Pareto rank 1 versus Pareto rank 2 and the Transformation 1 versus Transformation 2 methods.

The inconclusiveness of a clear leader based on performance and robustness implies that the decision can be based on other factors. Chapters 2 and 3 performed a great deal of complexity analysis, which culminated in the results presented in Table 4. Since most components are identical and Transformation 1 and Transformation 2 should have the same complexity, the remaining component is the Pareto rank, which was quite different. Recall

the complexity for Pareto rank 1 is

$$(2 + 2q)|P|^2 - (1 + 2q)|P|,$$

whereas the complexity for Pareto rank 2 is

$$\frac{2 + 2q}{3}|P|^3 + |P|^2 + \frac{10 - 2q}{3}|P|.$$

Therefore, there is a reduction of complexity with the Pareto rank 1 methods which should result in better run-times.

The total complexity of the fitness algorithms for the PR1-SD-T1-FS-HCD-FPS-MR13 and PR1-SD-T2-FS-HCD-FPS-MR10 becomes:

$$\begin{aligned} C(PR1 - SD - T - FS - HCD) &= C(PR1) + C(SD) + C(T2) + C(FS) + C(HCD) \\ &= (2 + 2q)|P|^2 - (1 + 2q)|P| + \\ &\quad |P|^3 + (1 + q)|P|^2 - (1 + q)|P| + \\ &\quad |P|\,(2 + 3(|LID| - 1)) + \\ &\quad (2 + q)|P|^2 - (1 + q)|P| + \\ &\quad (1 + 3q)\,|P| \\ &= |P|^3 + (5 + 4q)|P|^2 + (1q + 3(|LID| - 1))\,|P| \quad (128) \end{aligned}$$

There is still a dependence on $|P|^3$; but it is less than the $(2q/3)|P|^3$ term that would arise from the use of the Pareto rank 2 method.

The conclusions for this chapter are:

1. *Algorithm combinations PR1-SD-T1-FS-HCD-FPS-MR13 and PR1-SD-T2-FS-HCD-FPS-MR10 are robust ICEO algorithm combinations.*

2. *Hypercube distance scaling reduces the number of objectives required to reach the region of interest for a wide variety of algorithm combinations.*

3. *Dynamic objective thresholding can reduce the total number of objectives required, especially when there are more than two objectives to be solved.*

4. *Dynamic objective thresholding is best when the hardest objectives are solved first.* To place objectives in the order of increasing difficulty requires either *a priori* knowledge of the problem domain or the ability to learn the difficulty of objectives.

5. *Hypercube distance dynamic objective ordering can reduce the total number of objectives required.*

6. *Dynamic objective thresholding is best when combined with fitness proportionate selection.*

Table 13: Performance Rank for Pareto Rank Selected Algorithms for each of the 0/1 Knapsack and Deb's T Function Problems and each Region Interest

O/1 Knapsack Results											
Number of Objectives	**2**		**4**			**10**					
Region of Interest	**1**	**2**	**1**	**2**	**3**	**1**	**2**	**3**	**4**	**5**	**Total**
PR1-SD-T1-FS-HCD-FPS-MR13	4	4	2	4	4	1	2	4	4	1	30
PR1-SD-T2-FS-HCD-FPS-MR10	1	3	1	2	3	3	3	3	3	3	25
PR2-SD-T1-FS-HCD-FPS-MR2	2	2	3	1	1	2	4	2	1	2	20
PR2-SD-T2-FS-HCD-FPS-MR12	3	1	4	3	2	4	1	1	2	4	25

Deb's T Function Results													
Algorithm	T_1		T_2		T_3		T_4		T_5		T_6		
Region of Interest	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**Total**
PR1-SD-T1-FS-HCD-FPS-MR13	1	1	1	1	2	3	4	4	1	4	4	1	27
PR1-SD-T2-FS-HCD-FPS-MR10	4	4	3	3	1	4	4	4	4	1	4	2	38
PR2-SD-T1-FS-HCD-FPS-MR2	2	2	2	2	3	1	4	4	2	3	4	4	33
PR2-SD-T2-FS-HCD-FPS-MR12	4	3	4	4	4	2	4	4	3	2	4	3	41

CHAPTER V

FLARE PATTERN OPTIMIZATION

Chapter 1 introduced the basics of MOEA optimization. Chapter 2 provided descriptions of several of the existing MOEA methods. Chapter 3 presented the details of the ICEO algorithms contributed by this thesis. Chapter 4 presented the evaluation and down-selection of combinations of methods developed by this research using fast-running 0/1 Knapsack and Deb's T functions. All of these chapters now reach a climax in a true ICEO application.

The resulting good performing algorithm combinations from Chapter 4 are now applied in this chapter to the optimization of flare patterns for aircraft self-protection. Flares are expendable pyrotechnic or other heat generating devices ejected from aircraft to confuse missile seekers into losing track of their target, and thus missing the targeted aircraft. Flare patterns are optimized by traversing the search space of flare pattern attributes in search of a flare pattern that is effective at multiple aspect angles and ranges.

Unlike the evaluation of the 0/1 Knapsack and Deb's T functions, flare pattern objective evaluation contains objectives that are independent and computationally expensive, i.e., they require a separate, computationally expensive simulation for the evaluation of each objective. Additionally, objective run-times are dependent on both the objective being evaluated and the genome being evaluated, i.e., run-times are dependent on the threat launch position and the effectiveness of the flare pattern. Because of these dependencies, the problem provides a valid and challenging evaluation of the dynamic objective ordering methods.

Section 5.1 provides a description of the flare pattern optimization problem. This section details the search space of flare attributes, which are flare types, flare dispensers, and flare timing. The section also describes the Imaging Seeker And Missile Simulation

ISAMS, which is the function evaluator for this problem. The objectives created by the function evaluator are the performances of the flare patterns against simulated missiles launched from various positions.

Section 5.2 details the performance of different sets of the algorithms against the flare pattern design problem. Use of this truly ICEO problem allows the comparison of the time required for evolution instead of the less meaningful number of objectives evaluated presented in Chapter 4. A total of six experiments was performed. These six experiments provide quantitative data for five comparisons to be made for the effectiveness of dynamic objective thresholding, HCD objective ordering, auto objective ordering, generation synchronization, and HCD scaling. The conclusions reached during this chapter are then given in Section 5.3.

5.1 Problem Description

The overall objective of this optimization is to design a flare pattern that can be dispensed from an aircraft that can defeat an imaging seeker missile from multiple aspect angles and ranges. The constraints on dispensing the flare pattern define the search space. Flare patterns are evaluated with the GTRI-developed Imaging Seeker And Missile Simulation, ISAMS. The objective is to move the aircraft as far as possible from the center of the field of view of the missile. The region of interest is defined as moving the aircraft outside the field of view for all aspects and angles. The following subsections discuss each of the aspects in detail.

5.1.1 Search Space

The search space is defined by the characteristics of the flares ejected. A flare pattern contains up to twenty flares. Each flare has three associated attributes. First, multiple flare types are possible, each with different burn profiles, resulting in differing intensity and radial profiles. For this experiment, eight different flare types were created. The intensity

and radial profiles do not model any existing flares to remove dependence on potentially classified information. Second, the flare is ejected from a dispenser on the host aircraft. For this experiment, 16 ejector locations were specified for the host aircraft. The third and final attribute controlling the spatial dispersion of the flares is the time of ejection of the flare. For this experiment, flares can be ejected from 0.0 to 4.095 seconds after the pattern is initiated, with a minimum resolution of one millisecond.

Three attributes for each of the 20 flares define a 60-dimensional search space. Each flare has

$$8 * 16 * 4,096 = 2^{3+4+12} = 2^{19} = 524,288 \tag{129}$$

possibilities, resulting in

$$2^{19*20} = 2^{380} = 2.46 * 10^{114} \tag{130}$$

possible flare patterns.

5.1.2 Function Evaluator

The function evaluator for flare pattern evaluation is the ISAMS model [21]. This simulation was developed by GTRI under funding from the Naval Research Labs and Air Force Wright Laboratories. The simulation provides a model of several possible missile airframes. The airframe is controlled using the output of a model of several possible imaging tracker models. The imaging tracker model is presented with imagery of the view of the background, target aircraft, and any associated countermeasures, e.g., flare patterns.

For this experiment the ISAMS1 missile frame was utilized. The target aircraft for the experiment was the A-10 aircraft flying straight and level at a speed of 100 meters per second at an altitude of 1000 meters.

The background utilized is a clear sky condition. Because there are no other sources of false signals, this background is considered to be the most difficult condition for countermeasure. The sensor for experiments is a 256 by 256 focal plane array sensor with a 15 Hz update rate. The sensor has a 2.0 by 2.0 degree field of view.

The tracking algorithm has a track gate G with static size of 128 by 128 pixels centered on the previous track position. The background level B for the algorithm is determined by taking the average value of the pixels on a single pixel wide band around the track gate G. The track position is determined using an area-weighted centroid tracking algorithm. The track point is the centroid of the pixels in the track gate larger than the background level. Let $I(i, j)$ be a pixel of the resulting image, where i and j are the horizontal and vertical positions of the pixels respectively. The horizontal track point is

$$T_h = \frac{\sum_{(i,j)\in G \wedge I(i,j)>B} i}{\sum_{(i,j)\in G \wedge I(i,j)>B} 1} \tag{131}$$

and the vertical track point is

$$T_v = \frac{\sum_{(i,j)\in G \wedge I(i,j)>B} j}{\sum_{(i,j)\in G \wedge I(i,j)>B} 1} \tag{132}$$

This experiment provides an example of the importance of the selection of a proper objective value. Although the simple objective is to cause a miss, an objective value that is continuous is required to allow comparisons of individuals' performance before break-lock occurs. Therefore, the measure uses the angle of the aircraft from the center of the field of view. But, selection of a single value for this angle from the center of the field of view is not representative during early stages of evolution. For example, a flare pattern may cause a disturbance in the track point, but the aircraft is re-acquired by the end of the simulation. Likewise, it is anticipated that a loss of track at the end of the simulation is more important than a loss of track at the beginning of the simulation. Therefore, the final objective measure for an individual i is:

$$f(i) = \sum_{t=0}^{end} t * E(t) \tag{133}$$

where t is the simulation time in 10 millisecond steps, and $E(t)$ is the error signal, or the angle of the aircraft from the center of the field of view.

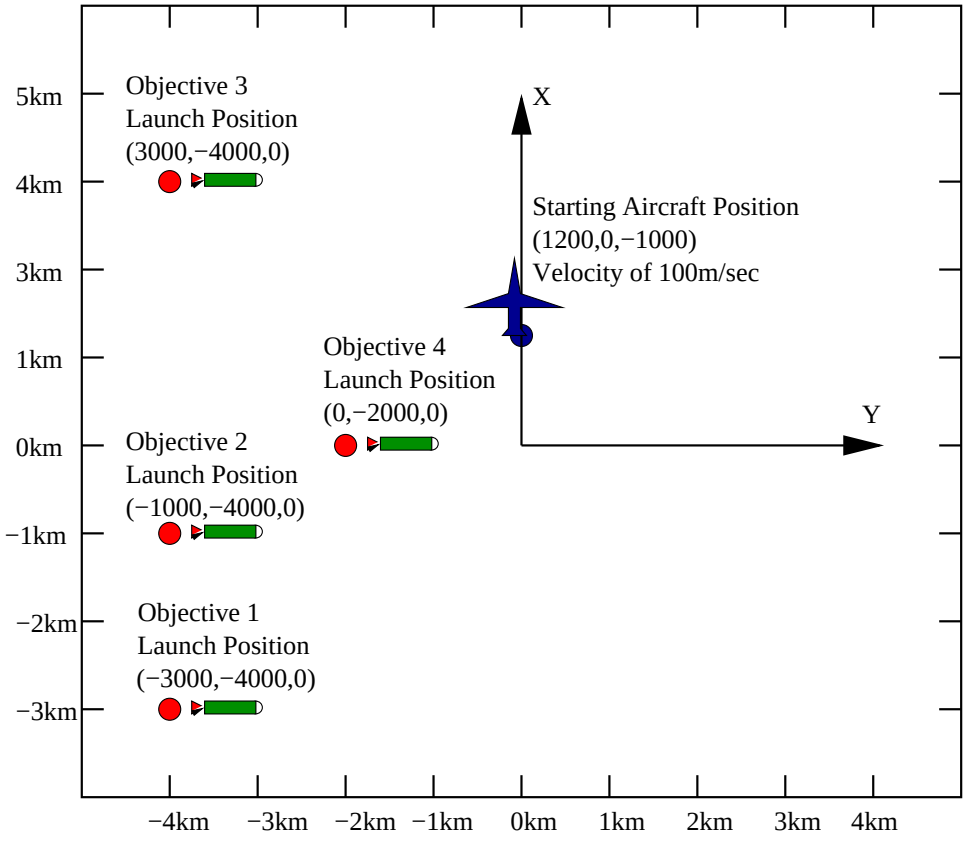

Figure 101: Launch Positions for the Four Objectives

5.1.3 Objective Space

Each objective is a separate simulation of a missile launch from a different location. Four launch locations are selected, as illustrated in Figure 101. These launch points result in viewing the aircraft from multiple aspect angles and multiple ranges.

Differences in launch points result in different amounts of simulation times required to reach the target. These different amounts of simulation time then result in different quantities of run-time to complete the simulation. The run-times are further affected by the individual being evaluated. When an aircraft leaves the field of view of the imaging tracker, the simulation time is reduced because the simulation is no longer required to render the aircraft. Table 14 displays example run-times required for each of the objective evaluations without a flare pattern, with a random flare pattern, and with an optimized flare pattern.

Table 14: Positions of Objectives, and Comparisons of Simulations With Various Types of Flare Patterns.

Objective	Position (Meters)	Simulation Time (sec.)	No Pattern Run-time (sec.)	Random Pattern Run-time (sec.)	Optimized Pattern Run-time (sec.)
1	(-3000,-4000,0)	25.20	348	344	193
2	(-1000,-4000,0)	11.87	164	161	154
3	(3000,-4000,0)	8.45	119	120	106
4	(0,-2000,0)	5.96	88	87	78

Figure 102 displays the results of an example output from the ISAMS simulation for each of the four objectives. Note that the aspect angle of the aircraft becomes apparent. This figure was created with a randomly distributed flare pattern, which does not cause break-locks in any of the scenarios.

5.1.4 Experiment Design

A series of six experiments were executed, where each experiment contained 10 trials. Each of the 10 trials contained a different random number resulting in different initial individuals as well as different objective orderings. The six experiments were:

1. *No Objective Thresholding.* An experiment with no dynamic objective ordering and no dynamic objective thresholding serves as a basis for the run with dynamic objective thresholding.

2. *No Objective Ordering.* An experiment with dynamic objective thresholding and no dynamic objective ordering was compared to the previous experiment for the measurement of the effectiveness of dynamic objective thresholding

3. *HCD Objective Ordering.* An experiment with HCD objective ordering was compared to the previous experiment for measurement of the effectiveness of HCD objective ordering.

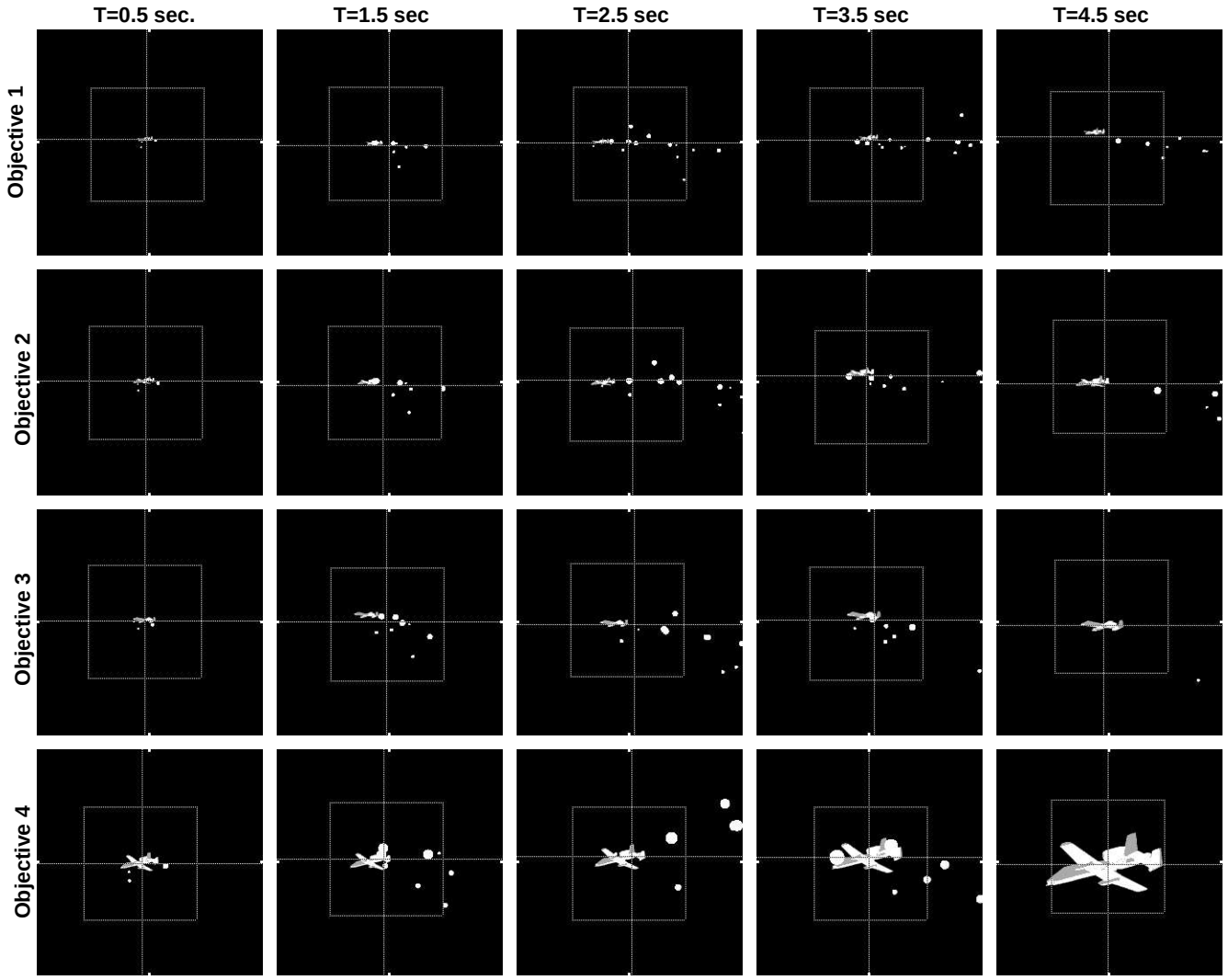

Figure 102: Random Flare Pattern Results for All Four Objectives

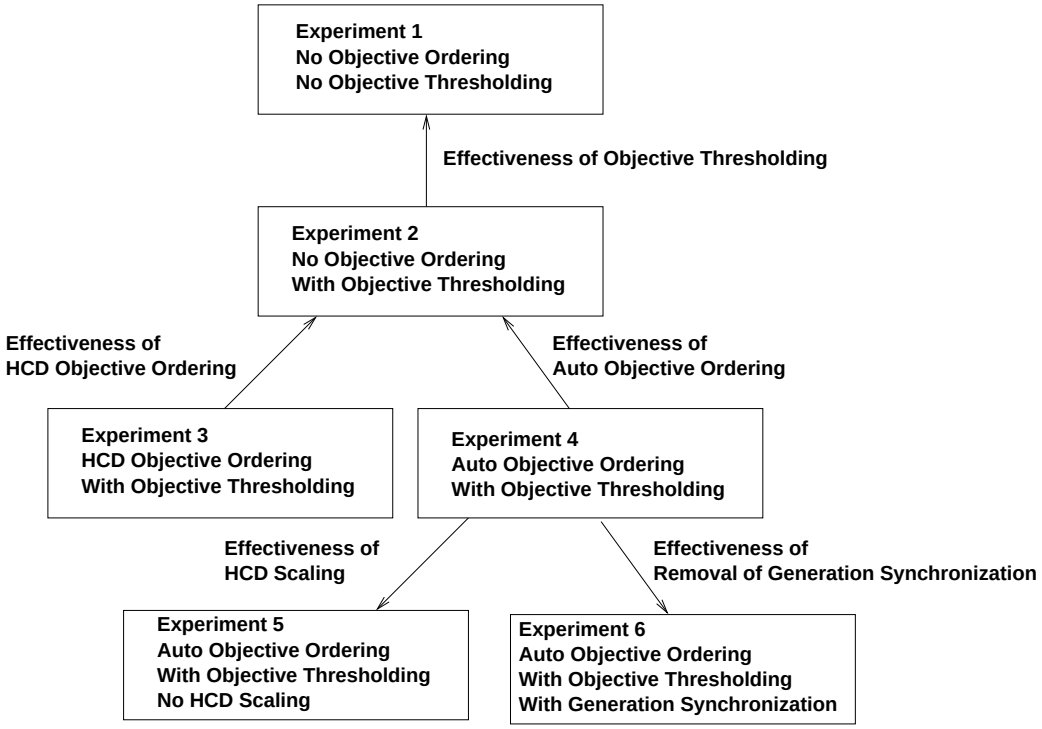

Figure 103: Relationships of Flare Pattern Experiments

4. *Auto Objective Ordering.* An experiment with auto objective ordering was compared to experiment two for measure of the effectiveness of auto objective ordering.

5. *No HCD Scaling.* An experiment with the HCD scaling disabled was compared to experiment four for measure of the effectiveness of HCD scaling.

6. *Generation Synchronization.* An experiment with the generation synchronization enabled was compared to experiment four for measure of the effectiveness of the removal of the need for generation synchronization.

These six experiments allow the five comparisons to be made for the effectiveness of dynamic objective thresholding, HCD objective ordering, auto objective ordering, generation synchronization, and HCD scaling. Figure 103 illustrates the relationships of the six experiments resulting in the five comparisons.

5.1.5 Multiple Objective Evolutionary Algorithm Setup

The fitness function utilizes the Pareto rank 1 "raw fitness" calculation followed by four fitness scalers. The first fitness scaler was the SPEA2 density scaler. The second fitness scaler was the Linear Interpolated Data Transformation 1 utilized in Section 4.1.2.1, which contained samples of the function

$$T_1(x) = \frac{1}{2^{(x-1)}}.$$

Table 7 itemizes and Figure 43 illustrates the data used for the LID transformations. The third fitness scaler was the fitness-sharing algorithm of the NPGA as described in Section 2.2.4.2. This fitness-sharing algorithm was initialized for 30 niches. The fourth and final fitness scaler was the HCD Scaler as detailed in Section 3.3. For the HCD scaler, the Holder coefficient was set to two for Euclidean distance, $p = 2$, and the offset was set one fifth, $\alpha = 0.2$. The HCD scaler was not used for experiment five in order to determine the effectiveness of runs with and without HCD scaling.

Selection was performed using fitness-proportionate selection. Crossover was performed using the clone operator for twenty percent of the individuals. The remaining eighty percent of the individuals were created using single point crossover.

Mutation was performed using bitwise mutation with a probability of mutating each bit of 0.0025. From equation 130, there is a total of 380 bits required to encode the genome, resulting in a probability of mutation for an individual of

$$P_m = 100\left(1.0 - (1.0 - 0.0025)^{380}\right) = 61.4\% \tag{134}$$

Dynamic objective threshold was performed with lower limits of

$$\boldsymbol{S} = \{5000, 5000, 5000, 5000\}, \tag{135}$$

and the upper limits, defining the region of interest, of

$$\boldsymbol{L} = \{20000, 20000, 20000, 20000\}. \tag{136}$$

The weights for each objective are

$$\boldsymbol{W} = \{0.75, 0.75, 0.75, 0.75\}. \tag{137}$$

The value of 5000 was found via experimentation to be obtainable by some random individuals. Likewise, the value of 20000 results break-locks for the experiments. For experiment 1, dynamic objective thresholding was disabled by setting the weights to zero,

$$\boldsymbol{W} = \{0.0, 0.0, 0.0, 0.0\}. \tag{138}$$

Elitism was performed using an elite gene pool with a maximum of 400 individuals. Elitism was found by removing genome with duplicate objective values, removing genome with Pareto rank greater than five, and then pruning the individuals with the smallest fitness value until the elite pool contained only 400 individuals.

Each run was initialized with 500 random individuals. Each generation contained 20 individuals. Generation synchronization was disabled for all but experiment six. Ten 1,800 MHz Athlon processing nodes were dedicated to the optimization.

Objective ordering was utilized for all cases. The initial order of objectives was set randomly to remove any bias from the initial objective ordering. A sample of 10 runs, each with a different random number seed, was performed to sample the various ordering possibilities. The number of individuals in the historic pool $|H_O|$ was set to 500. For most experiments, the number of individuals evaluated between ordering updates D_O was set to 200. For experiments one and two the objective order was disabled by setting D_O to a large number.

5.2 Results

Table 15 presents the results for each of the 10 runs for each of the six experiments. The times given are wall clock times required for the optimizations to reach the region of interest defined by $\boldsymbol{L}$. A total of 30.9 calendar days was required to complete the optimizations,

reflecting a total of 309 central processing unit (CPU) days. The following subsections detail each of the five comparisons illustrated in Figure 103 made for the six experiments.

Figure 104 displays the results of an optimized flare pattern. Note the loss of the aircraft from the field of view by 4.5 seconds for all four objectives.

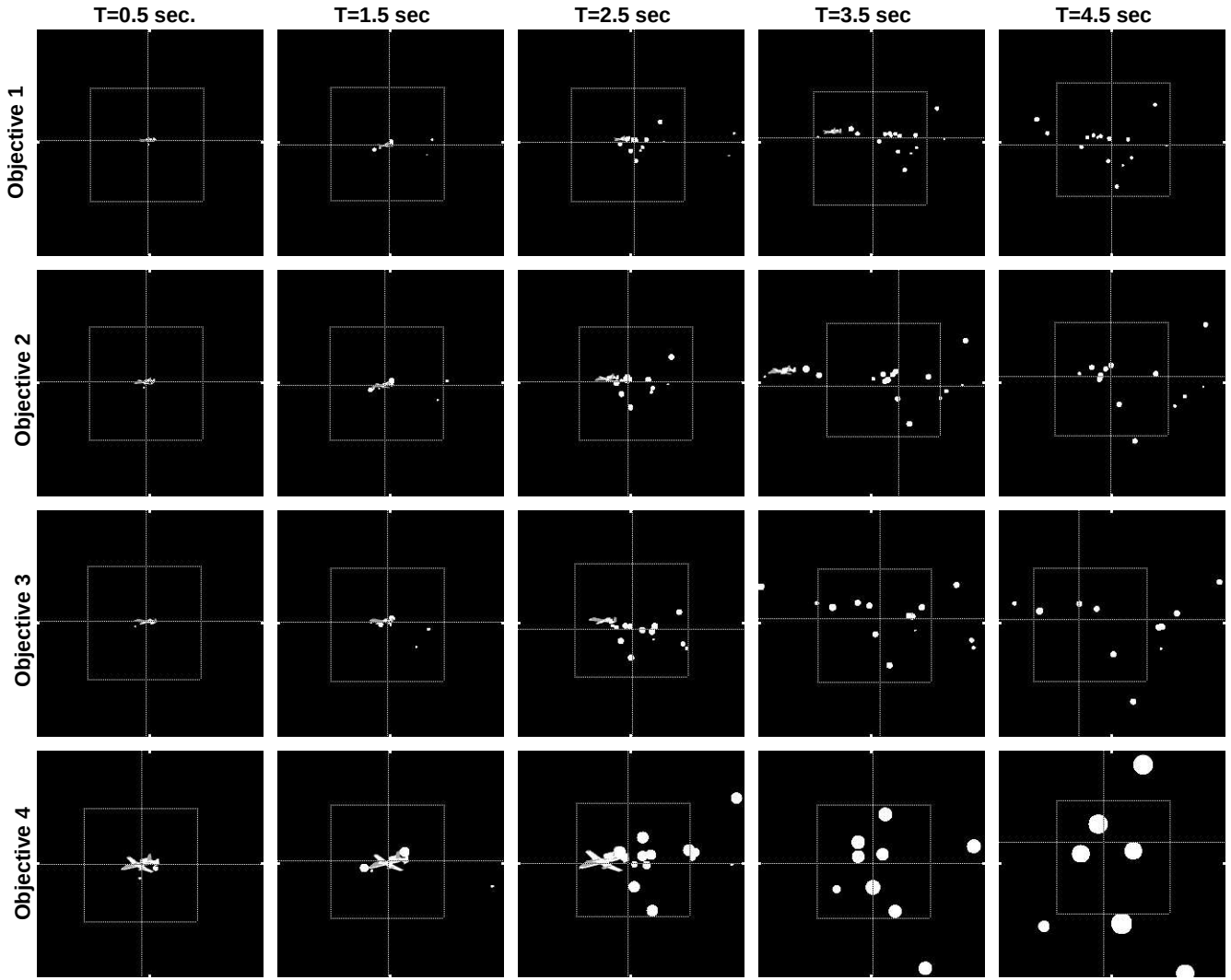

Figure 104: Optimized Flare Pattern Results for All Four Objectives

Table 15: Results of Flare Pattern Design Experiments

Run #	Experiment 1 No Dynamic Threshold			Experiment 2 No Objective Ordering			Experiment 3 HCD Objective		
	Time (min)	# Obj.	# Ind.	Time (min)	# Obj.	# Ind.	Time (min)	# Obj.	# Ind.
1	1,375	4,861	1,211	1,046	4,933	1,832	1,674	6,492	2,332
2	1,097	3,894	964	427	2,063	1,123	335	1,445	886
3	1,319	5,015	1,253	1,038	3,195	1,352	921	3,017	1,257
4	1,275	4,621	1,149	823	3,721	1,675	562	2,430	1,257
5	1,110	3,888	971	516	2,048	1,109	635	2,897	1,909
6	887	3,093	733	655	1,651	872	944	3,153	1,269
7	874	3,016	749	224	1,071	678	243	1,127	688
8	1,216	4,376	1,094	336	1,330	768	573	2,291	1,468
9	771	2,531	630	387	864	590	729	2,054	953
10	696	2,209	550	792	2,189	1,010	1,582	5,357	2,094
Total	10,620	37,504	9,304	6,244	23,065	11,009	8,198	30,263	14,113
Avg/Min	N/A	3.53	0.88	N/A	3.69	1.76	N/A	3.69	1.72

Run #	Experiment 4 Auto Objective Ordering			Experiment 5 No HCD Scaling			Experiment 6 With Generation Synchronization		
	Time (min)	# Obj.	# Ind.	Time (min)	# Obj.	# Ind.	Time (min)	# Obj.	# Ind.
1	693	2,968	1,288	1,283	5,587	2,070	568	1,916	960
2	310	1,569	992	288	1,405	1,006	193	895	658
3	601	1,792	890	1,795	5,772	2,379	1,422	3,978	1,038
4	660	2,837	1,266	630	2,360	1,191	444	1,655	1,661
5	913	4,177	2,118	939	3,890	1,750	1,102	3,862	895
6	567	1,597	912	1,155	3,855	1,647	919	2,266	1,116
7	157	904	672	169	950	688	304	1,177	739
8	424	2,034	1,184	299	1,225	873	410	1,290	778
9	357	731	547	357	737	550	370	738	558
10	586	1,633	890	806	2,484	1,253	781	1,927	958
Total	5,268	20,242	10,759	7,721	28,265	13,407	6,513	19,704	9,361
Avg/Min	N/A	3.84	2.04	N/A	3.66	1.74	N/A	3.02	1.44

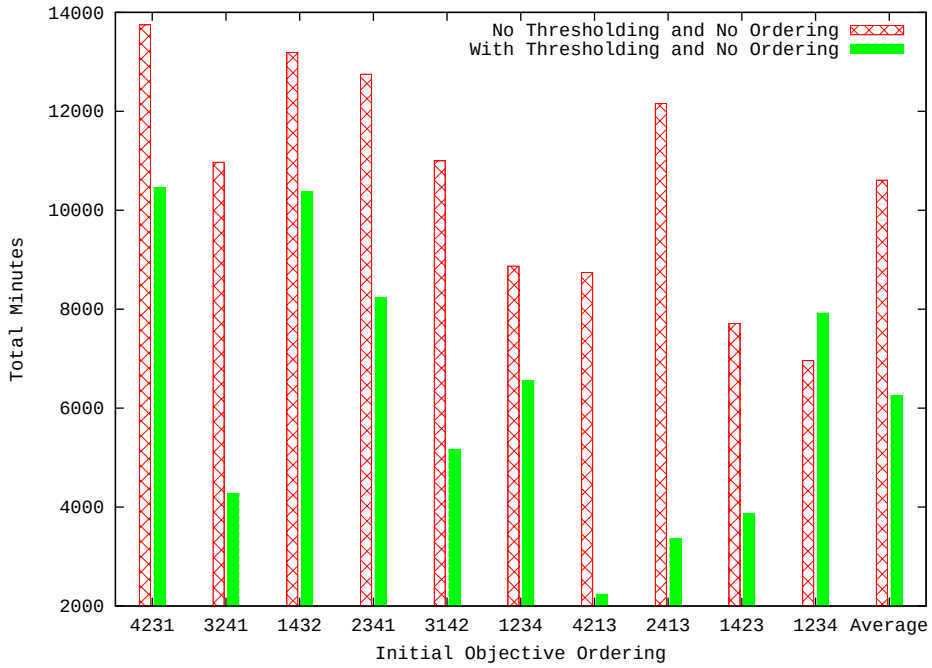

Figure 105: Comparison of Dynamic Objective Thresholding Results

5.2.1 Dynamic Objective Thresholding Comparison

Dynamic objective thresholding is used to terminate the evaluation of an individual before all objectives are evaluated if it is determined on the basis of the evaluated objectives that the individual is not likely to improve performance. It is only by not evaluating all objectives for every individual that speed improvements can be made. Details of the dynamic objective thresholding method can be found in 3.4.1.

Figure 105 illustrates a comparison of the results of experiment one and two from Table 15. Dynamic objective thresholding runs require, on an average, only 58.8 percent of the time required for those runs made without dynamic objective thresholding. In contrast to the 0/1 Knapsack runs of Section 4.1.3, these significant improvements were made despite the low number of individuals required for each evolutionary epoch.

A larger run-time was required with dynamic objective thresholding for only one out of 10 of the evolutionary runs. Note that this larger time was received on the evolutionary run that required the least time and fewest individuals to find a solution without dynamic

objective thresholding. This implies that dynamic objective thresholding should further improve run-time for problems that require more individuals.

Dynamic objective thresholding does not decrease the number of individuals required. In fact, an 18.3 percent increase in the number of individuals, from $9,304$ to $11,004$, is required. But, the number of objectives decreased by 38.5 percent, from $37,504$ to $23,065$, for runs using dynamic objective thresholding. This decrease in the number of objectives reduced the average time for an individual from 11.4 minutes for runs without dynamic objective thresholding to 5.68 minutes per individual for runs with dynamic objective thresholding.

5.2.2 Hypercube Distance Ordering Comparison

Each time the HCD ordering method is called, the average objective distance to the region of interest is calculated for those individuals with solved objective values in each objective dimension. The objective with the largest average distance to the region of interest is then placed at the front of the objective list. Likewise, the remaining objectives are ordered to contain successively decreasing average distances to the region of interest. Further technical details of the HCD ordering method can be found in Section 3.4.2.1.

Figure 106 displays the time required for evolution for each of the 10 runs for experiments two, three and four. Although this section is only concerned with the hypercube distance ordering methods, the figure supports the analysis of the hypercube distance ordering and auto ordering methods. The figure and Table 15 illustrate that the HCD ordering method only outperforms the runs without dynamic objective ordering for three out of 10 trials and on average required 31.3 percent longer to run. Interestingly, the number of objectives individuals evaluated per minute is very close to that required for those without objective ordering. The HCD ordering method simply requires 28.2 percent more individuals to be evaluated.

Figure 107 displays the transitions in objective order made for each of the 10 evolutionary trials. Analysis of this figure displays the propensity of objective one to evolve to be executed first or second, regardless of the initial random objective order. Table 16 itemizes the percent of times each objectives were evaluated in each of the four possible objective orderings during the final two objective orders. The final two objective orders were selected to weight each run equally, and to remove the initial transition effects during the first orderings.

From this table, note that the first two objectives were selected first 95 percent of the time. This implies that these objectives were likely the most difficult objective; i.e. resided furthest from the region of interest most of the time. Also note that if objective one is selected first, objective two is rarely selected second. The converse is also true. This

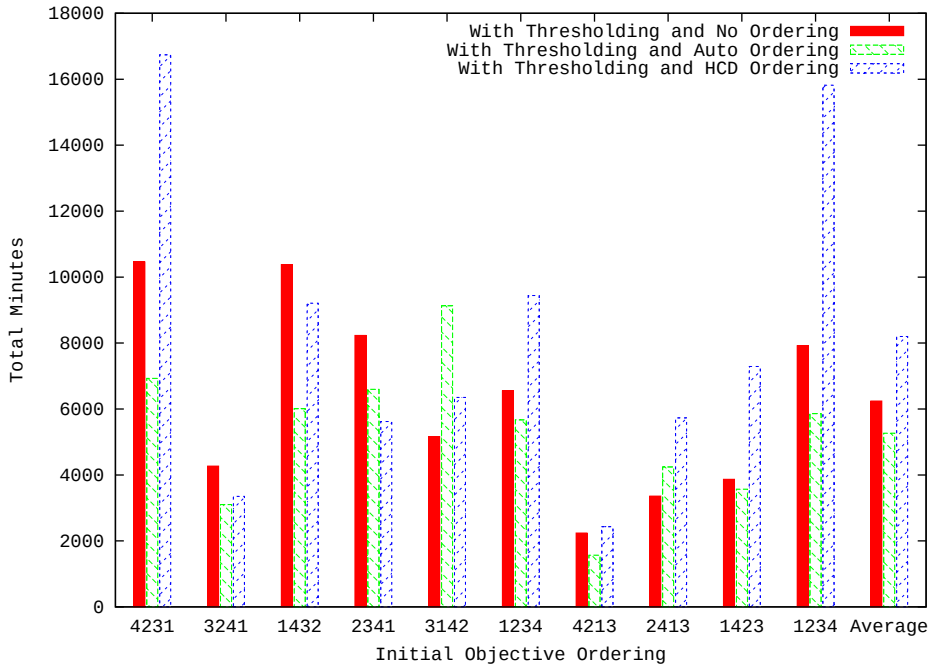

Figure 106: Comparison of Dynamic Objective Ordering Results

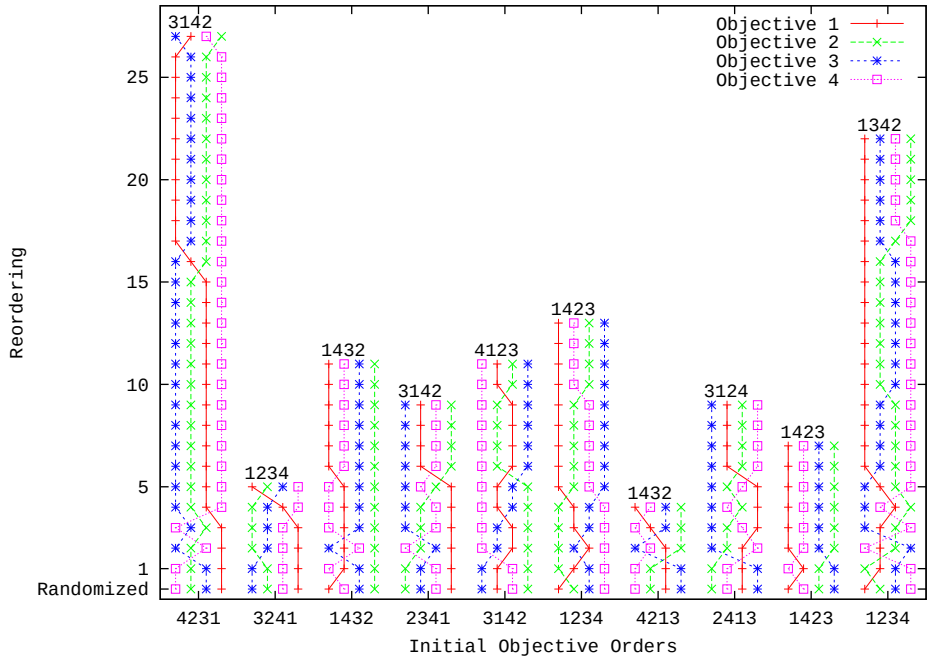

Figure 107: Order of Objectives for Hypercube Distance Objective Ordering Method

Table 16: Percent of Final Two Orderings For Each Objective For Ten Trials That Each Objective Was In A Particular Objective Order for the HCD Objective Ordering Method

Objective	**Percent**			
	First	Second	Third	Fourth
1	55	5	25	15
2	40	5	20	35
3	5	35	35	25
4	0	55	20	25

implies that objectives one and two, although both difficult, are orthogonal in objective space.

Contrary to the conjectures in Section 3.4.2.2, even though the slow running objective number one was selected to be executed first 55 percent of the time, it was actually the increased number of individuals that resulted in the reduced performance and not the increase in run-time for each individuals.

5.2.3 Auto Objective Ordering Comparison

The auto objective ordering method attempts to order objectives not just based on the objective values, but also based on the time required to evaluate each objective. Individuals maintained in the historic population H_O are "replayed" to determine the objective order that minimizes the time required to evaluate the historic population and still result in a population of individuals where the Pareto optimal solutions are similar to the Pareto optimal solutions of the current objective ordering. Relative to the HCD ordering method, the auto ordering method is much more complex. Section 3.4.2.2 further details the algorithmic operation of the auto objective ordering method.

Table 15 and Figure 106 give the results for the comparison of the auto objective ordering method of experiment four with the dynamic objective thresholding only method of experiment two. These sources illustrate that the auto objective ordering method required 18.5 percent less time to evaluate. The auto ordering methods helped the speed of evolution in two ways. First, the time required to evaluate individuals was reduced by 13.8 percent from 5.67 to 4.89 minutes per individual. Second, the number of individuals was reduced slightly by 2.3 percent to $10,759$.

Figure 108 displays the transitions in objective order made for each of the ten evolutionary trials. Table 17 itemizes the percent of times each objective was evaluated in each of the four possible objective orderings during the final two objective orders. From this table, note that 80 percent of these objective orders begin with the third and fourth objective. Recall from Table 16, that 95 percent of the objective ordering began with objectives one and two for the HCD objective ordering method. These differences are caused by the fundamental differences in the objective ordering methods. The HCD objective orderings were selected based on their relative difficulty in reaching the region of interest, thus the selection of objectives one and two. In contrast, the auto objective ordering method ordered objectives based on the time required to evaluate each objective. Table 14 showed that objectives three and four were the fastest running objectives. These two objectives

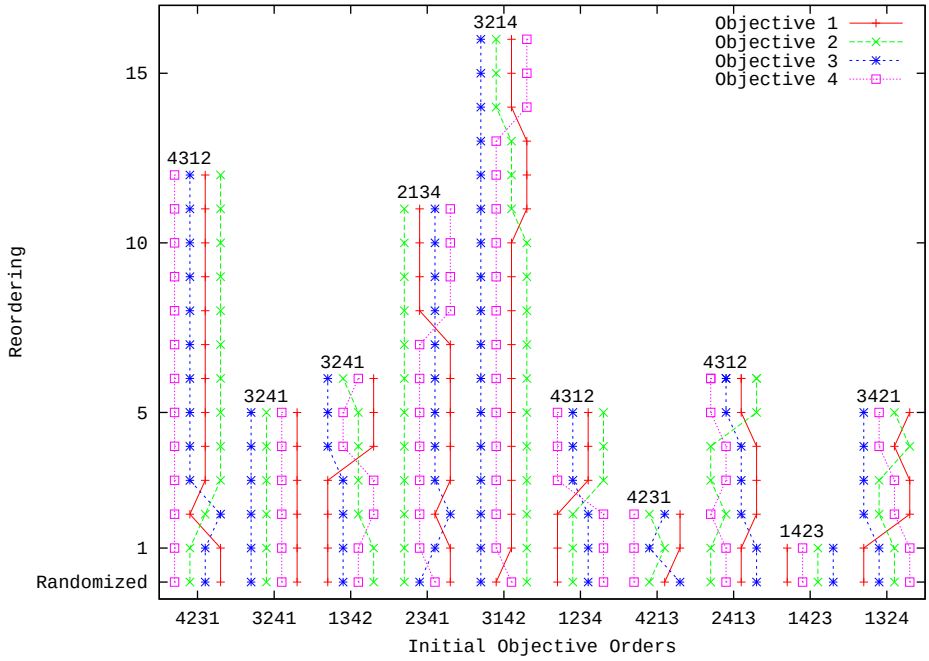

Figure 108: Order of Objectives for Auto Objective Ordering Method

were selected to be executed first for 80 of percent of the orders.

Table 17: Percent of Final Two Objectives Orders That Each Objective Was In A Particular Objective Order for the Auto Objective Ordering Method

Objective	Percent Evaluated			
	First	Second	Third	Fourth
1	10	10	40	40
2	10	30	35	25
3	45	25	15	15
4	35	35	10	20

5.2.4 Hypercube Distance Fitness Scaling

The HCD scaling algorithm utilizes the distance to the hypercube to scale fitness values. The basic premise of the HCD scaling method is that those individuals that reside closest to the region of interest are most likely to produce children that reside yet closer to the region of interest. Since the region of interest is defined by a hypercube in objective space, the closer the individual is to the hypercube the higher the probability of selection. Section 3.3

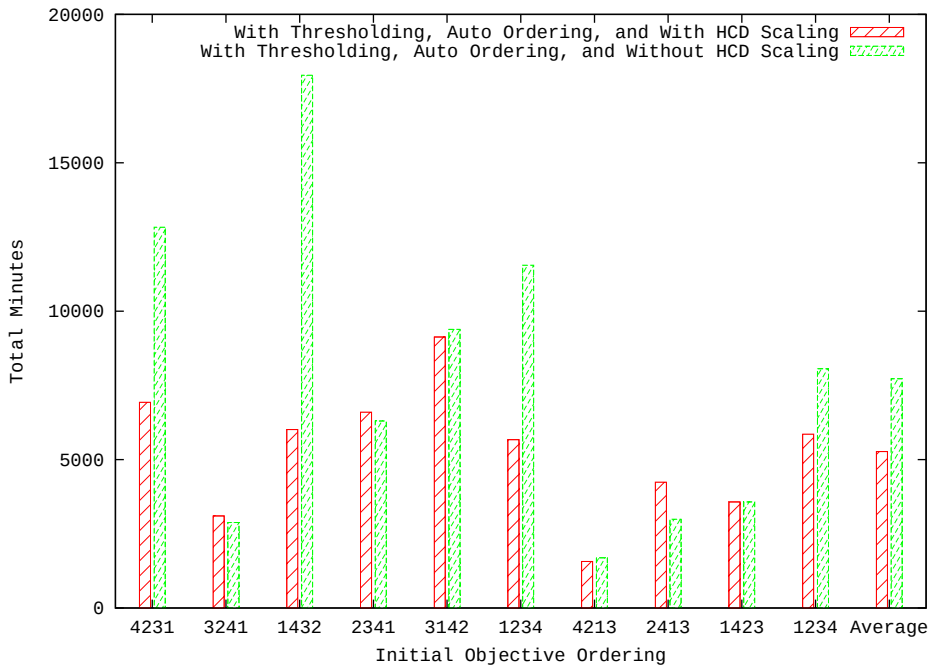

Figure 109: Comparison of Results With and Without Hypercube Distance Scaling

provides a detailed description of this algorithm.

Table 15 and Figure 109 support comparison of the auto objective ordering method with and without HCD distance fitness scaling. Using experiment five, the experiment without HCD scaling, as the basis for our comparison to experiment number four, the experiment with HCD scaling, supports the following discussions in terms of the improvements added by HCD scaling.

Runs with HCD distance scaling reduced the run-time by 31.8 percent from $7,721$ minutes to $5,268$ minutes. From 109, note that six out of ten of the runs with HCD scaling only performed slightly better or slightly worse than the runs without HCD scaling. But, the remaining four runs performed much better, resulting in a greatly improved average performance and reduction in the standard deviation from 480 to 218 minutes. Note that these four runs contained some of the largest run-times either with or without HCD scaling. Therefore, HCD scaling most benefits conditions where the initial objective ordering is poorest, e.g., 4321, 1432 and 1234. Similar reductions in the standard deviation were found

for the 0/1 Knapsack problem illustrated in Figure 66 and discussed in Section 4.1.3.2.

The improved performance was a result of two factors. First, the total number of individuals decreased by 19.8 percent from a total of 13,407 to 10,759 individuals. Second, the average time per individual decreased by 15.1 percent from 5.76 to 4.89 minutes per individual. These two decreases imply there were two sources of improvements. The reduction in the number of individuals evaluated was expected by the premise of the HCD scaling algorithm, which is that by giving individuals nearer the region of interest higher probability of mating the region of interest will be reached faster.

The reduction in the average time per individual is more difficult to explain. These reductions must be a result of the dynamic objective thresholding, which is in turn coupled to the dynamic objective ordering method. Therefore, by focusing selection on individuals near the region of interest, the HCD scaling method must provide the ordering method with a more appropriate ordering for those individuals near the region of interest.

5.2.5 Synchronization Effects

Generation synchronization times occur at the end of each generation when processors must wait for the evaluation of other individuals being evaluated on other processors before the next generation of individuals is available for evaluation. These synchronization times become most detrimental for this particular class of problems where:

1. The computation time for objective evaluations is large in comparison to the time required to perform the EA operations.

2. The evaluation time may be dependent on the genome being evaluated.

3. Disparate numbers of objectives may be evaluated because of dynamic objective thresholding.

The only other possible source of synchronization times is that processors may have various processing speeds due to their processing speed and loading. This source of synchronization problems was purposefully obviated by using dedicated, identical processing nodes to allow for meaningful comparisons.

To overcome the synchronization problems, this research contribution has designed and implemented an approach that obviates the need for synchronization. To implement this approach, each processor is given an individual to evaluate. Remaining individuals in the current generation, C, are given to a processor for evaluation as they finish the evaluation of their current individual. When there are no more individuals in the current generation that need evaluating, or are not in the process of being evaluated, then the software creates the next generation based on the current population and the elite population. Further details of this method can be found in Section 3.2.2.

Table 15 and Figure 110 support comparison of the generation synchronization method. Using experiment six, the experiment with generation synchronization, as the basis for the comparison to experiment number four, the experiment with generation synchronization removed, supports the following discussions in terms of the improvements made by removing generation synchronization.

Runs without generation synchronization reduced the run-time by 19.2 percent from $6,513$ minutes to $5,268$ minutes. From Figure 110, note that four out of ten of the runs with generation synchronization performed better than those without generation synchronization. But, the remaining six runs performed better without generation synchronization, with the largest three run-times all being dramatically reduced. Overall this compression of run-times resulted in a reduction of the standard deviation from 394 to 218 minutes.

The total number of individuals increased by 14.9 percent from $9,361$ to $10,759$. The increase in the number of individuals is due to the decrease in the quality of individuals available at the time of selection. Reviewing the setup, there were 10 processors, $n = 10$, 20 individuals per generation, $|C| = 20$, and a maximum of 400 individuals in the elite pool,

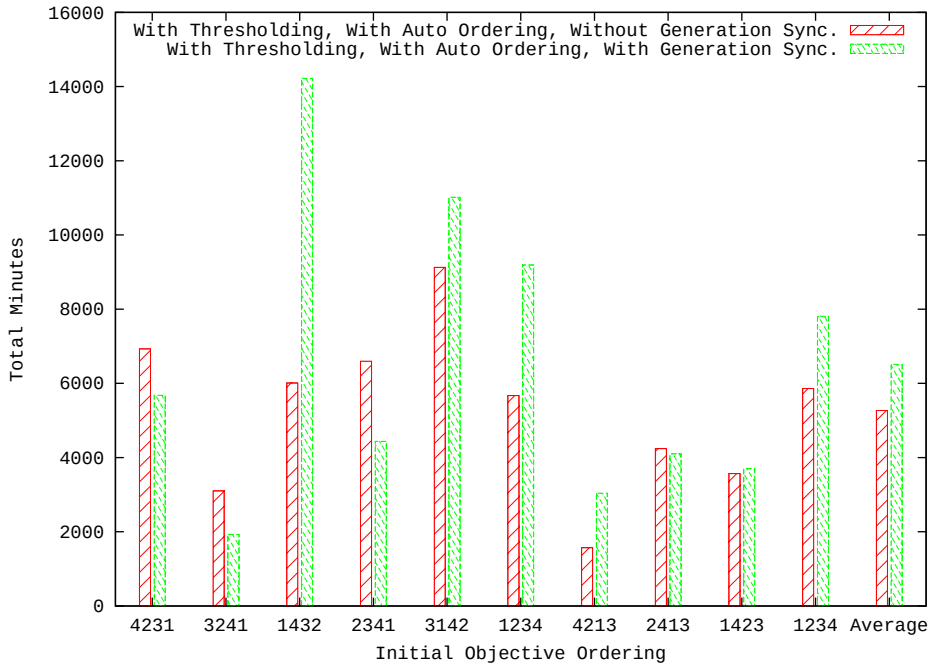

Figure 110: Comparison of Results With and Without Generation Synchronization

$|E| = 400$. At the time of creating a new generation the population of individuals used in selection for runs with generation synchronization is:

$$P_S = E \cup C \tag{139}$$

Those runs without generation synchronization will have $|C| - n = 20 - 10 = 10$ individuals from the present current population completely solved, and the previously unsolved ten individuals from the previous generation. Therefore, the population of individuals for the selection without generation synchronization does not have all of the information from the current generation that is available with generation synchronization. In addition, especially during the early phases of evolution, the current generation contains the best individuals, i.e., individuals that dominate the current elite individuals.

Despite the increased number of individuals evaluated, the removal of generation synchronization still resulted in improved performance because processors are not idle. The removal of idle time decreased the average time required per individual by 15.1 percent from $T_{GS} = 6.95$ to $T_{NGS} = 4.89$ minutes per individual.

This example points out the tradeoffs that can be made with the size of the current population, $|C|$. As $|C|$ increases, the percent of the total time required to evaluate the entire generation spent wait for synchronization is reduced. But as the value of $|C|$ increases, the average time between when a good performing individual is evaluated and when it creates children is also increased, which is likely to also increase the number of children required to be evaluated.

5.3 Conclusions

Five innovations were evaluated for performance improvements: dynamic objective thresholding, HCD objective ordering, auto objective ordering, HCD fitness scaling, and removal of generation synchronization. *All innovations, except the HCD objective ordering method, were shown to be effective for flare pattern development.*

Dynamic objective thresholding is able to reduce the run-time by 41.2 *percent for the flare pattern design problem. When dynamic objective ordering is coupled with auto dynamic objective ordering, the run-time is reduced by* 50 *percent.*

The HCD ordering method was shown to be effective for four objective 0/1 Knapsack problems in Sections 4.1.3.4. Under the stressing conditions of the flare pattern design problem, the HCD ordering method not only selected the slowest running objective to be evaluated first, as anticipated could happen, but also increased the total number of objectives required. *Therefore, the HCD objective ordering method is only recommended for those problems with objectives that require nearly identical run-times, and that are also very similar in their difficulty.*

The HCD fitness scaling method reduced the run-time by 31.8 *percent.* The improved performance was a result of a decrease in number of individuals evaluated, and a decrease in the average time per individual. A decrease in the number of individuals evaluated was expected based on the premise of the HCD scaling routine that giving individuals nearer the region of interest a higher probability of mating improves the progress toward the region

of interest. The improvement in the average time per individual was not expected, and is argued to originate from focusing selection on individuals near the region of interest, which allows a more appropriate ordering for those individuals near the region of interest to be found by the auto objective ordering methods.

The removal of the need for generation synchronization reduced the run-time by 19.2 *percent.* This increase was strictly a result of a dramatic improvement in the average run-time of individuals by the removal of wait times. There was actually an increase in the number of individuals required for evolution due to reduced knowledge provided at the time of creating a new generation due to the evaluation of only half of the current generation at that time.

CHAPTER VI

AAR-44A OPERATIONAL FLIGHT PROGRAM OPTIMIZATION

The AAR-44A is a missile warning receiver utilized on United States Air Force Special Operation Forces aircraft. It operates by sensing energy over a range of frequencies in the infrared spectrum, and makes a determination of whether the energy is from a background source or a missile. If a source is determined to be a missile, then the missile warning receiver makes a declaration resulting in a counter-measure action by other systems of the aircraft.

This chapter summarizes the use of MOEAs for optimizing the performance of the AAR-44A Operational Flight Program (OFP). This effort was performed under contract number F09603-99-D-0207-0023, identified as Georgia Tech Research Institute, (GTRI), project number A-6706 with Warner Robins Air Logistics Center Warner Robins GA, WR-ALC/LNEXS. Unclassified and classified reports were created for this effort [51, 50].

The purpose of this research was to improve the performance of the AAR-44A OFP. With the aid of government technical input, four measures of performance were selected: probability of declaration, false alarm performance, time to intercept minimum, and time to intercept average. These objectives were further divided into threat groups and regions of performance to create a twenty-two dimensional objective space. By inspection of the source code, 156 constants within the OFP were identified as having the possibility of improving system performance. These constants, along with their associated limits, define a 156-dimensional search space.

To optimize the OFP performance against field collected false alarms data, "live-fire"

missile data, and simulated missile data, MOEAs were applied to find the set of Pareto optimal individuals in a twenty-two dimensional objective space. The objective space was then reduced back to a four dimensional objective space mentioned above using a weighted-sum. The results of this optimization showed dramatic improvements in all four dimensions.

During this study the following improvements and innovations were made:

1. The performance of the AAR-44A was dramatically improved, (see Section 6.3.2).
2. A method for automated extraction of performance measures, and meta-data for existing data sources were created, (see Sections 6.1.3, and 6.1.4).
3. The important constants for future optimization were identified, (see Section 6.4).
4. Tools for modification of source code to support dynamic setting of constant values were created, (see Section 6.1.3.2).
5. Techniques for creating genetically diverse initial population were derived, (see Section 6.2).

This chapter is divided into four subsections. Section 6.1 describes the infrastructure required for objective evaluation during optimization and evaluation, including descriptions of the OFP evaluator, search space, objective space, flight test data, live-fire data, and simulation data. Section 6.2 presents the purpose, setup and results of asexual reproduction, which were necessary to create a genotypicly and phenotypicly diverse population. Section 6.3 presents the setup and results of more traditional sexual reproduction. Section 6.4 presents the sensitivity analysis that was able to reduce the number of constants required to implement the optimization from 156 to 74. Section 6.5 presents the results of the evaluation of the Pareto optimal individuals from the sexual optimization. Section 6.6 presents the results of a specialized optimization for high-altitude cases only. Section 6.7 then provides the conclusions reached during this chapter.

6.1 Operational Flight Program Objective Evaluation

Before optimization, a high fidelity simulation of the system had to be improved, tested and verified. Field test data had to be tested and categorized. Simulation data was created to supplement the field test data where field test data was unavailable. The search space and objective space had to be selected. The function evaluator was modified to accept a description of an individual within the search space. Finally, the function evaluator was modified to automate the extraction of the objectives. The following sub-sections discuss many of these details.

6.1.1 Search Space

The search space for this problem was created by a visual inspection of the OFP. The OFP was written in C++, and thus constants were implemented in one of two ways; as a "#define" for a macro, or as the setting of constant during initialization. The source code contained very few "hard-coded" constants listed directly within the calculations. Even constants such as ninety percent were declared with a "#define" macro. The 156 constant names, data types, and valid ranges were placed in an XML file that met the syntax of the *GTMOEA* search space DTD found in Section 3.1.1.

6.1.2 Objective Space

For the AAR-44A, twenty-two objectives were identified before starting optimization: eighteen for threats and four for false alarm performance as described below.

6.1.2.1 Threat Objectives

As with most pattern recognition systems, the two basic performance measures are the true positive and false positive performance. For a missile warning receiver the true positive performance results in the probability of detection, P_d, defined as the number of valid declarations for threats that occurred divided by the total number of threat runs.

Missile warning receivers must not only make a declaration, they must also make the declaration while there is still enough time to perform a counter-measure reaction. Therefore, there is a trade-off between increasing the quantity of data analyzed by the algorithms and making a declaration fast enough to provide an effective counter-measure response. The time available for counter-measure is the time to intercept, TTI; i.e., the time between the declaration and the time that the missile will impact the aircraft if a counter-measure is not employed. To measure this performance, two objectives were selected. First, the time to intercept minimum, TTI_{Min}, is defined as the minimum time before intercept that a declaration is received the same set of true positive cases. Second, the time to intercept average, TTI_{Avg}, is defined as the average time before intercept that a declaration is received for a set of true positive cases. It is highly desirable to increase both of these measures.

Threat data were divided into six groups. First, the threats were divided into Tier 1, 2 and 3 classes, which allowed application of different weights of importance for different types of threats. Second, the launch ranges for the threats were divided into short and long. For each of these six threat groups, three objectives were defined, resulting in a total of eighteen threat objectives.

6.1.2.2 False Alarm Objectives

With missile warning receivers, a false positive results in a false alarm, FA, which in turn results in a highly undesirable and unnecessary counter-measure action. False alarm performance can be measured in several ways. For this effort, the desire was to minimize the number of false alarms. Because the optimization software tries to find maximum values for all objectives, the measure of performance was selected as the Negative of the False Alarm Count, *NFAC*.

After discussions with the government, four *NFAC* objectives were agreed upon by dividing the data into four separate categories. First, the data was divided into urban and rural based on the notations made during data collection. Second, the data was divided into

low and high altitudes based on the associated platform state extracted from the recorded navigation data.

6.1.3 Function Evaluator

The function evaluator for the determination of objective values was Version 6.1 of the AAR-44A OFP modified to support the following: execution on the Linux platform, insertion of input sensor and navigation data from data files, setting of constants from standard input, and automated extraction of performance measures. The following sub-sections detail the modifications of the simulation to accomplish these requirements.

6.1.3.1 Rehost of Simulation to Linux Platform

Under previous programs, GTRI had developed a simulation of the AAR-44A OFP that operated on a Windows platform. This capability was ported to the Linux platform so that a cluster of Linux machines could be used to perform the optimization. During this porting process, the starting OFP was substituted with version 6.1, and many problems in the sensor and source simulation were identified and corrected. The performance of the resulting simulation was compared to results of the hardware, resulting in differences that were minor and attributable to the differences in the processor and compiler [11].

6.1.3.2 Automated Setting of Constant Values

Given a valid version of the rehosted OFP, then the source code was modified to support modification of the constant values from an external source. To meet the requirements of the optimization executable, *GTMOEA*, the source code must allow setting of constants interactively via an XML description. This was accomplished by providing support for the acceptance of an individual's description via standard input. An individual's XML description, defined by the search space description, was standardized into a DTD that described the `AAR44Settings` XML node. This description was flexible enough to support the specification of as many or as few of the possible constants as desired. If a constant was

not specified, the previous value loaded into the OFP was utilized. This capability allowed quick pruning of the search space during the training, sensitivity analysis and evaluation processes. The acceptance of the `Default` tag via standard input was also added to support the return of all constants to their default values.

Modification of the OFP source code was accomplished with the development of a special purpose perl script. Using a list of constants, this perl script modified the OFP source to change the setting of *constant*s and *#define*s into variables. It also supplied the hooks to change these variables via the XML description. The perl script worked well with the current version 6.1 of the OFP. Requirements analysis, users manual and test reports were created for this task [14, 12, 13].

6.1.3.3 Automated Measure Extraction

The base AAR-44A OFP was responsible for making declarations. It was not aware of whether the declarations were valid. To support automated optimization, the simulation had to be modified to take the declarations of the OFP and compare them with the input data to determine if the declaration was a valid declaration or a false alarm and to update the measures of performance appropriately [28].

As will be described in Section 6.1.4, meta-data was added to all input sensor and navigation data to facilitate the calculation of the four basic objectives measures: P_d, TTI_{Min}, TTI_{Avg}, and *NFAC*. The modified simulation provided the following capabilities: reinitialization of the objective measures, the input of many different threat or false alarm data sets, and the report of the objective measures for these sets of data. All of these actions were performed via standard input and output providing the interface of the MOEA software with multiple instantiations of the AAR-44A simulation.

6.1.4 Training and Evaluation Data

The term training data is used to identify the data used during the optimization process. A separate, slightly larger, set of data was reserved for the evaluation of the resulting solutions after optimization was completed. Three types of data were utilized. First, data was available for a small number of live-fire missile shots. Second, these live-fire missile shots were supplemented with simulations of missile shots to provide a significant number of true positive events. Third, false positive data was available from data recorded during flights of the system around areas containing many potential false alarm sources.

All data was placed in a Pass-1000 ARC file format to support use by both the simulation and the hardware. For each ARC file, a meta-data description was developed to aid in the automated extraction of objective values. Specifically for threats, the time of launch and the time of closest approach were listed. Also for threats, an azimuth and elevation window were included. This allowed the automated objective solvers to only count declarations in the valid temporal and spatial windows as true declarations. Other declarations were considered false alarms. Likewise, if multiple declarations occurred for a single missile, then only the first declaration was utilized in the calculation of the objective measures.

Some periods of time for ARC files should be ignored. In particular, declarations can occur from live-fire data when the missile is past the aircraft. By ignoring this time period, this event was not classified as a declaration or a false alarm. Likewise, invalid declares can occur in the first few seconds of startup for the system. The warmup period is required because the OFP must receive valid navigation data, which occurs asynchronously to the sensor data. The warmup period was ignored in our simulations. The following sub-sections detail each of these sources of data.

6.1.4.1 Live-Fire Data

Four requirements were determined for live-fire shots:

1. Only live-fire, free flight data can be used. Track launches or static firings are not usable.

2. The missile launch and flight is such that the missile is representative of an actual threat launch.

3. Data from an AAR-44A system was recorded for the specific case.

4. Data was available at GTRI.

Criterion 1 limited the data set to live-fire launches at the White Sands Missile Range (WSMR) Aerial Cable Facility. Three WSMR data sets were available for consideration: WSMR 1998, WSMR 2000, and WSMR 2001. No usable data was found from WSMR 1998 because no shots met criteria three and four. Only twenty-eight missile shots were found from the remaining WSMR 2000 and WSMR 2001 data sets that met all of the criteria [49].

The live-fire missile shots were perhaps the weakest link in the training and evaluation of the OFP for two reasons. First, the quantity of live-fire data was not sufficient for statistically significant performance analysis of the probability of detection. Second, this data was used to create the signature data files that feed the simulation of threats for the AAR-44A. Therefore, the simulated data used to increase the statistical significance of true positive data also may not contain the proper statistical variations in signature.

The twenty-eight shots available were divided into two sets of fourteen shots for training and evaluation. All shots were also classified into six sub-categories based on the threat type (tier one, two or three), and the launch distance (short or long). In addition to providing the 1553 data and meta-data for optimization and evaluation, it was also necessary to provide recommendations for division of all selected data into training and evaluation groups. This division was done to ensure threats from each class existed in both the training and evaluation data sets [48].

6.1.4.2 Simulated Threat Data

Simulated missile shots were created to mimic shots at an aircraft in various operational scenarios. Five scenarios were chosen with government input on the requirements [38]:

1. Launch against an aircraft in straight and level flight at 500 feet.
2. Launch against an aircraft in orbit at 6,000 feet.
3. Launch against an aircraft in orbit at 10,000 feet.
4. Launch against an aircraft in takeoff.
5. Launch against an aircraft in landing.

Launches were made using the DISAMS missile models. All launches were then used to create ten different AAR-44A simulation inputs. This was done by varying the sensor initial azimuth angle by thirty-six degree increments. This was intended to capture the significant differences in sampling of the temporal signature that could be realized in operation. A total of 4,440 simulated missile shots were created [39].

These simulated shots were again classified into six sub-categories based on the threat type (tier one, two or three), and the launch distance (short or long). Three tenths of the data was used for training (1,332 shots), and seven tenths of the data was used for evaluation (3,108 shots). The division was made by selecting three of the ten possible initial azimuth angles for training and the remaining seven for evaluation.

6.1.4.3 False Alarm Data

All false alarm data was captured from flight tests, which occurred in late 1996, and early 1997. With the input and concurrence of the government, the data was divided into four categories based on the conditions of rural or urban, and high or low altitude. The conditions of rural or urban were noted during data collection. The altitude was determined from navigation data. There were approximately forty hours of false alarm data available [49].

The data was divided into two groups for training and evaluation. Efforts were made during this division to create nearly equal divisions in the four categories, and in the number of tracks within each data set [48].

Note that the data as delivered contained errors in the navigation data. The roll had been inverted. This data was post-processed by GTRI to correct this error. The corrected navigation data was used during optimization and evaluation.

6.1.4.4 Binary Formating

Before optimization began, the data was transformed from an ARC format to a binary data format. This was done by saving the data out of the simulation after it had been read into the data structures required by the simulation. Also, many messages not used by the OFP were removed.

This translation was performed for two reasons. First, converting to a binary data format slightly improved the speed of the simulation by placing the data in a format closer to that required by the simulation. Second, and more importantly, it reduced the size of the data files required for training by a factor of two. This was important because the entire 1 gigabyte of data required for training was placed on the local drive of every computer node used during the optimization process to drastically reduce the quantity of data served by the NFS file servers.

6.2 Asexual Reproduction

This section details the motivation, setup and results of asexual reproduction.

6.2.1 Motivation

Asexual reproduction was utilized in response to problems found when trying to start evolution with sexual reproduction. Sexual reproduction is typically initialized with a population that is uniformly distributed; i.e., each attribute of each genome is randomly selected to lie within the attribute boundaries. In many optimization problems this is a valid approach,

because each location in the search space maps to a non-trivial location in objective space. This is true with most research problems such as Deb's T functions as well as with other military application such as flare pattern design. Unfortunately, when a uniform distribution of values was used with AAR-44A OFP optimization problem, it resulted in a population in which 99.5 percent of the individuals created mapped to the same location in objective space. That location contained no declaration for any data; i.e., a system with zero false alarms, but also zero probability of detection.

The 0/1 Knapsack problem is able to use "genome correction" to correct fatal individuals. A similar approach was dismissed as too costly for the OFP optimization because it requires computationally expensive objective evaluations to correct the genome. Thus, to overcome these problems, a method was needed to create an initial non-trivial population. These problems and the solution represent an important innovation.

To create a population of non-trivial solutions, it was necessary to start with a solution that was known to be valid. The default settings of the OFP were just such a solution. Therefore, solutions were needed that contained attribute settings that were close to the default values. The ability to mutate variables with a normal distribution provided the ability to create such values. The default settings of the OFP became the starting population of one. This initial individual was then cloned and modified with a high probability mutation to produce more individuals. The following generations then utilized selection and mutation only to produce more children. Without crossover, this method of optimization becomes similar to the asexual reproduction of organisms.

6.2.2 Setup

Evolution then utilized two of the three evolutionary operators: selection, and mutation. Much of the setup was the result of the recommendations of Chapter 4. The asexual reproduction process was seeded with evaluation of the default instance. Each subsequent generation contained 200 individuals. Since this was a very computationally expensive

problem with a total of 30 minutes required for the full evaluation of an individual, a total of 70 processors with various loadings were utilized. The large evaluation times, and the disparity of processor loadings made use of the GTMOEA software *without generation synchronization* imperative (see Section 3.2.2).

Asexual optimization was performed in a four dimensional space of the following objective: the probability of declaration, P_d, the negative of false alarm count, *NFAC*, the time to intercept minimum, TTI_{Min}, and the time to intercept average, TTI_{Avg}.

The fitness function utilized the Pareto rank 1 "raw fitness" calculation followed by two fitness scalers. The first fitness scaler was the SPEA2 density scaler. The second fitness scaler was the Linear Interpolated Data Transformation 2 utilized in Chapter 4, which contained samples of the function

$$T_2(x) = \frac{1}{x}.$$

Table 7 itemizes and Figure 43 illustrates the data used for the LID transformations.

Fitness proportionate selection was used for the selection of the best individuals. Note that individuals were not allowed to mate, they were simply cloned for the mutation operator. Because asexual reproduction only creates individuals with mutation and does not have crossover, there is a non-zero probability that the individual created will be identical to the parent. To prevent the wasting of processor time for evaluation of identical individuals, children were compared to the parents to insure they were not identical. If they were identical, the child was removed.

As described in Section 3.1.3, the *GTMOEA* software supports the assignment of the type of mutation to any named attribute in the search space. Each attribute was assigned a normal distribution with a standard deviation equal to ten percent of the default value. Those attributes with values of zero were given a standard deviation of 0.001. Note that large deviations can be obtained by the accumulation of smaller changes over many generations. Each attribute was given an independent ten percent probability of mutation, resulting in a $(1.0 - (1.0 - 0.1)^{156}) * 100 = 99.999993\%$ probability of mutation for each

individual.

For the asexual optimization, the first objective evaluated was the probability of the declaration, P_d. The minimum value for this objective was set slightly lower than the measured performance of the default system. Any individual with less than this performance was not allowed to evaluate the remaining objectives. This harsh threshold provides termination of evaluation for individuals that do not have better P_d than the default system. The maximum value for the objective is 1.0, representing a perfect declaration probability. The maximum weight element was specified as 0.5, thereby failing fifty percent of candidate individuals.

The second objective was the time to intercept minimum, TTI_{Min}. The values of the minimum and maximum of this objective may be considered sensitive and are thus not presented here. The weight for the dynamic thresholds was specified as 0.0, allowing any individual with an objective value above the minimum to be further evaluated.

The third objective was the time to intercept average, TTI_{Avg}. The values for the minimum and maximum of this objective may be considered sensitive and are thus not presented here. A dynamic threshold between these two values was defined with the maximum weight of 0.5, thus failing fifty percent of candidate individuals.

The fourth objective was the negative false alarms count, *NFAC*. Because this was the final objective, the minimum value and maximum weight elements are not important for truncation of evaluation. But, the maximum value of zero does help define the region of interest. A value of zero was assigned indicating the desire to increase the *NFAC* to have no false alarms.

Dynamic objective ordering was not utilized during the optimization for several reasons. First, objectives were not independent. The objectives P_d, TTI_{Min} and TTI_{Avg} were all calculated simultaneously based on a set of threat data. Therefore, there was no advantage to separating these objectives. Second, knowledge of the problem domain allowed recognition that the threat data was able to run much faster than the false alarm data. Therefore, the threat data objectives were evaluated first. Then, the false alarm data was evaluated if

the threat data performed at an acceptable level.

A set of four rules were used to prune the size of the elite pool, E. The first rule provided genotypic diversity by ensuring that no individuals with identical genome values were included. The second rule provided phenotypic diversity by ensuring that no individuals with identical objectives values were used. The third rule ensured that the individuals were near Pareto optimal by removing individuals with a Pareto rank greater than five. The fourth and final rule ensured that only fifty individuals reside in the elite pool, $|E| \leq 50$. If over fifty individuals resided in the elite pool after the application of the previous rules, only the fifty individuals with the best fitness values were retained in the elite pool.

6.2.3 Results

Two runs of asexual reproduction were made. These two runs evaluated a total of $9,400$ individuals. The genetically diverse, Pareto optimal solutions from these runs were then used to seed the sexual reproduction as presented in Section 6.3.

Objective values were recalculated by presenting a "quality measure" for each objective. The quality for each objective is defined as:

$$Q = \frac{O_A - O_S}{O_M - O_S} * 100.0, \tag{140}$$

where O_A is the objective value achieved, O_S is the objective value from the default system, and O_M is the maximum possible objective value. For each objective, Q is the percent increase of the remaining objective gains that can be made. This removes the ability to determine the potentially classified value of the starting objective value, O_S.

For false alarms, O_M is zero. Therefore, the quality measure becomes,

$$Q = \frac{NFAC - NFAC_S}{-NFAC_S} * 100.0, \tag{141}$$

which is the percent reduction in false alarms.

For the probability of detection, the O_M is 1.0. Thus, the quality measure becomes

$$Q = \frac{P_{d,A} - P_{d,S}}{1 - P_{d,S}} * 100.0. \tag{142}$$

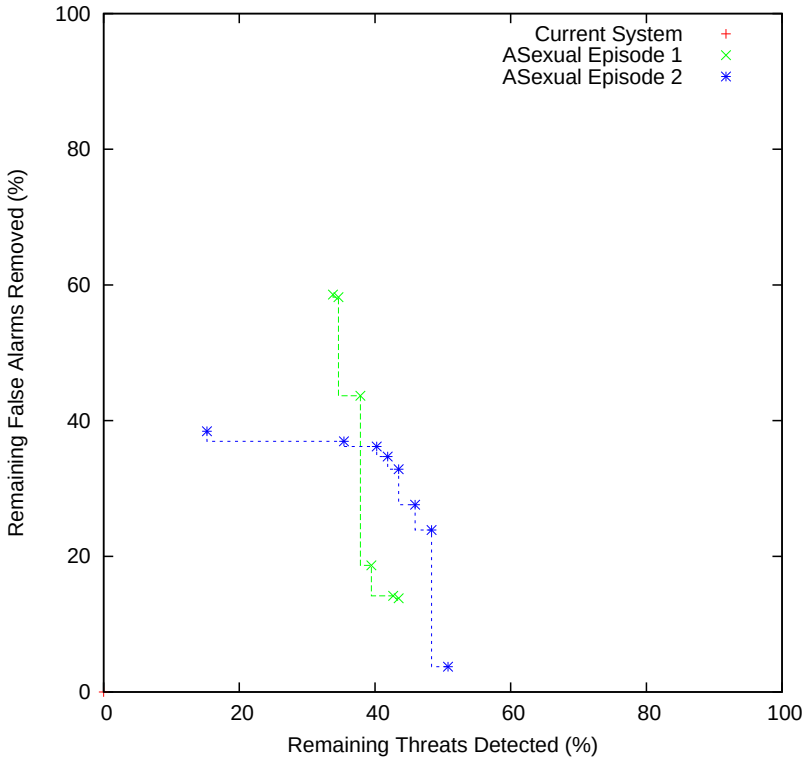

Figure 111: AAR-44A Asexual Reproduction Unweighted Training Results

For TTI_{Avg} and TTI_{Min} a value was chosen as the O_M was chosen as TTI_M. Thus, the quality measure becomes

$$Q = \frac{TTI_A - TTI_S}{TTI_M - TTI_S} * 100.0. \tag{143}$$

Figure 111 displays the quality measures for the P_d and *NFAC* objectives, which are Pareto optimal in these two dimensions. Note that with the evaluation of only 9,400 individuals, asexual reproduction was able to produce a sizeable improvement in the probability of detection and false alarm performance.

Table 18 itemizes the quality measures for the individuals that were Pareto optimal in the four dimensional objective space and were in the region that outperformed the default OFP in all dimensions. These Pareto optimal individuals were used to seed the sexual reproduction detailed in Section 6.3.

Table 18: Four-Dimensional Pareto Optimal Individuals from AAR-44A Asexual Reproduction

ID	% of Remaining Threats Removed	% of Remaining FA Removed	% of Remaining TTI_{Min} Removed	% Remaining TTI_{Avg} Removed
3282	33.80	58.58	0.11	0.87
3440	34.61	58.21	0.06	0.80
3499	33.80	53.36	0.11	0.88
3164	20.89	52.24	0.11	0.97
2033	16.04	45.52	0.05	3.32
3973	37.84	43.66	0.01	1.36
1799	11.20	40.30	0.05	3.35
3139	35.42	36.94	0.15	3.14
3624	40.26	36.19	0.10	2.94
3666	39.45	36.19	0.10	3.07
3828	40.26	35.82	0.15	3.22
3567	37.84	35.82	0.15	3.51
3368	41.88	34.70	0.15	3.11
3141	37.84	34.33	0.15	3.53
4106	43.49	32.84	0.15	3.06
3605	41.88	32.84	0.15	3.15
3576	41.07	31.72	0.15	3.34
3209	38.65	30.60	0.20	3.61
4328	45.91	27.61	0.16	1.62
2192	23.31	24.25	0.10	3.93
4788	48.33	23.88	0.15	3.29
3616	36.23	22.76	0.10	4.75
3168	28.15	22.39	0.15	4.45
4655	47.53	22.01	0.24	2.23
4253	44.30	16.42	0.20	4.93
4082	45.11	15.67	0.15	4.84
3604	45.11	14.18	0.15	4.97
4217	44.30	14.18	0.15	4.98
3908	45.11	13.06	0.15	4.98
4173	44.30	12.69	0.15	5.02
4371	45.11	9.70	0.20	5.04
3202	32.19	9.70	1.53	5.42
3157	32.19	8.96	1.53	5.47
3092	33.80	7.84	1.48	5.31
3504	33.80	6.72	1.53	5.39
4821	50.76	3.73	0.23	6.50
5015	50.76	2.99	0.23	6.51
5048	49.95	0.75	0.23	6.53

6.3 Sexual Evolution

Sexual reproduction was performed using three separate demes, or populations, which provided transfer of genetic material between the populations. The GTMOEA software was initialized with the output population from the asexual reproduction. The following subsections detail the various aspects of this optimization, which was more complicated than the asexual reproduction.

6.3.1 Setup

Because the product of this program was the optimized OFP, and not pure MOEA research, it was impossible to repeat the optimization multiple times. In fact, the MOEA setup was modified several times throughout the optimization process in an effort to improve the quality of the solutions found. A total of five episodes of evolution were performed to optimize the performance of the OFP using slightly different setups. Subsections 6.3.1.1 through 6.3.1.10 describe the setup of the first episode. Subsections 6.3.1.11 through 6.3.1.14 describe the differences in the setup of each of the subsequent episodes relative to their predecessor.

To create a very diverse overall population and to encourage progress in many areas, three demes were created, each with varying objectives. A multiple deme approach was selected because of the need to evolve a solution as quickly as possible. With the full evaluation of an individual requiring thirty minutes, there was little possibility of repeating evolution. Thus, multiple demes allowed evolution with three methods of thresholding simultaneously. Exchange of genetic material between demes facilitated the sharing of the characteristics of the best individuals with other demes.

6.3.1.1 Five Objective Deme

The five objective deme extracted four primary objectives, P_d, *NFAC*, TTI_{Min}, and TTI_{Avg}. An additional objective, the P_d of live-fire data only, $P_{d,\text{LF}}$, was added for fast removal of

poorly performing individuals.

The $P_{d,\mathrm{LF}}$ objective contained all fourteen training live-fire shots, and a threshold was test for $P_{d,\mathrm{LF}}$ that was equal to that achived by the default OFP from these same live-fire shots. Because only solutions that outperform the current system in all dimensions were desired, the elimination of individuals with the evaluation of only this small quantity of data allowed concentration of computational resources on individuals with the best performance.

The second objective was the P_d of all threat scenarios, both live-fire and simulated. This was accomplished using a weighted-sum of the live-fire and simulation shots, where the weights for both categories were 0.5. This was slightly different from the evaluation of evolved individuals in that the weights for the different threat tiers were not included.

The third and fourth objectives were the TTI_{Min}, and TTI_{Avg} of the live-fire and simulation shots respectively. These values also utilized the weighted-sum described above for the P_d, which were calculated by the underlying wrapping script and AAR-44A simulation at the same time as the P_d, but were only requested by the *GTMOEA* software if the previous objectives passed their associated dynamic thresholds.

The fifth and final objective was the *NFAC*. To prevent the evaluation of any other objectives, the minimum threshold for this objective is set to a positive value, which by definition can never be achieved.

6.3.1.2 Seven Objective Deme

The seven objective deme was designed to contain less training data, but contained data that was considered the most difficult. It was intended to evaluate more individuals than could be evaluated by an equal number of processors working on the five and twenty-two objective demes.

The first three objectives were P_d, TTI_{Min}, and TTI_{Avg} of the live-fire scenarios. Again,

failure to declare on at least as many live-fire training data cases as the default OFP prohibited evaluation of other objectives.

The fourth objective was the *NFAC* for the most difficult false alarm sources. Analysis of the false alarm data revealed that by careful selection of the ten minutes files, 6 out of the 120 files could be chosen that contained a large percentage of the false alarms. This reduced evaluation to only one out of twenty hours of the false alarm data.

The fifth, sixth and seventh objective were P_d, TTI_{Min}, and TTI_{Avg} of one of the five scenarios for the simulated threats listed in Section 6.1.4.2 that was determined to be the most difficult.

6.3.1.3 Twenty-two Objective Deme

The twenty-two objective deme contained the twenty-two objectives for optimization (see Section 6.1.2 for details). The first eighteen objectives were divided into six sets of the three threat objectives: P_d, TTI_{Min}, and TTI_{Avg}. The six sets were divided based on launch range and threat tier as follows:

- Objectives 1-3. Tier one threats at long range.
- Objectives 4-6. Tier two threats at long range.
- Objectives 7-9. Tier three threats at long range.
- Objectives 10-12. Tier one threats at short range.
- Objectives 13-15. Tier two threats at short range.
- Objectives 16-18. Tier three threats at short range.

The final four objectives, all dealing with false alarm performance, were the *NFAC* for the following operating conditions:

- Objective 19. *NFAC* for urban sources at high altitudes.

- Objective 20. *NFAC* of the rural sources at high altitudes.
- Objective 21. *NFAC* of the urban sources at low altitudes.
- Objective 22. *NFAC* of the rural sources at low altitudes.

6.3.1.4 Region of Interest

The *region of interest*, ROI, was then defined as the upper bound for each objective, which typically comes from the requirements. For this optimization, the region of interest was defined as a set of unreachable goals for the system. Specifically, all P_d objectives were desired to be 1.0, TTI_{Min} and TTI_{Avg} values were desired to be TTI_{Goal} seconds (a value greater than most missile flight times), and all *NFAC* values were desired to reach 0.0. This would result in an OFP that would provide a declaration for every threat possibility at least TTI_{Goal} seconds before impact and would have no false alarms. These were not realistic goals, but they were important for the ICEO methods employed to find good individuals. In particular, the hypercube distance scaler utilizes these methods to scale the fitness values.

6.3.1.5 Dynamic Objective Thresholding

The minimum thresholds for all objectives were set to be slightly lower than the performance of the default OFP for these objectives. Thus, only OFPs that outperformed the default OFP were fully evaluated. The maximum values were determined by the region of interest described in the previous section. The weights for dynamic objective thresholding were initially set to zero to disable the dynamic setting of thresholds, implying truncation of individuals with poorer performance than the current system. But, the Pareto front for the region of interest in front of the default OFP is explored.

6.3.1.6 Fitness Function

The fitness for all demes was calculated using the same method. The method included the same Pareto rank fitness solver, followed by the same SPEA2 density scaler, and the same

inversion transform as used in the asexual reproduction methods. In addition to the SPEA2 density and inversion transform scalers, the hyper-cube distance (HCD) scaler was used.

Each of the four components of the fitness calculation served a unique and distinct purpose. First, the Pareto rank 1 method determined the "raw" fitness value based on the relative Pareto optimality of the solutions. Second, the SPEA2 density methods scaled the fitness value to de-weight individuals that reside close together in objective space. This was intended to spread the probability of mating more equally along the Pareto front. The third operation, the inversion transformation, transformed values to a space proportional to the probability of mating. The fourth operation, the HCD scaler, then increased the probability of mating for individuals close to the region of interest.

6.3.1.7 Elitism

The elite rules are similar to those for asexual reproduction found in Section 6.2.2, but the parameters were varied slightly. Specifically, the maximum rank permitted was lowered from five for asexual reproduction to three for sexual reproduction to force individuals to be closer to the Pareto front for consideration. Also, the size of the elite pool, $|E|$, was increased to 100 individuals to allow for more genetic diversity.

6.3.1.8 Selection Method

Sexual reproduction utilized fitness proportionate selection from multiple populations, as described in Section 6.2.2. A majority, ninety percent, of parents were chosen from the current deme. The remaining ten percent were chosen equally from the two remaining demes. For example, the five objective deme chose ninety percent of parents from the five objective deme, five percent of objectives from the seven objective deme, and five percent of objectives from the twenty-two objective deme.

Since two parents were required for crossover, this implied that 0.9^2 or eighty-one percent of children had both parents from the current deme. Likewise, 0.1^2 or one percent of children had neither parent from the current deme, and the remaining eighteen percent of

children contained one parent from the current deme and one parent from a different deme. Thus, nineteen percent of children had genetic material from other demes.

6.3.1.9 Crossover Method

The location crossover method, described in Section 3.1.2, was utilized in this optimization to specify the exact locations for crossover. Crossover was permitted at the boundaries of all 156 attribute locations by specifying the names of all attributes.

6.3.1.10 Mutation Method

The same mutation method used during asexual reproduction was utilized (see Section 6.2.2). In particular, each of the 156 attributes was assigned a normal distribution with a standard deviation equal to ten percent of the default value. Those attributes with values of zero were given a standard deviation of 0.001. Mutation of each attribute occurred with an independent two percent probability, resulting in a $1.0 - 0.98^{156} = 95.7\%$ probability of an individual being mutated.

As noted before, each generation is independently mutated using the values from the previous generation, so the normal mutation can accumulate into large deviations over many generations. Uniform distributions were not utilized due to deleterious effects that were observed when variables were selected randomly.

6.3.1.11 Setup for Episode 2

The setup for the second episode was modified to increase the maximum number of individuals from $100,000$ to $1,000,000$. Dynamic thresholding was also enabled by changing the weights for the P_d objectives from 0.0 to 0.25 for the five and seven objective deme. The weights for the P_d objectives for the twenty-two objective deme were changed from 0.0 to 0.1. To decrease the time spent in the evolutionary operators, the number of individuals within each generation was increased from five to ten.

6.3.1.12 Setup for Episode 3

By the end of the second episode 332, 642 individuals were evaluated. To slow the mutation rate further, the rate of mutation for each variable was lowered from two percent to one percent, resulting in a $1.0 - 0.99^{156} = 79.2\%$ probability of mutation for each individual.

6.3.1.13 Setup for Episode 4

Because the results were only being used from the five objective deme, episode four reduced the number of demes from three to one. The five objective deme was kept. In hindsight, this precluded the capability to weight the results from the twenty-two objective deme after optimization in order to find the most desirable solutions.

6.3.1.14 Setup for Episode 5

The setup for episode five was identical to episode four. This run was required to restart evolution after a catastrophic computer system failure.

6.3.2 Results

As indicated by Figure 112, a total of five episodes of evolution were completed, representing a total of 576, 929 individuals. The individuals in the Figure are the Pareto optimal individuals in the P_d and *NFAC* objectives with TTI_{Min} and TTI_{Avg} that are larger than the default version of the OFP. The individuals were taken from the five objective deme for each episode. Note that the resulting convex Pareto front from episode 5 results in the ability to reduce the false alarms by 88.4 percent or to the ability to reduce the number of threats not declared by 73.4 percent.

Table 19 presents the number of objectives evaluated for each objective of each deme of each episode of evolution. From this chart note that only 48.8 percent of the 576, 929 individuals had all objectives evaluated. Also note for episodes one, two and three that the seven objective deme was able to evaluate roughly twice as many individuals as the other demes. This increase in individuals is a result of the smaller quantity of data required by

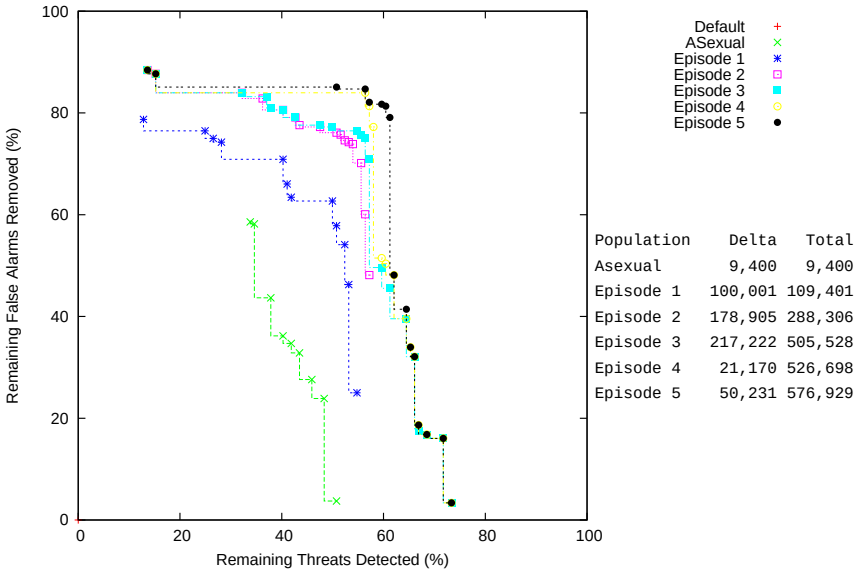

Figure 112: AAR-44A Sexual Reproduction Unweighted Training Results

this deme since it focuses on the difficult data, (see Section 6.3.1.2 for details). Also note that in general the number of individuals evaluated increases for each episode, indicating an overall improvement in the quality of individuals created. But even after many generations there are still some individuals created that can be determined not to be of interest, and can thus be terminated before objective evaluation is complete.

Figure 113 presents the four dimensional objective results for select individuals. Each of the four axes contains the response for one of the dimensions. An individual's objective response is then indicated by a shape. The innermost shape is the response of the default Version 6.1. Note that all individuals outperform the default OFP in all dimensions. Those individuals with greatest performance in each objective dimension were also selected for display. Because there were a total of 580 total Pareto optimal individuals in the four dimensional objective space, all Pareto optimal individuals could not reasonably be displayed. Therefore, starting with a list of individuals including the default individual and the four individuals with the largest value in each objective, individuals that contained the largest minimum distance in objective space as compared to any other individual in the list were added to the list. This method is similar to the SPEA clustering algorithm and

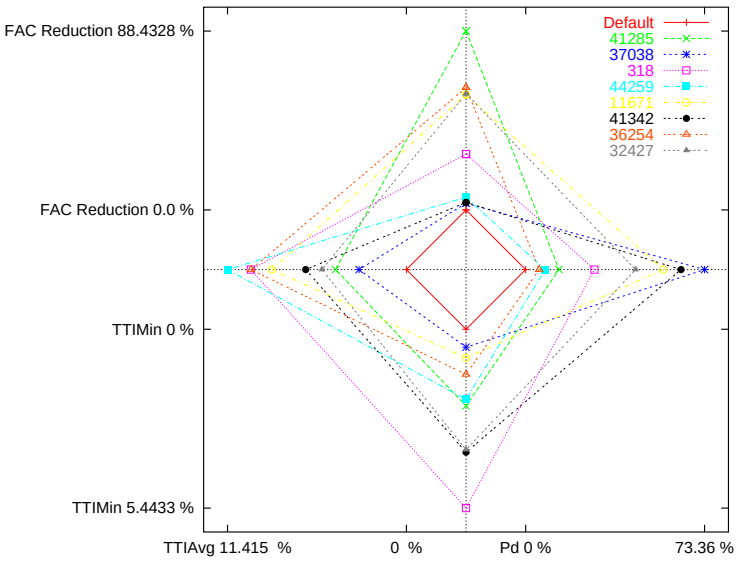

Figure 113: Four-Dimensional Sexual Reproduction Training Results

attempts to find individuals with the greatest phenotypic diversity.

Table 19: Number of Objectives and Individuals for Each Deme of Each Episode

Deme/ # Obj.	Objective Values					Total Indiv.	% Fully Solved	Total Obj.
Overall						576,929	48.8	4,048,008
Episode 1						100,001	45.6	711,731
5	24,221	18,836	17,793	16,905	13,361	24,221	55.1	91,116
7	52,603	41,673	41,310	39,743	38,020	52,603	43.2	270,385
	34,329	22,707						
22	23,177	18,994	18,989	18,938	17,419	23,177	41.3	350,230
	17,419	17,418	16,760	16,583	16,583			
	16,081	16,081	16,076	14,812	14,812			
	14,812	14,564	14,564	14,436	11,952			
	10,181	9,579						
Episode 2						178,905	47.9	1,320,484
5	41,501	33,466	32,338	30,046	23,287	41,501	56.1	160,638
7	94,032	76,004	75,135	71,569	69,305	94,032	46.4	494,887
	65,166	43,676						
22	43,372	36,022	36,011	35,993	33,647	43,372	43.2	664,959
	33,647	33,643	31,923	31,692	31,692			
	30,226	30,226	30,208	27,673	27,673			
	27,673	27,506	27,506	27,380	23,113			
	19,397	18,736						
Episode 3						217,222	56.6	1,707,628
5	49,532	42,334	41,525	38,542	32,390	49,532	65.4	204,323
7	108,226	95,035	93,516	87,243	85,108	108,226	68.2	626,778
	83,796	73,854						
22	59,464	49,451	49,390	49,249	45,519	59,464	28.3	876,527
	45,519	45,404	43,928	43,322	43,322			
	41,232	41,232	40,757	37,053	37,053			
	37,031	36,747	36,747	36,716	22,503			
	18,077	16,811						
Episode 4						21,170	82.0	94,273
5	21,170	19,830	18,482	17,430	17,361	21,170	82.0	94,273
Episode 5						50,231	76.6	213,892
5	50,231	45,359	41,022	38,781	38,499	50,231	76.6	213,892

6.4 Sensitivity Analysis

This section examines the sensitivity of the constants determined after optimization.

6.4.1 Constants With No Effect

After optimization, an effort was made to determine which constants resulted in a change in the performance of the OFP. The performance of the system was determined for both the default OFP and the optimized OFP. The sensitivity analysis was accomplished by performing two tests for each constant. First, all values were set to their optimized values except the constant under test, which was modified from the optimized value to the default value. Second, all values were set to their default values except the constant under test, which was modified from the default value to the optimized value. If the performance of the system was not modified from the values collected above, the constant was determined as a candidate to be non-sensitive.

This same analysis was performed for several Pareto-Optimal versions of the OFP to create several lists of non-sensitive candidates. The set containing the intersection of non-sensitive candidates was then determined to be non-sensitive. *This resulted in a set of constants reduced from* 156 *constants to* 76 *constants*.

6.4.2 Constants Resulting In Assertion Errors

During the preparation of software for delivery, it was determined that setting of two constants resulted in assertion errors when modified from the default values. These constants were removed from the list with minimal impact on OFP performance. Therefore, OFPs delivered for hardware evaluation did not include modification of these two constants. This resulted in a set of constants reduced from seventy-six constants to seventy-four constants.

6.4.3 Constants With Large Effect

Many constants appear to produce a small change in the OFP performance. To determine which constants were strongly coupled to OFP performance, the remaining constants were

again set back to their default values one at a time, and the performance results were gathered. This resulted in a large number of constants that when considered independently did not drastically change the performance. These values were removed from consideration, and the performance of the optimized values for the remaining variables calculated. From this analysis, only twenty-two of the seventy-four constants produced the majority of the optimization effect, although, the inclusion of the all seventy-four variables was required to obtain equivalent TTI_{Min} and TTI_{Avg} performance.

6.5 Evaluation

Approximately fifty individuals that were Pareto optimal in the four dimensional training objective space were selected for evaluation against the data reserved for evaluation. This data included fourteen live-fire shots, twenty hours of false alarm data, and 3108 simulated threats. The data from these runs was then used to find the individuals that were Pareto optimal in the four dimensional evaluation objective space. When calculating the objective values for the four dimensional space, a weighting was utilized. For each group of threat data live-fires were weighted by $\frac{1}{2}$ and simulated threats were weighted by $\frac{1}{2}$. Then each threat tier was also given a weight.

$$P_{d,\text{LF}} = \frac{4}{7}P_{d,\text{LF}}(\text{Tier1}) + \frac{2}{7}P_{d,\text{LF}}(\text{Tier2}) + \frac{1}{7}P_{d,\text{LF}}(\text{Tier3}) \quad (144)$$

$$P_{d,\text{Sim}} = \frac{4}{7}P_{d,\text{Sim}}(\text{Tier1}) + \frac{2}{7}P_{d,\text{Sim}}(\text{Tier2}) + \frac{1}{7}P_{d,\text{Sim}}(\text{Tier3}) \quad (145)$$

$$P_d = \frac{1}{2}P_{d,\text{LF}} + \frac{1}{2}P_{d,\text{Sim}} \quad (146)$$

Likewise,

$$TTI_{\text{Min,LF}} = \frac{4}{7}TTI_{\text{Min,LF}}(\text{Tier1}) + \frac{2}{7}TTI_{\text{Min,LF}}(\text{Tier2}) + \frac{1}{7}TTI_{\text{Min,LF}}(\text{Tier3}) \quad (147)$$

$$TTI_{\text{Min,Sim}} = \frac{4}{7}TTI_{\text{Min,Sim}}(\text{Tier1}) + \frac{2}{7}TTI_{\text{Min,Sim}}(\text{Tier2}) + \frac{1}{7}TTI_{\text{Min,Sim}}(\text{Tier3}) \quad (148)$$

$$TTI_{\text{Min}} = \frac{1}{2}TTI_{\text{Min,LF}} + \frac{1}{2}TTI_{\text{Min,Sim}} \quad (149)$$

and

$$TTI_{\text{Avg,LF}} = \frac{4}{7}TTI_{\text{Avg,LF}}(\text{Tier1}) + \frac{2}{7}TTI_{\text{Avg,LF}}(\text{Tier2}) + \frac{1}{7}TTI_{\text{Avg,LF}}(\text{Tier3}) \quad (150)$$

$$TTI_{\text{Avg,Sim}} = \frac{4}{7}TTI_{\text{Avg,Sim}}(\text{Tier1}) + \frac{2}{7}TTI_{\text{Avg,Sim}}(\text{Tier2}) + \frac{1}{7}TTI_{\text{Avg,Sim}}(\text{Tier3}) \quad (151)$$

$$TTI_{\text{Avg}} = \frac{1}{2}TTI_{\text{Avg,LF}} + \frac{1}{2}TTI_{\text{Avg,Sim}} \quad (152)$$

For the false alarm data, all data was weighted equally.

Figure 114 illustrates the evaluation performance for the resulting Pareto optimal individuals in two dimensions. Note that the false alarm performance spans the same range for evaluation that was spanned by the training data. In contrast, the probability of detection performance is somewhat reduced. This reduction in performance results from two sources. First, the training results were not weighted, whereas the evaluation results were weighted by Equations 144, 145 and 146. Second, the small number of live-fire shots coupled with the threat weights allows a single shot to drastically affect the results. For the training data all live-fire shots results were declared before and after optimization, removing the sensitivity to the live-fire data. This was not true for the evaluation live-fire data.

6.6 Use of the Altitude For Specialized Optimization

An additional evolution was performed to test the ability to evolve better solutions against smaller, more specific sets of data. Specifically, optimization was performed for high altitude shots. This optimization was seeded with the best results of the overall optimization at the time, i.e., the results of Episode 2. Figure 115 displays the starting population for the optimization and the results of continued optimization, using all altitudes versus high-altitudes only. These results are presented in the all-altitude and high-altitude objective domains. The starting and ending results for the all-altitude optimization were mapped from the all-altitude domain to the high-altitude domain for comparison. The all-altitude optimization required an additional $288,623$ individuals. The high-altitude optimization started with these mapped individuals and was able to produce improvement in P_d over

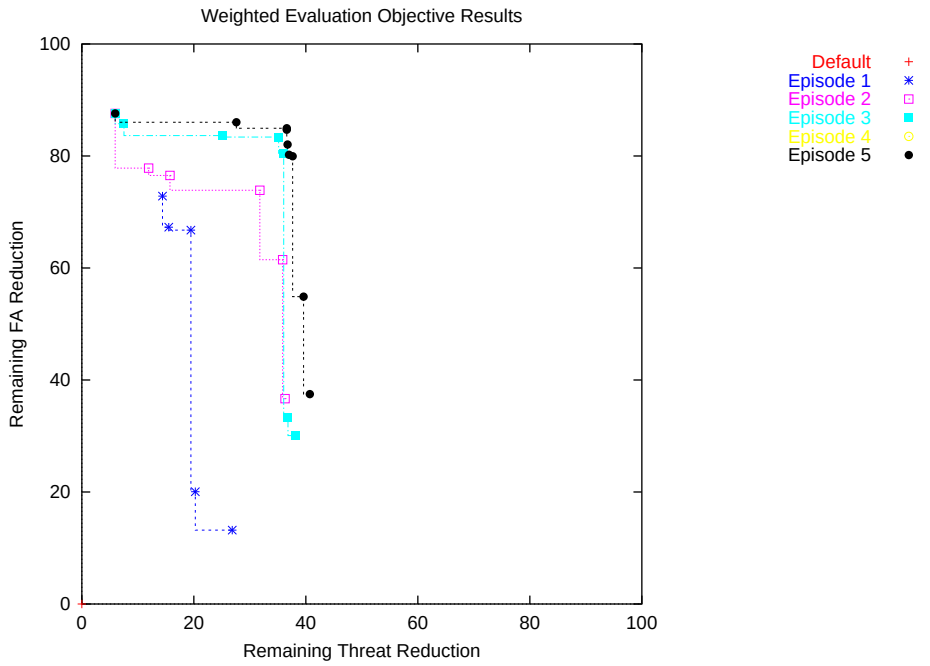

Figure 114: AAR-44A Sexual Reproduction Weighted Evaluation Results

that of the all-altitude optimization with only 56,018 individuals. Unfortunately, when the individuals optimized for high-altitudes were mapped back to the all-altitude domain, they performed worse than the default OFP.

The source of the poorer performance for the high-altitude optimization, when mapped back to the all-altitude domain, was a result of a reduction in the TTI_{Min} performance for lower altitude shorter shots. Therefore, the optimization had found the ability to utilize the additional time afforded by the longer range shots to improve performance. This concentrated optimization on high-altitude was able to significantly improve the P_d and *NFAC* performance without affecting the TTI_{Min} and TTI_{Avg} performance for the high-altitude cases. This improved performance could be realized in the OFP by switching the values of constants based on the altitude of the aircraft.

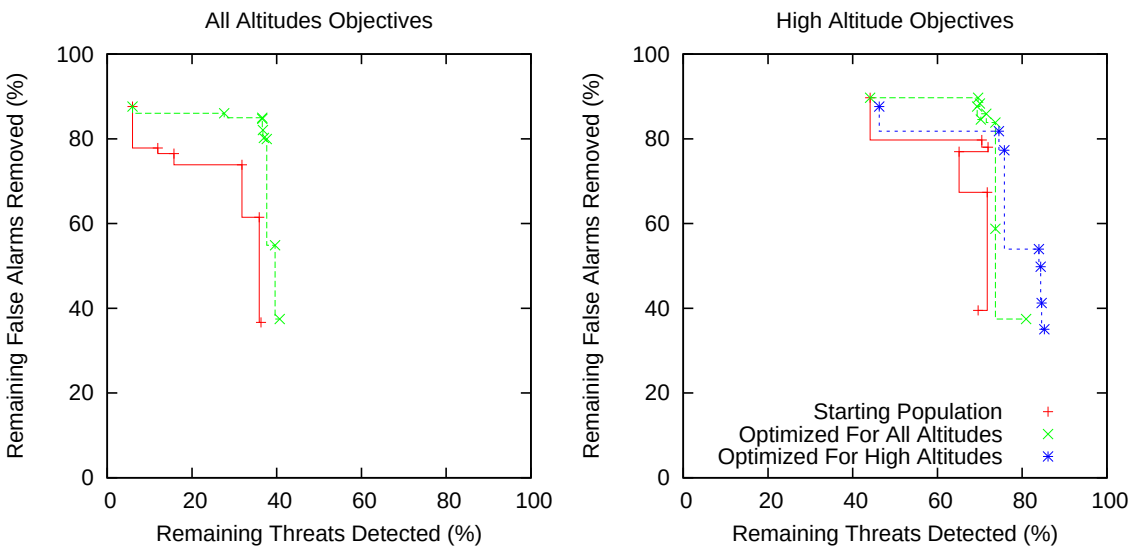

Figure 115: AAR-44A Sexual Reproduction Weighted Evaluation Results for High Altitudes

6.7 Conclusions

The research in optimization of the AAR-44A resulted in several conclusions that were applicable to MOEA optimization in general. First, the combinations of previous MOEA algorithms coupled with the new ICEO methods of dynamic objective thresholding, hypercube distance scaling, and multiple demes is able to improve the performance of a very complex system. Even the evaluation of 576, 929 individuals is able to be achieved using parallel evaluation and objective thresholding within one month of calendar time.

Second, tuning a system is different from designing a new system. As such, the ability to start from a single known solution is required for tuning a system. In retrospect, asexual reproduction is not required. What is required is a short period of high mutation using normal distributions to create a genotypicly and phenotypicly diverse population. This high mutation period is then followed by a longer period of lower rate mutation to provide evaluation of different combinations of the new genetic material.

Third, although identification and removal of variables from the search space does dramatically reduce the size of the search space, it does not dramatically reduce the time

required to explore the search space when tuning an existing system. For example, imagine the worst case where a variable would result in a fatal allele if modified to anything but a known good value. Under this circumstance the variable can only be modified from the known good value by mutation, i.e. (crossover does not affect the attribute values because both parents have the same attribute value). Therefore, bad individuals will only be created by the rate of the mutation P_m for that attribute, which typically is around one to two percent. Therefore, it is better to add another variable to the search space than to miss one important variable. As illustrated by this optimization, 156 variables are selected, and sensitivity analysis at the end show only 74 of the variables are important.

CHAPTER VII

GENETIC PROGRAMMING

The term Genetic Programming (GP) was coined by John Koza [34, 36]. Examination of the term programming will help in understanding this term. A simple description of a computer program is that it receives input from the user via a keyboard, mouse or data file, performs a set of calculations on the inputs, and presents the results of the calculation to the user. In an abstract sense, a computer program is a filter that modifies a set of inputs to create a set of outputs. GP utilizes EAs to optimize a filtering function by using the genetic operators directly on the filter program. The definition of GP is used not just for computer filtering programs, but also for other types of signal processing filters and system descriptions as well.

EAs are used to optimize any multi-input, single output function. Given a computer program, EAs are able to traverse the search space of possible parameters settings to find a set of optimized solutions that produce the desired output. GPs not only modify the parameters associated with the processing components, but are also able to modify to inter-connections of the components to create new and novel topologies.

This chapter introduces the basics of traditional inverted tree structure GP methods along with several example applications in Section 7.1. Then, given these basics, a new block-diagram oriented method is proposed in Section 7.2 that utilizes a linear genome, the higher level components and simulation capabilities of Ptolemy II [40], and the MOEA ICEO contributions of Chapters 3 and 4.

Section 7.3 introduces a standard pattern recognition problem based on data from the 1994 census [32]. This problem is then solved using a weighted-sum optimized by MOEA

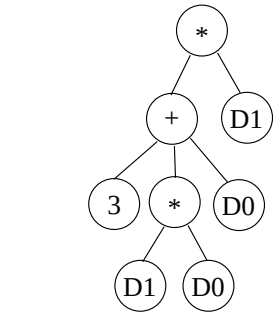

Figure 116: Genetic Programming Tree

methods in Section 7.4 to find a Pareto front using a simple static topology. The block-diagram oriented method is then applied in Section 7.5 to the same census problem to explore the ability of the proposed block-diagram technique to find more complex and better performing solutions than can be found with the weighted-sum. This chapter is concluded with Section 7.6, which summarizes the contributions of this chapter.

7.1 Traditional Genetic Programming Methods

7.1.1 LISP Expressions

Although GP has been implemented in C, C++, Java, and Mathematica, the concepts of GP are most easy to understand in the LISP programming language. LISP S-expressions are *prefix notations*, meaning that functions are followed by their arguments. For example, the C expression `1 + 2 + 3` is written in an S-expression as `(+ 1 2 3)`. An argument to a function can be a function evaluation itself. In this case the nested function is enclosed in parentheses as well. For example, the C expression `1 + (2*3) + 4` has an S-expression `(+ 1 (* 2 3) 4)`. Most programs support reuse by allowing varied input from the user. These inputs are denoted as (`D0, D1,...`). Figure 116 illustrates a GP root tree for an S-Expression.

7.1.2 Genetic Programming Operators

GP utilizes the same four concepts of fitness evaluation, selection, crossover, and mutation as the GA process. The majority of GA applications work on a fixed length genome description. GP, on the other hand, must support the growth of a program.

In order to associate a GP genome with its performance, the program is executed and the results compared to the desired output. For signal processing applications such as the circuit design of a bandpass filter, this is the comparison of the circuit output to the desired output. A typical method is to minimize the sum of the squares of the difference between the desired and derived output for a number of frequencies. Typically multiple objectives are combined into a single-objective using a weighted-sum. No literature was found that used multiple objectives to find a set of Pareto optimal solutions for GP. Fortunately, there are no expected problems in using Pareto based MOEA techniques, and results may be improved by the genetic diversity of solutions.

The crossover operation in GP is implemented by combining sub-trees from two parents. In this process a node in the tree structure of each of the two parents is randomly selected as a crossover location. This node becomes the fragment, and the remaining tree structure containing the root is defined as the remainder. Two children are then created by combining the remainder of parent 1 with the fragment of parent 2 and the fragment of parent 1 with the remainder of parent 2. Figure 117 illustrates this operation as two parent S-expressions crossover to create two children.

GP mutation is performed on a small percentage of children by selecting a subtree of the child and deleting this subtree. A new random sub-tree is grown in a manner identical to the creation of the initial random population. This new sub-tree is attached at the point of deletion.

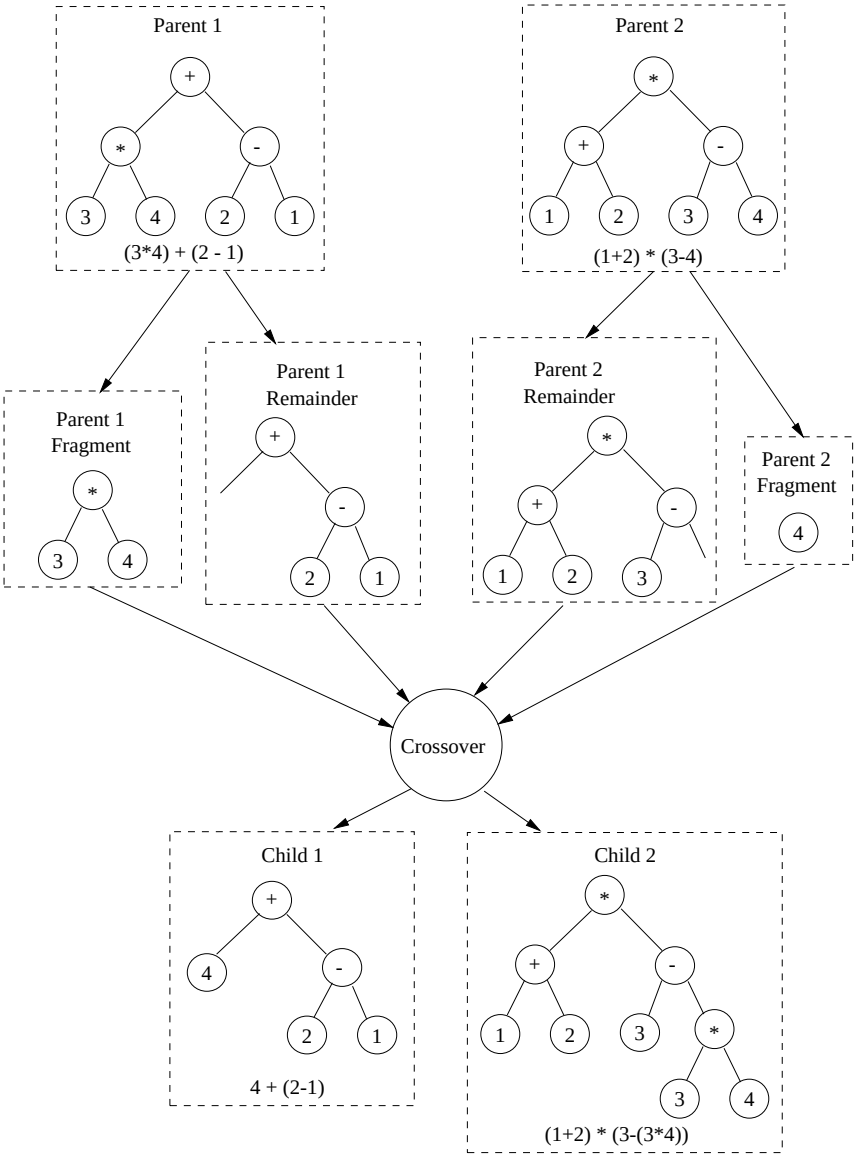

Figure 117: Genetic Programming Crossover Example

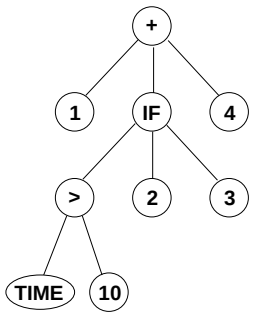

Figure 118: Conditional Tree with Ordered Branches for Expression `(+ 1 (IF (> TIME 10) 2 3) 4 )`

7.1.3 Genetic Programming Components

This section details, in terms of computer programming components, those elements desired in creating a GP. Because GP works in the programming space, the ability to manipulate the same concepts used by human programmers is desirable.

7.1.3.1 Conditional Statements

Conditional statements can be formulated as ordered arguments to a function. For example, the C-expression

```
{temp=1; if(TIME > 10) temp+=2; else temp+=3; return(temp+4);}
```

can be written in an S-expression as

```
(+ 1 (IF (> TIME 10) 2 3) 4 ).
```

Figure 118 illustrates the previous S-expressions decomposed into a root point-labeled tree with ordered branches. Other traditional conditional statements such as case statements and loops can be supported in a similar way.

7.1.3.2 Automatically Defined Functions

Computer programmers and circuit designers commonly organize components into larger reusable components. For a computer programmer, the larger component is a subroutine.

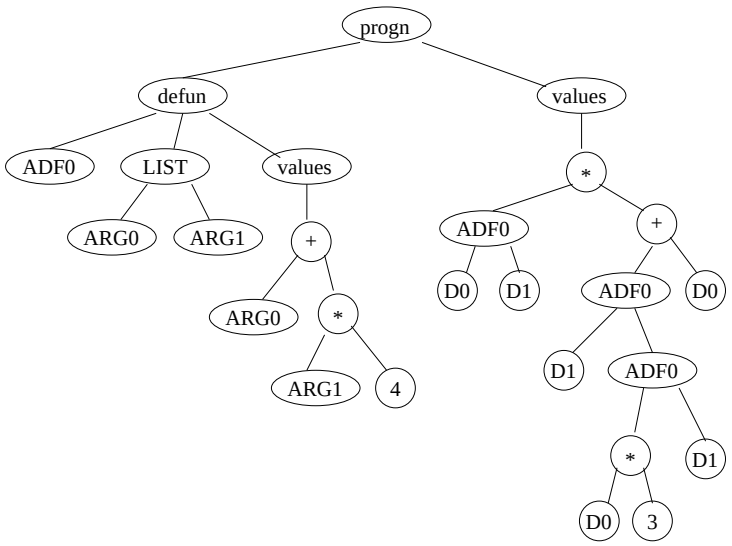

Figure 119: Genetic Programming Tree with Automatically Defined Functions

For a circuit designer, the larger element may be an amplifier circuit. These components may be used multiple times within the program, with varying arguments.

These same concepts are supported in GP with *Automatically Defined Functions* (ADFs). Each ADF contains zero or more variables as arguments. ADFs are evolved along with the main program that references these functions. The root of the program contains two types of nodes; *define functions* (`defun`) nodes define an ADF, and a `values` node contains the main program. Figure 119 illustrates a tree that references an ADF. The `defun` node contains three ordered nodes; the name or index of the function, a list of arguments for the function, and the ADF processing node. Other ADFs can be added with additional `defun` nodes.

7.1.3.3 Automatically Defined Storage

Computer programmers find it convenient, and often necessary, to use variables for the communication of information to other portions of a computer program. This is typically done by humans by assigning a descriptive name to the value. There are an infinite number of possible names that can be assigned to a computer program. An infinite number of

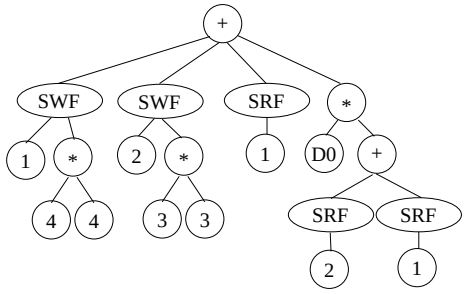

Figure 120: Genetic Programming Tree with Automatically Defined Storage

memory elements is not desirable to encode in a genome. Therefore, GP goes a step below human encoding of normal computer programs in its implementation of variables through Automatically Defined Storage (ADS). One method is to use indexed memory with two new functions. Let the *Storage Reading Function* (SRF) have one argument, which is the index into memory. Let the *Storage Writing Function* (SWF) have two arguments, which are the index of the element to be written and the values to be written. Figure 120 illustrates the use of the SRF and SWF to communicate between different branches of the tree. Other functions can be added to support interfaces with different data types.

7.1.4 Genetic Programming Sets

As have been described above, there are many function types and data types that can be transferred within the Genetic Program. The decision of the components to use is often domain specific, and may be implementation dependent. The components allowed are often specified in terms of various sets. The functions allowed for the *Automatically Defined Function* are given by F_{adf}, e.g., $F_{adf} = \{AND, OR, NAND, NOR\}$. The terminal set of are leaves of the inverted tree structures,i.e., those nodes with no inputs. The terminal set for the ADF is defined as T_{adf}, e.g., $T_{adf} = \{ARG0, ARG1\}$. Likewise the *Result Producing Branch* may have different function and terminal sets. For example, $F_{rpb} = \{AND, OR, NAND, NOR, SRB, SWB, ADF0, ADF1\}$ and $T_{rpb} = \{D0, D1, D2\}$. Note that the ADFs are included in the F_{rpb}.

7.1.5 Example Applications

This section supplies a summary of three applications of GP that span several domains. Knowledge of examples of "traditional" GP techniques helps in the motivation and critique of the proposed block-diagram method in Section 7.2.

7.1.5.1 Pattern Recognition

In 1999, Howard, Roberts and Brankin [31] utilized GP to find ships in Synthetic Aperture Radar SAR imagery. Interestingly, they used domain knowledge to break this problem down into two components. First, a filter was evolved that could detect all target pixels, even though it had some false positives. Then a second-stage filter was evolved to remove the false positives from the data. This two-stage process was implemented to reduce the processing time for the second-stage filter, as it only had to process the target pixels found in the first stage.

There were no ADFs. The T_{rpb} included integer constants, real constants and the pixel statistics from 10 variations of centered averages and perimeter averages. Training and test data was 50 and 100 meter SAR data of the English Channel taken by the European Remote Sensing (ERS) satellite. The training data contained 59 different targets. Some of the results evolved included components similar to that of human-designed systems. For example, the first stage derived a simple yet effective *spot* detection algorithm:

$$value = pixel\_value - local\_mean - local\_variance - constant \tag{153}$$

7.1.5.2 Signal Transforms

In 1996, Koza [37, 35] reported use of GP to create a band-pass filter. The fitness function evaluator used a weighted-sum of results from SPICE. An embryonic electrical circuit combined with architecture-altering functions were used to create the circuits. The genome did not directly contain the circuit, but rather contained a program tree containing architecture-altering and component-altering functions that when applied to the embryonic

circuit created a circuit for analysis.

The method also allowed for ADFs. This in effect created small circuits that could be easily duplicated. The use of ADFs manifested itself in the creation of repetitive, two, three and four rung ladder topologies, as well as Cauer topologies similar to human derived solutions. Most importantly, the GP finally created a variation of the Cauer filter that outperforms human-derived solutions.

7.1.5.3 Digital Signal Processing

The design of second-order Infinite Impulse Response IIR filters with low coefficient sensitivity was performed using GP as reported by Uesaka and Kawamata [56] in 2000. In their implementation,

$$T_{adf} = \{x, b_0, b_1, ..., b_7\} \quad (154)$$

$$F_{adf} = \{m, a, d\} \quad (155)$$

$$T_{rpb} = \{x, b_0, b_1, ..., b_7\} \quad (156)$$

$$F_{rpb} = \{m, a, d\} \quad (157)$$

where m is a multiplier, a is an adder, d is a delay, $\{b_0, b_1, ..., b_7\}$ are the coefficients, and x is the input. An example S-expression, direct form, and program tree taken from [56] are given in Figure 121.

The fitness function for their research was a sum of the measures of the sensitivity, size, physical realizability, and realization of desired transfer function. The method performed well for the design of both lowpass and bandpass filters.

7.2 Research Contributions

The research contribution of this chapter is a block-diagram oriented method for performing GP. Signal design is typically communicated by humans using block diagrams. Feedback within components is naturally communicated by these block diagrams. GP, on the

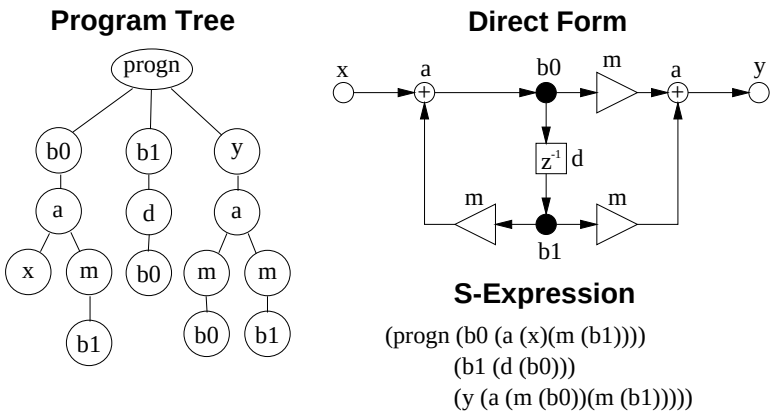

Figure 121: Program Tree, Direct Form, and S-Expression for DSP

other hand, typically uses embryonic circuits [37, 35], circular connection of branches [56], or Automatically Defined Storage [36], to allow feedback.

For signal processing design, it may be more useful to allow connections of components in a block diagram. Figure 122 illustrates a conceptual design for such a system. Each processing block is connected to a separate output line. The selection of inputs for a processing block is controlled by the genome and allows connection to any system input or processing block output. The type of function and the function-specific parameters are also controlled by the genome. The selection of possible functions is chosen from the problem domain. For example, image processing systems may include centered averages and perimeter averages [31]. The ADFs of GP provide another method commonly used by humans, which is the reuse of large components. By allowing several of the functions to be other block diagrams, this system also supports the GP power of ADFs. To realize the method of block-diagram design a software package identified as Pattern Recognition Evolutionary Synthesis Through Optimization (PRESTO) has been created.

The PRESTO software combines the pre-existing GTMOEA software together with the Ptolemy II software using a newly developed Modeling Markup Language (MoML) generator created by the author. Figure 123 illustrates how these software packages are combined to create the desired results. The following three subsections describe each of

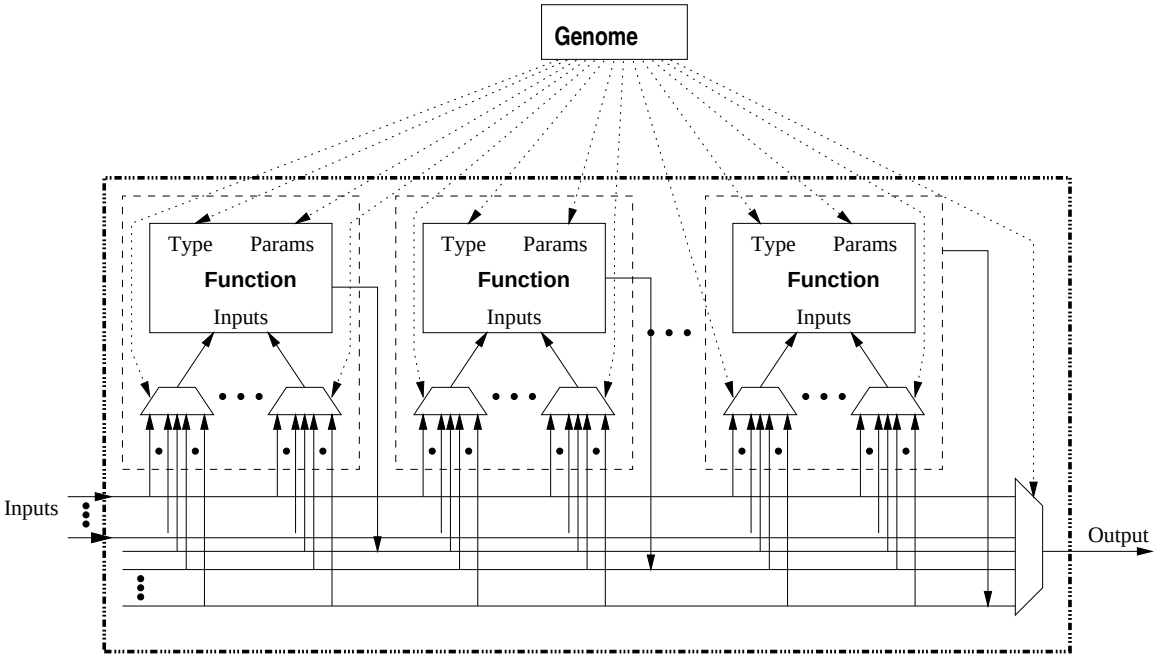

Figure 122: Proposed EA programmed Block Diagram

the software packages starting with the lowest level simulation package Ptolemy II.

7.2.1 Ptolemy II

The Ptolemy II software package is under active development by University of California Berkeley under the direction of Edward Lee using multiple sources of funding including the Defense Advanced Research Project Agency (DARPA). Components in Ptolemy II are based on the concept of actors [41], which are both data polymorphic and domain polymorphic. Many modern languages are data polymorphic, e.g., the addition operator works for integer, double, and string types. Ptolemy II adds the concept of domain polymorphism, which is the functioning of the actors in many simulation domains. Ptolemy II contains many simulation domains including: synchronous data flow, discrete event, process networks, finite state machine, communicating sequential processes and continuous time. Ptolemy II offers the ability to create hierarchical models [45], where different hierarchical components can utilize different simulation domains.

The major reason for the selection of Ptolemy II as the system simulation infrastructure

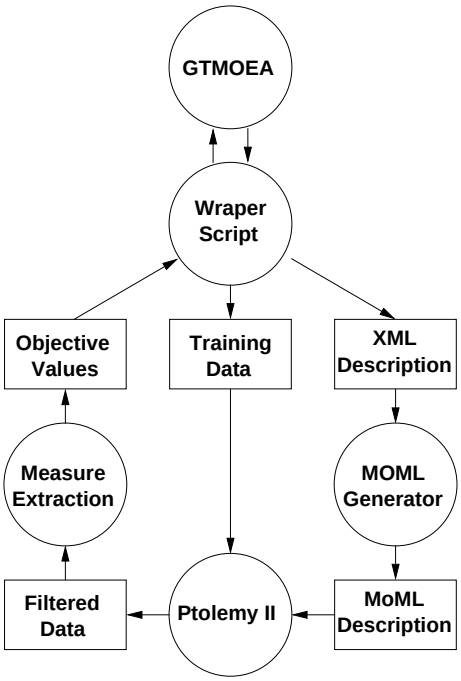

Figure 123: PRESTO Software Architecture

is the extensible simulation capability for many simulation and application domains [19]. The software offers a toolbox of existing objects such as low-level *and* and *or* gates, as well as higher-level objects such as FFTs and image processing algorithms. If the existing objects do not supply the desired functionality, additional user defined actors can be supplied by:

1. Building composite actors from the low-level Ptolemy actors.
2. Creating new actors in JAVA.
3. Import of actors created in Matlab.
4. Import of actors created using Python scripts.
5. Wrapping existing simulations from other languages using Java Native Interface (JNI).

A visual editor for Ptolemy II simulations called Vergil exists. For the purposes of

PRESTO, this tool allows visualization of the evolved simulations, as will be demonstrated in Section 7.5. But for PRESTO, the ability to create Ptolemy II simulations from a standard text file without the need for a visual editor was imperative. The standard text file format Ptolemy II utilizes to encode all simulations is a Modeling Markup Language (MoML). MoML is a dialect of XML which encodes the actors utilized, the attributes of the actors, and the relations between the actors. Appendix B gives the DTD for the MoML syntax.

7.2.2 Modeling Markup Language Generator

The MoML generator is responsible for translating model descriptions from the genome format of the MOEA software to the system description MoML format of the Ptolemy II software. Figure 124 illustrates the processing for the MoML generator. The translation from a Genome to MoML description requires additional information about the processing elements that is not available in the MoML description. Therefore, a set of meta-data associated with each element is kept in an element specification list. The DTD for the element specification list is given in as follows:

```
<!ELEMENT ElementSpecList (File*,ElementSpec*)>
  <!ELEMENT File (#PCDATA)>
  <!ELEMENT ElementSpec (Name,Type,
       (entity|port|property),MinInputs,Input*,
       MinOutputs, Output*,Attribute*, Feedback?)>
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Type (#PCDATA)>
    <!ELEMENT MinInputs (#PCDATA)>
    <!ELEMENT Input (Name,Type+,TypeMatch*,Min,Max)>
      <!ELEMENT TypeMatch (#PCDATA)>
      <!ELEMENT Min (#PCDATA)>
      <!ELEMENT Max (#PCDATA)>
    <!ELEMENT MinOutputs (#PCDATA)>
    <!ELEMENT Output (Name,Type+,TypeMatch*,Min,Max)>
    <!ELEMENT Attribute (Name,(Enum|Value|Double))>
      <!ELEMENT Enum (Map*)>
        <!ELEMENT Map (#PCDATA)>
      <!ELEMENT Value (Min,Max,Delta)>
        <!ELEMENT Delta (#PCDATA)>
      <!ELEMENT Double (Min?,Max?)>
```

```
<!ELEMENT Feedback (#PCDATA)>
```

The element specification list contains a set of file names and element specifications. The file names allow breaking apart the element specifications into multiple files for convenience. An element specification contains the name of the element, the type of the element, the MoML description of the element, information about the number and types of input and output ports for the element, information about any attributes, and a flag indicating if the element supports feedback. The element types supported are as follows:

- *INPUT* elements only supply inputs to the model, and thus do not need an input themselves for proper operation. For example, a data file reader is an input element.
- *OUTPUT* elements only receive outputs from the model, and thus do not need an output themselves for proper operation. For example, a data file writer is an output element.
- *INPUT_PORT* elements are Ptolemy II input port actors used by a composite actor to receive input from a higher level component.
- *OUTPUT_PORT* elements are Ptolemy II output port actors used by a composite actor to send output to a higher level component.
- *FILTER* elements are actors that requires both inputs and outputs for proper operation.
- *DIRECTOR* elements are Ptolemy II model directors used to specify the simulation domain used by the model or model composite actor. Director elements do not have input or output ports.

Information about the input and output ports includes the minimum number of inputs and outputs required for the actor, and a specification for each input and output port. The port specification for each port, whether input or output, contains the following:

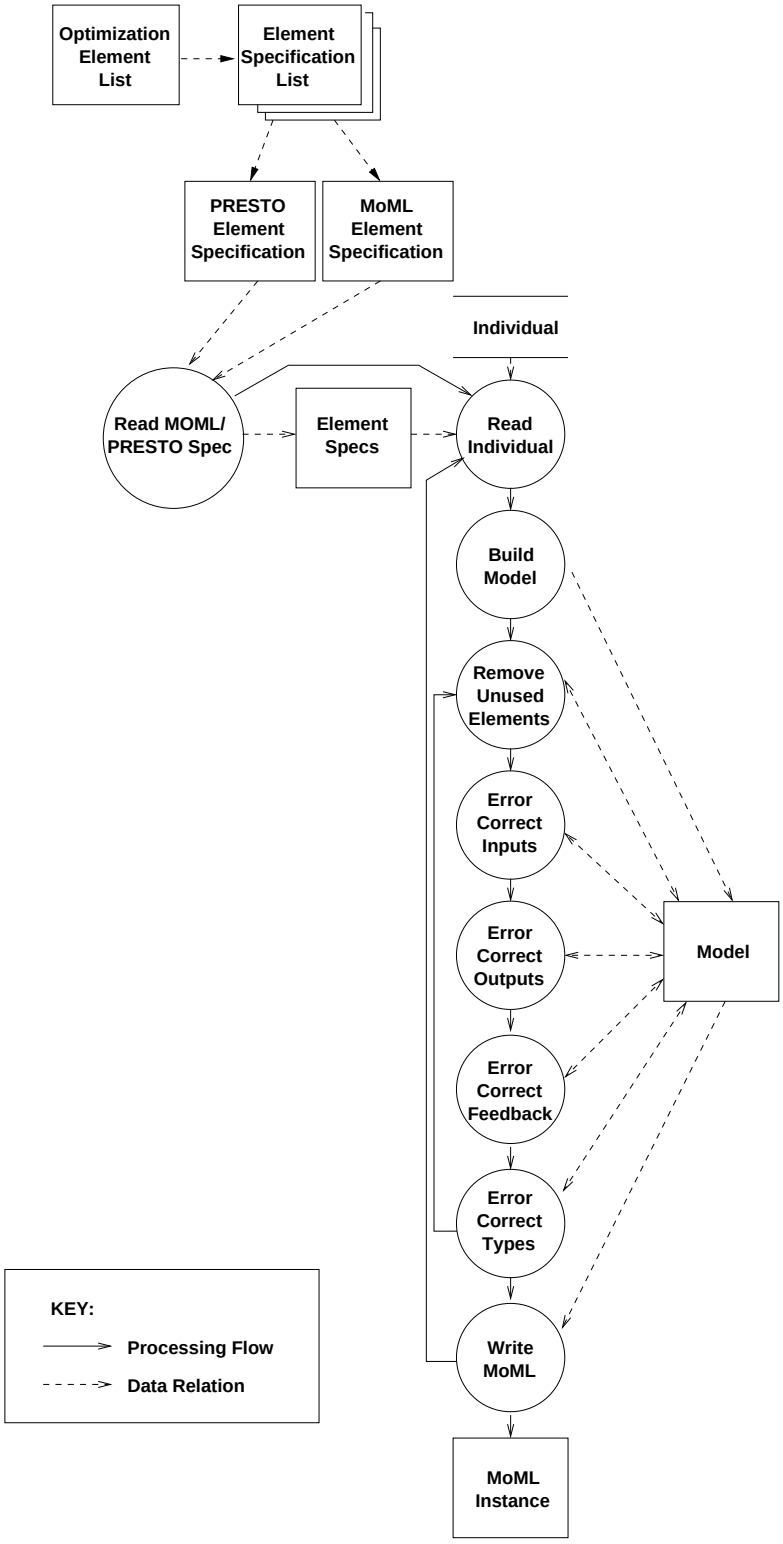

Figure 124: Modeling Markup Language Generator Processing Flow

- *Name* supplies a mnemonic for the port. It is used for references to Ptolemy II actor ports.
- *Type* strings provide a list of the types supported by this port, e.g., double, int, string.
- *TypeMatch* strings provide a list of names for ports whose types must match this port. For example, the type for the input and output ports must match for many actors.
- *Min* specifies the minimum number of connections to this port.
- *Max* specifies the maximum number of connections to this port.

The following listing provides an example element specification for the *AddSubtract* Ptolemy II actor.

```
<ElementSpec>
 <Name>AddSubtract</Name>
 <Type>FILTER</Type>
 <entity name="AddSubtract" class="ptolemy.actor.lib.↓
→    AddSubtract">
  <port name="plus" class="ptolemy.actor.TypedIOPort">
   <property name="input"/>
   <property name="multiport"/> </port>
  <port name="minus" class="ptolemy.actor.TypedIOPort">
   <property name="input"/>
   <property name="multiport"/> </port>
  <port name="output" class="ptolemy.actor.TypedIOPort">
   <property name="output"/> </port>
 </entity>
 <MinInputs>2</MinInputs>
 <Input>
  <Name>plus</Name>
  <Type>int</Type><Type>double</Type><Type>string</Type>
    <Type>boolean</Type>
  <TypeMatch>minus</TypeMatch>
  <TypeMatch>output</TypeMatch>
  <Min>0</Min><Max>100</Max>
 </Input>
 <Input>
  <Name>minus</Name>
  <Type>int</Type><Type>double</Type><Type>string</Type>
```

```
      <Type>boolean</Type>
    <TypeMatch>plus</TypeMatch>
      <TypeMatch>output</TypeMatch>
    <Min>0</Min><Max>100</Max>
  </Input>
  <MinOutputs>1</MinOutputs>
  <Output>
    <Name>output</Name>
    <Type>int</Type><Type>double</Type><Type>string</Type>
        <Type>boolean</Type>
    <TypeMatch>plus</TypeMatch>
        <TypeMatch>minus</TypeMatch>
    <Min>1</Min><Max>100</Max>
  </Output>
</ElementSpec>
```

This example provides some simple examples of the need for the information supplied for port parameters. This element contains two input ports, named *plus* and *minus*, and one output port, named *output*. The total number of inputs required is set to two, even though the minimum number of inputs required by the *plus* and *minus* ports is set to zero. This implies that there needs to be at least two input connections; but it does not matter if it is one from each port or both from only one of the ports. The typing of the ports indicates that the ports individually support int, double, string and boolean data types. The type match specifications then indicate that all three of the ports must match in their data types for a specific instance.

Other actors will of course have attribute values that may also be manipulated by the genome. The following listing provides an example of the *Scale* actor that has two attributes: a scale *factor* of type double, and *scaleOnLeft* flag with enumerated types of *true* and *false*.

```
<ElementSpec>
  <Name>Scale</Name>
  <Type>FILTER</Type>
  <entity name="Scale" class="ptolemy.actor.lib.Scale">
    <property name="factor" class="ptolemy.data.expr.↓
→       Parameter" value="1">
```

```
    </property>
    <property name="scaleOnLeft" class="ptolemy.data.expr.↓
→       Parameter" value="true">
    </property>
    <port name="input" class="ptolemy.actor.TypedIOPort">
     <property name="input"/> </port>
    <port name="output" class="ptolemy.actor.TypedIOPort">
     <property name="output"/> </port>
   </entity>
   <MinInputs>1</MinInputs>
   <Input>
    <Name>input</Name>
    <Type>int</Type><Type>double</Type>
    <TypeMatch>output</TypeMatch>
    <Min>1</Min><Max>1</Max>
   </Input>
   <MinOutputs>1</MinOutputs>
   <Output>
    <Name>output</Name>
    <Type>int</Type><Type>double</Type>
    <TypeMatch>input</TypeMatch>
    <Min>1</Min><Max>100</Max>
   </Output>
   <Attribute>
    <Name>factor</Name>
    <Double></Double>
   </Attribute>
   <Attribute>
     <Name>scaleOnLeft</Name>
     <Enum><Map>true</Map><Map>false</Map></Enum>
   </Attribute>
  </ElementSpec>
```

The conversion software reads a description of the genome in XML format, and must build an appropriate MoML model. As indicated in Figure 122 each element is assigned an output "bus", which is implemented by assigning a port number to each output port of the model elements. The elements' inputs are then linked to these ports by associating an input element's port name with an output port number. The following provides an example of a genome description for an *AddSubtract* element.

```
<Element>
  <Name>AddSubtract</Name>
  <InputPort><Name>plus</Name><Value>8</Value>
    </InputPort>
  <InputPort><Name>plus</Name><Value>17</Value>
    </InputPort>
  <InputPort><Name>plus</Name><Value>30</Value>
    </InputPort>
  <InputPort><Name>minus</Name><Value>3</Value>
    </InputPort>
  <InputPort><Name>minus</Name><Value>7</Value>
    </InputPort>
  <InputPort><Name>minus</Name><Value>9</Value>
    </InputPort>
</Element>
```

For this example, the *plus* input port is connected to output ports 8, 17 and 30. The *minus* input port is connected to output ports 3,7 and 9. The output port number for this element is implicit, with the port number determined by the position of the element in the file.

Once the genome description of the model has been read, all of the port numbers are translated to links between elements. Not every instance of a MoML description is valid. Invalid descriptions result in fatal individuals. Much as with the 0/1 Knapsack problem from Section 4.1.1, use of a genome correction algorithm can greatly reduce or eliminate the number of fatal individuals created, thus improving evolutionary performance. As indicated in Figure 124, the following sources of errors are iteratively searched for and corrected until there are no more errors found by any of the five error detection methods:

1. Unused Elements
2. Incorrect number of output ports
3. Incorrect number of input ports
4. Impossible feedback loops

5. Incorrect data types

All error correction methods are made by traversing recursively backward from the output of the model through all the elements connected to the input of each element. As an element is checked, it is marked as processed. Once an element is marked as processed, it does not have to be checked again. This method prevents rechecking sections of the system that have already been checked, and prevents an infinite loop in the case of systems that contain feedback.

Unused elements are determined by recursively marking all elements used by the output port. Elements that have no influence on the output are then eliminated from the model.

An incorrect number of output ports occurs when more input ports are connected to the output port than allowed by the element specification list. If too many input ports are connected to the output of the current element, then links are removed between the output of the current element and input to the elements until the desired number of connections is reached by removing the links from the end of the list of links. In practice, this rarely occurs because output ports can typically be connected to a large number of other actors, as indicated in Figure 122.

Error correction for the number of input ports is determined by finding the total number of elements attached to all ports, and checking against the minimum number of inputs required. If there are not enough input connections to an element it is deleted from the model. Prevention for connecting too many inputs to a single port is handled during the creation of the model. Once the specified maximum number of input connections to an element is reached, no other inputs are allowed to connect to that element.

Error correction for data types is also handled from the output to the input. This ensures that the output has the desired data type. Where required, this process forces the input ports to take on the data types of the output port. With the data type of the input set, then if an element's output connected to this input does not support the type requested, an additional type conversion element must be inserted between the input of the element under

investigation and the output of the element connected to the input. If the type conversion is not possible, then the link to that input element is removed.

Feedback within a system is only allowed if an element exists that produces outputs not based on the current inputs. In the case of the synchronous data flow modeling domain, an element such as delay element must exist within the feedback loop to allow for proper scheduling of the model. Feedback is checked for each element within the model. For each element, the model works forward through the outputs to determine if a loop exists back to the elements input without an element that supports feedback. Once an element that supports feedback is found, the test is complete and there is no need to recurse further on this path. Feedback correction can be enabled or disabled. Once feedback is found, the problem can be automatically corrected through the insertion of a delay element. Inclusion of the delay element implies the model has some state-saving features. State saving between input data elements is not appropriate for many problems including the census problem presented in Section 7.3. Thus if delay is not desired, the problem is corrected by disabling the link causing feedback.

After the model has been traversed, and none of the error correction techniques find any more errors, then the MoML description of the model is output. The MoML contains the elements required and the specified attributes along with Ptolemy II *relations*, which link the ports together in the specified topology.

7.2.3 Multiple Objective Evolutionary Algorithm Software

At first, this GP task may appear dissimilar to the MOEA efforts of the remainder of this thesis. But, GP requires a large quantity of *training* data to properly train the filter. For example, recall from Section 7.1.5.1 the 5000 false positive (FP) and 50 true positive (TP) images required by Howard [31] for the SAR application. By dividing sets of training data into objectives, the objective ordering system will find those sets of training data that quickly span the feature space. The remaining training data that provides little additional

information will be placed at the end of the objective list.

The power of GP is hypothesized to come from its ability to try thousands of *ideas* that would be prematurely dismissed by human designers. Thus, this chapter not only creates a new method for GP, but also utilizes new MOEA methods to treat sets of training data as objectives to quickly identify and eliminate *bad* ideas proposed by the evolutionary process.

7.3 Census Pattern Recognition Problem

The data for this problem originates from the 1994 census [32]. The task is to determine whether a person makes more or less than $\$50,000$ per year based on a set of other attributes. There are 14 attributes for the data; 6 are continuous values and the remaining 8 are enumerated. Table 20 details the types and values of all attributes.

A total of total $48,842$ instances were extracted from the census data using the following conditions:

$$((AGE > 16)\&\&(GI > 100)\&\&(FNLWGT > 1)\&\&(HRS\,WK > 0)). \tag{158}$$

where AGE and FNLWGT are described in Table 20, and GI is the gross income. Of these instances $45,222$ did not contain unknown values. The remaining values were randomly split into training and evaluation sets using the package MLC++ [1] where two thirds were selected randomly for training $(30,162)$, and one third for evaluation $(15,060)$. The training data has 24.78% of individuals earning an income above $\$50,000$, and 75.22% of the individuals earning less than or equal to $\$50,000$. Results, presented in Figure 21, were created using MLC++ for fifteen machine learning techniques. All training data, evaluation data and the results are available from Barry Becker[33].

For this problem, define the following:

- A true positive, TP, event is a correct prediction of an individual earning $> \$50,000$.
- A true negative, TN, event is a correct prediction of an individual earning $<= \$50,000$.

- A false positive, FP, event is an incorrect prediction of an individual earning > $50,000.
- A false negative, FN, event is an incorrect prediction of an individual earning <= $50,000.

The error rates indicated in Table 21 are simply the sum of the FP and FN instances, divided by the total number of data points. Based on the distribution of data an estimate of always negative would result in a 24.78% error rate.

Table 20: Attributes For Census Pattern Recognition Problem

Attribute	Type	Values
age	integer	Greater than zero
workclass	8 enum.	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked
fnlwgt	integer	Greater than zero
education	16 enum.	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool
education-num	integer	non-negative
marital-status	7 enum.	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse
occupation	14 enum.	Tech-support, Craft-repair, Other-service, Sales Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces
relationship	6 enum.	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried
race	5 enum.	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black
sex	2 enum.	Female, Male
capital-gain	$’s	non-negative
capital-loss	$’s	non-negative
hours-per-week	integer	non-negative
native-country	41 enum.	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holland-Netherlands

Table 21: Error Rates for Census Pattern Recognition Problem for Various Algorithms

Algorithm	Error Rate (%)
FSS Naive Bayes	14.05
NBTree	14.10
IDTM (Decision table)	14.46
C4.5-auto	14.46
HOODG	14.82
C4.5 rules	14.94
OC1	15.04
C4.5	15.54
Voted ID3 (0.6)	15.64
CN2	16.00
Naive-Bayes	16.12
Voted ID3 (0.8)	16.47
T2	16.84
1R	19.54
Nearest-neighbor (3)	20.35
Nearest-neighbor (1)	21.42

7.4 Weighted-Sum Optimization of Adult Census Pattern Recognition Problem

For the weighted-sum and block-diagram GP method, the census problem is recast into a multiple objective problem by trying to increase the number of true positives and decrease the number of false positives. To allow the ICEO techniques to eliminate poor performing individuals quickly, these objectives are duplicated for two sets of data. The first $5,000$ data points of data are utilized to extract the first two objectives. These $5,000$ data points define the values for f_1 and f_2.

$$f_1 = \frac{TP_1}{TP_1 + FN_1} \tag{159}$$

$$f_2 = 1.0 - \frac{FP_1}{FP_1 + TN_1}, \tag{160}$$

where TP_1, TN_1, FP_1, and FN_1 results from the first set of objective data.

The remaining $26,162$ items are then utilized if the thresholds are met for the individual under test. These define the third and fourth objectives defined as:

$$f_3 = \frac{TP_2}{TP_2 + FN_2} \tag{161}$$

$$f_4 = 1.0 - \frac{FP_2}{FP_2 + TN_2}, \tag{162}$$

where TP_2, TN_2, FP_2, and FN_2 results from the first set of objective data.

To allow comparison with previous machine learning techniques, the error rate can also be extracted from these measures of performance by scaling the results based on the distribution of the input data. Let $N_1 = 5,000$ and $N_2 = 25,162$ be the number of data points in each grouping of data. Let $N = N_1 + N_2 = 30,162$ by the total number of data points. Let $T = 7,508$ and $F = 22,654$ be number of true and false points, respectively, in the total distribution of data. The the error rate is:

$$\begin{aligned} E &= 1.0 - \frac{(T/N)(N_1 f_1 + N_2 f_3)}{N} - \frac{(F/N)(N_1 f_2 + N_2 f_4)}{N} \\ &= 1.0 - \frac{T(N_1 f_1 + N_2 f_3) - F(N_1 f_2 + N_2 f_4)}{N^2} \\ &= 1.0 - \frac{7,508(5,000 f_1 + 25,162 f_3) + 22,654(5,000 f_2 + 26,162 f_4)}{30,162^2}. \end{aligned} \tag{163}$$

The four dimensional objective space can also be collapsed to a two dimensional objective space based on the performance for true positives and false positives and the distribution of each of these in the input data.

$$\begin{aligned} P_t p &= \frac{(N_1 f_1 + N_2 f_3)}{N} \\ P_f p &= \frac{(N_1 f_2 + N_2 f_4)}{N} \end{aligned}$$

The purpose of this section is to provide a comparison of the ability to evolve a solution with a method that is a subset of the GP topology. To support interfacing to the weighted-sum, the enumerated attributes defined in Table 20 were recast into a number of binary attributes equal to the number of enumerations. This recasting expanded the number of attributes for each data point from 14 to 105.

The search space for the weighted-sum problem is the 105 weights for each of these attributes. These weights are constrained to a range $[-1.0, 1.0]$. To support a standard range for all weights, the values for the data points are also recast from their original input range to the range [0.0,1.0]. The decision function is:

$$D = \begin{cases} 0 & if \sum_{i=1}^{105} W_i D_i \leq 0 \\ 1 & otherwise \end{cases} \tag{164}$$

where W_i are the 105 weights of the search space, and D_i are the 105 attributes values of the data point.

7.4.1 Weighted-Sum Census Optimization Setup

The fitness function utilizes the Pareto rank 1 "raw fitness" calculation, as described in Section 2.2.4.1 followed by four fitness scalers. The first fitness scaler is the SPEA2 density scaler. The second fitness scaler is the Linear Interpolated Data Transformation 1 utilized in Section 4.1.2.1, which contained samples of the function

$$T_1(x) = \frac{1}{2^{(x-1)}}.$$

Table 7 itemizes and Figure 43 illustrates the data used for the LID transformations. The third fitness scaler is the fitness-sharing algorithm of the NPGA as described in Section 2.2.4.2. This fitness-sharing algorithm is initialized for one-hundred niches. The fourth and final fitness scaler is the HCD Scaler as detailed in Section 3.3. For the HCD scaler, the Holder coefficient is set to two for Euclidean distance, $p = 2$, and the offset is set one fifth, $\alpha = 0.2$.

Selection is performed using fitness-proportionate selection. Crossover is performed using the clone operator for twenty percent of the individuals. The remaining eighty percent of the individuals use the location crossover, with crossover allowed at any of the weight boundaries.

As described in Section 3.1.3, the *GTMOEA* software supports the assignment of the type of mutation to any named attribute in the search space. Each weight is assigned a normal distribution with a standard deviation equal to one at a rate of one percent. Each weight is also mutated with a uniform distribution at a rate of one percent. Therefore, the total probability of mutation of any part of an individual is:

$$P_m = 100\left(1.0 - \left((1.0 - 0.01)^2\right)^{105}\right) = 100\left(1.0 - (1.0 - 0.01)^{210}\right) = 87.88\%. \qquad (165)$$

Dynamic objective threshold is performed with lower limits of $\boldsymbol{S} = \{0.1, 0.1, 0.2, 0.6\}$, and the upper limits, defining the region of interest, of $\boldsymbol{L} = \{0.6, 1.0, 0.6, 1.0\}$. The weights for each objective are $\boldsymbol{W} = \{0.1, 0.1, 0.3, 0.3\}$. Note the increased difficulty applied from both the lower limits, and weights for the second set of objectives. Equation 163 reveals the increased dependence of the final measure of performance on the second and fourth objectives versus the first and third objectives. Therefore, the region of interest is moved from the expected $\{1.0, 1.0, 1.0, 1.0\}$ to $\boldsymbol{L}$ to accentuate these more important objectives.

A set of three rules is used to prune the size of the elite pool, E. The first rule provides phenotypic diversity by ensuring that only a single individual with identical objectives values is used. The second rule ensures that the individuals are Pareto optimal by removing dominated individuals. The final rule ensured that only 750 individuals reside in the elite

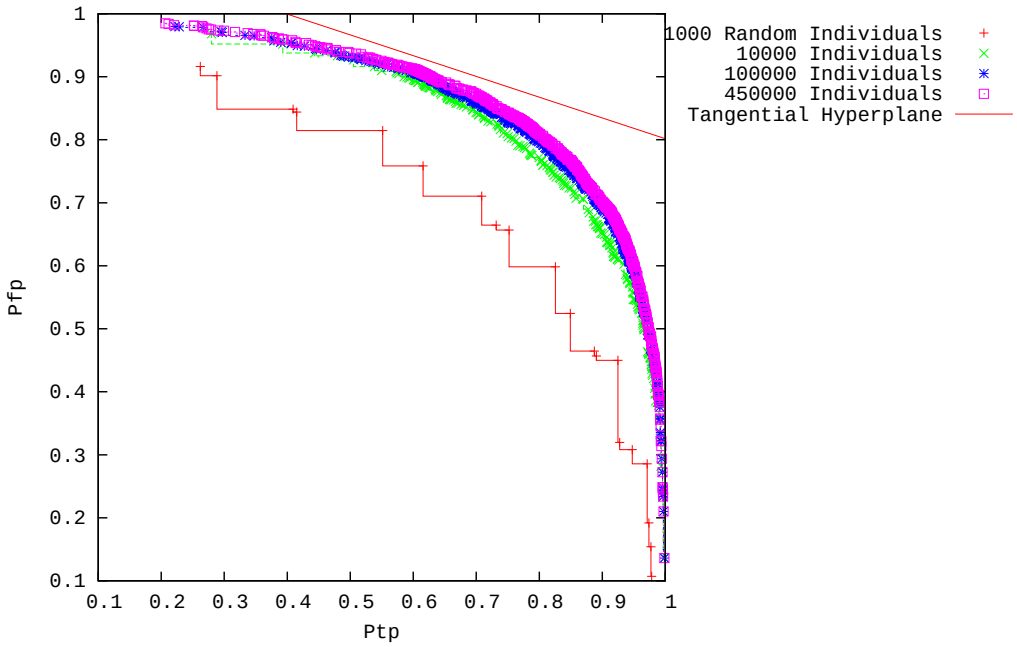

Figure 125: Pareto Fronts During Training of Weighted-Sum for Census Problem

pool, $|E| \leq 750$. If over 750 individuals reside in the elite pool after the application of the previous rules, only the 750 individuals with the best fitness values are retained in the elite pool. A more detailed description of these elitism rules can be found in Section 2.2.

Optimization is initialized with 1,000 random individuals. To prevent the wasting processor time for evaluation of identical individuals, children are compared to the parents to insure they are not identical. One hundred individuals are generated for each generation, i.e., $|C| = 100.0$.

7.4.2 Weighted-Sum Census Results

A total of $450,000$ individuals are evaluated for the optimization with the weighted-sum. Figure 125 displays the performance of the Pareto Optimal solutions at various points during the optimization. This figure also displays the tangential hyperplane, which is defined by the distribution of true and false instances in the training data, e.g. $(0.2478, 0.7522)$.

Two measures of performance are used to visualize the changes in performance of the

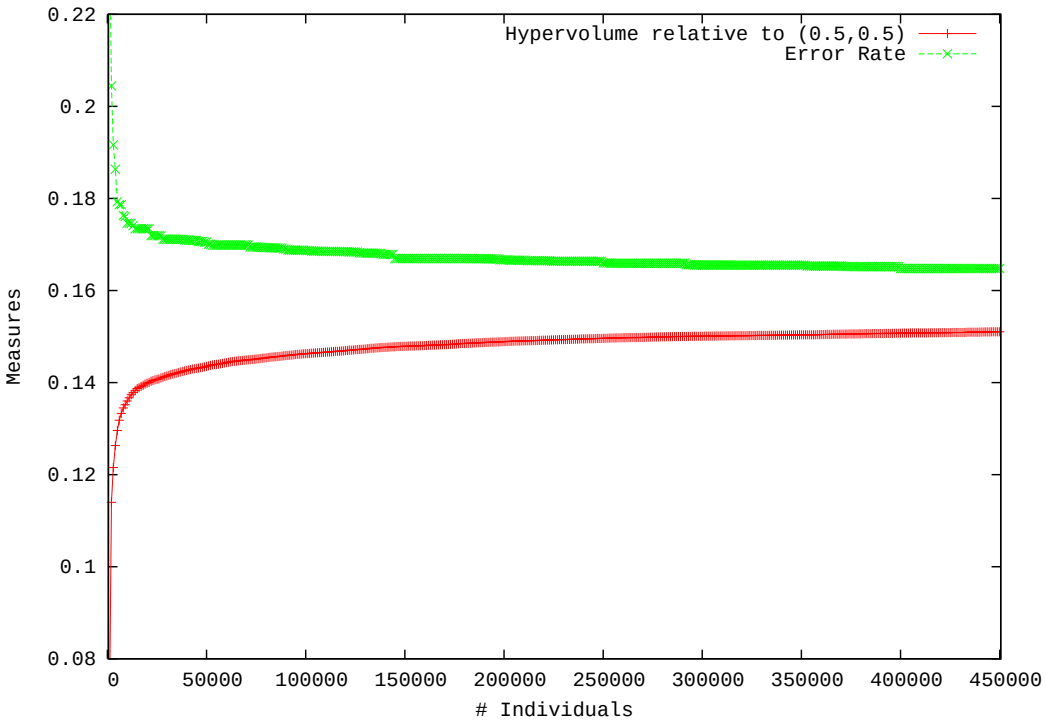

Figure 126: Measures of Performance During Training of Weighted-Sum for Census Problem. Hypervolume calculated using Equation 3. Error rate calculated using Equation 163.

resulting population. First, the final measure given in Equation 163 is the error rate to be reduced. This measure is reduced as individuals move closer to the tangential hyperplane as illustrated in Figure 125. The error rate is dependent on the performance of a single individual, and thus does not indicate improvements that may be occurring on other sections of the Pareto front. The second measure of performance is the hypervolume of solutions relative to the targeted area defined by the point $(0.5, 0.5)$. This measure gives an indication of the progress of the population, and is less dependent on a single individual. Figure 126 displays the performance of these two measures over the evaluation of the $450,000$ individuals. There is very little change in either measure by the end of the $450,000$ individuals.

Figure 127 displays the resulting elite individuals in the two-dimensional objective space and the value of the resulting final measure for both the training and evaluation data. Note that the evaluation results are worse than those from training. The best measure achieved during training is 16.49%. The best measure achieved during evaluation is

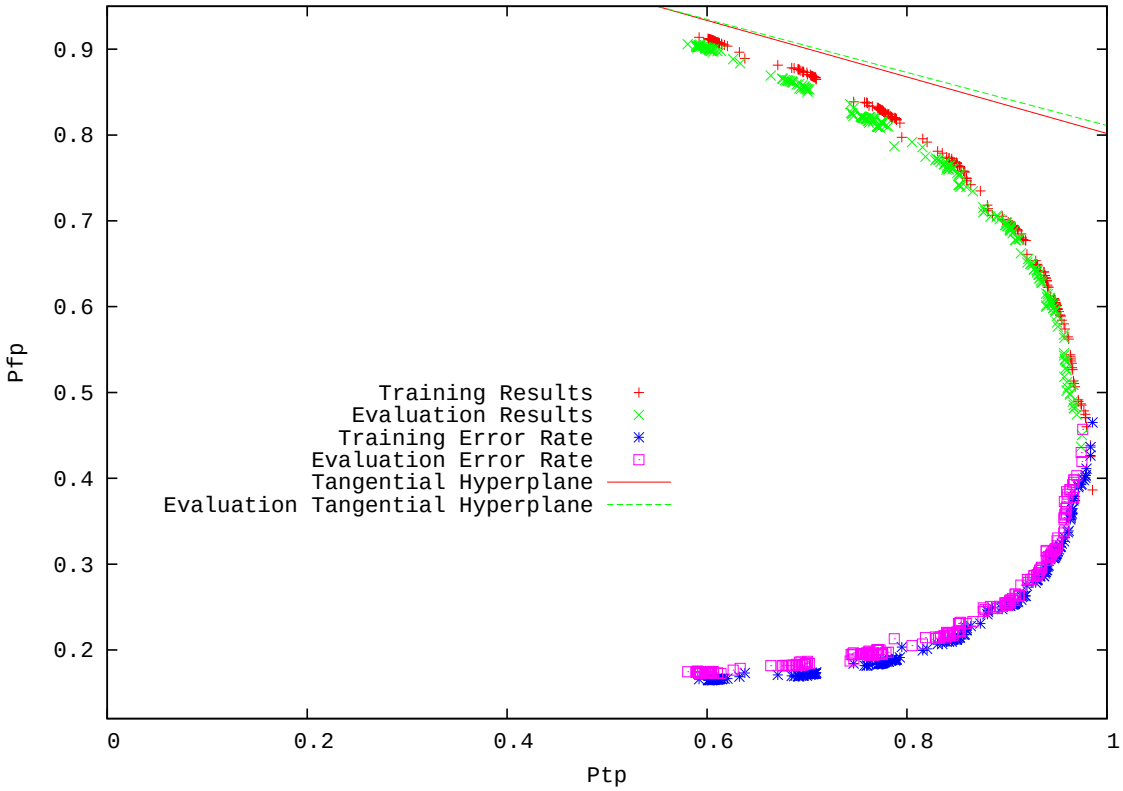

Figure 127: Comparison of Training and Evaluation Pareto Fronts and the Training and Evaluation Error Rates as defined in Equation 163 After 450,000 Individuals For The Weighted-Sum Census Problem

17.21%. From Table 21 the final evaluation performance outperforms only three of sixteen machine learning algorithms: 1R, Nearest-neighbor (3), and Nearest-neighbor (1).

7.5 Evolutionary Programming Optimization of Adult Census Pattern Recognition Problem

Given the prior results of others listed in Table 21 and the results of the weighted-sum method itemized in the previous section, this section gives an example of an optimization of the same problem using the block diagram innovations of this chapter. Three runs are examined.

1. One that is initialized with individuals with random genome description resulting in random attribute and topology selections

2. The second that is initialized with descriptions that mimic elite weighted-sum individuals from the weighted-sum after the evaluation of $50,000$ individuals with a cascading set of adders.

3. The third that is initialized with individuals that also mimic the weighted-sum results, but are implemented using a hierarchy of adders.

7.5.1 Search Space

The search space for all three problems is equivalent. As shown in Table 22, five functions are allowed to evolve. The first function implements a set of math functions that operate on a single input. The remaining four functions receive multiple inputs and produce a single output.

A math function has a total of seven attributes set by the genome: the function type, two parameters values (p_1, p_2), and four parameters that perform offset and scaling (m_1, b_1, m_2, b_2). The math function first transforms the input value x using $m_2x + b_2$. The output of a function, f, is transformed similarly, resulting in $g(x) = m_1 f(m_2x + b_2) + b_1$. Some functions require several optional parameters, and thus the generic description becomes $g(x, p_1, p_2) = m_1 f(m_2x + b_2, p_1, p_2) + b_1$. The function $f(x)$ is also specified by the genome, and allowed to be any of the twenty-five functions listed in Table 22. A constant function is implemented with $f(x) = 0$, which results in the constant $g(x, p_1, p_2) = m_1b_2 + b_1$. Likewise a scaler function is implemented with $f(x) = x$, which results in $g(x, p_1, p_2) = m_1m_2x + m_1b_2 + b_1$. A clipping function requires both arguments, and is implemented with $f(x) = min(max(x, p_1), p_2)$.

The remaining four functions are the *Add/Subtract*, *Multiply*, *Maximum* and *Minimum* functions. The *Add/Subtract* function receives up to five inputs on each of the plus and minus ports creating a single output value. The *Multiply* function finds the product of up

to five inputs on the input port. This actor supports a division port as well, but it is not enabled in order to prevent division by zero errors. The *Maximum* and *Minimum* functions also accept up to five inputs and output the maximum or minimum value respectively on the output port.

Table 22: Basis Functions for Evolutionary Programming Optimization

Function	Attributes	Inputs	Output
Math Function	$f(x, p_1, p_2) = \{atan(x), asinh(x), cos(x), cosh(x), sin(x), sinh(x), tan(x), tanh(x), abs(x), ceil(x), compare(x, p_1), exp(x), floor(x), log(x), log10(x), log2(x), max(x, p_1), min(x, p_1), pow(x, abs(p_1)), sgn(x), sqrt(x), sinc(x), 0.0, x, min(max(x, p_1), p_2)\}$, $m_1, m_2, b_1, b_2, p_1, p_2$	x	$m_1 f(m_2 x + b_2, p_1, p_2) + b_1$
Add/ Subtract	None	$\{p_1, \cdots, p_5\}$ $\{m_1, \cdots, m_5\}$	$\sum_{k=1}^{5} p_k - \sum_{k=1}^{5} m_k$
Multiply	None	$\{x_1, \cdots, x_5\}$	$\prod_{k=1}^{5} x_k$
Maximum	None	$\{x_1, \cdots, x_5\}$	$\max_{k=1}^{5} x_k$
Minimum	None	$\{x_1, \cdots, x_5\}$	$\min_{k=1}^{5} x_k$

The search space enables each block to be any of the five function blocks described in Table 22. A maximum of 300 processing blocks are utilized in the processing. The *Add/ Subtract* function has ten attributes describing the input connectivity. The *Multiply, Maximum,* and *Minimum* functions require five attributes to describe input connectivity. The *Math Function* has one attribute for input connective, one attribute to select the function and the six attributes $m_1, m_2, b_1, b_2, p_1, p_2$ for each function. There is also one attribute for each function block determining which type of function block is expressed. This implies each block requires $10+3*5+8+1 = 34$ attributes. Therefore, there are $34*300 = 10,200$ total search space dimensions. At any given time only $300 * (Max(10, 5, 8) + 1) = 3,300$ dimensions are expressed.

7.5.2 Setup of Runs

Each of the runs is initialized with the same input as the weighted-sum method given in Section 7.4.1 with the following exceptions, mostly due to differences in the genome description. The location crossover operator is initialized to allow crossover at the boundaries of any of the five function description boundaries.

Mutation is applied using three separate mutation operators. First, uniform mutation is applied to the each of the 300 attributes that define which of the five basis functions is expressed. Second, uniform mutation is applied to each of the attributes values, i.e. $p_1, p_2, m_1, b_1, m_2, b_2$, the type of math function used, and the ports connected to the inputs. Third, normal mutation is also applied to this same list of attribute values, i.e., $p_1, p_2, m_1, b_1, m_2, b_2$, the type of math function used, and the ports connected to the inputs. The rates of mutation are different for each experiment. The random initialized experiment uses a mutation rate of one percent, the cascade initialized experiment uses a mutation rate of 0.5 percent, and the hierarchical method uses a mutation rate of 0.05 percent for each of the three methods. A discussion of the effects of mutation is given in Sections 7.5.5 and 7.5.6 to provide insight into the selection of these values.

7.5.3 Random Initialized Results

The first run is initialized with purely random values for each part of the genome. A total of $3,000$ random individuals is used to start the optimization. This run is designed to test the ability to evolve with no "preconceived notions" of what the final system design should look like.

Figure 128 displays the two measures of performance during training for this run along with the results for weighted-sum optimization for comparison. From this figure, note that the method is very slow to evolve when compared with the results of the weighted-sum.

Figure 129 displays the resulting Pareto fronts at several locations through the optimization compared to the results of the weighted-sum optimization. These results also

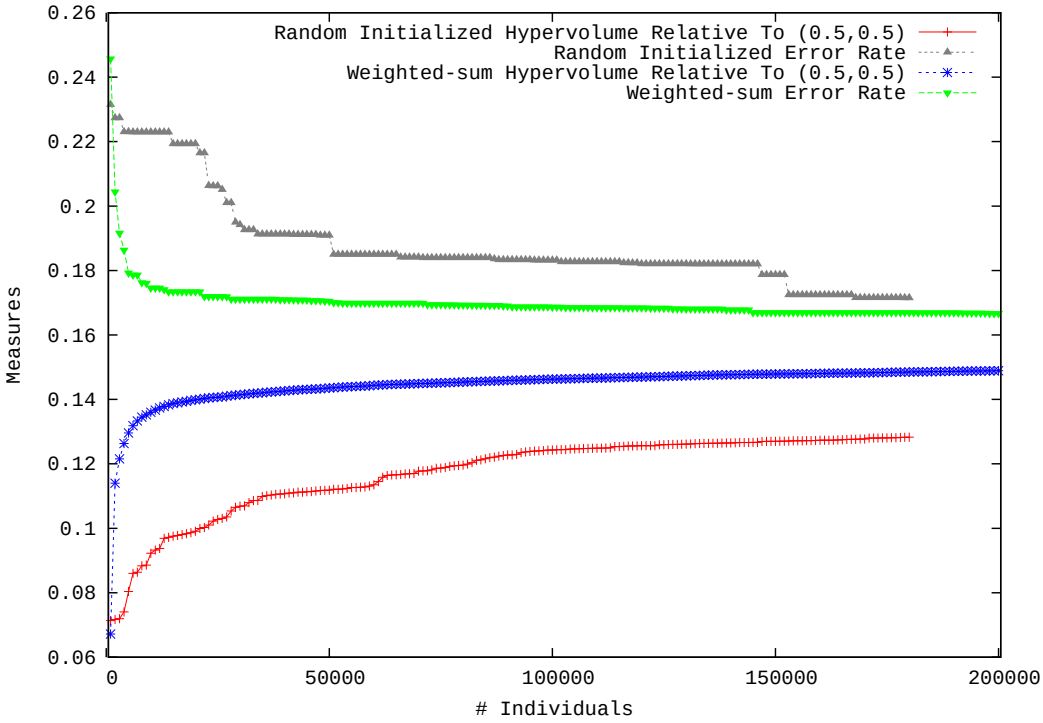

Figure 128: Measures of Performance During Training of Random Initialized Evolutionary Programming Compared to Weighted-Sum Results for Census Problem. Hypervolume calculated using Equation 3. Error rate calculated using Equation 163.

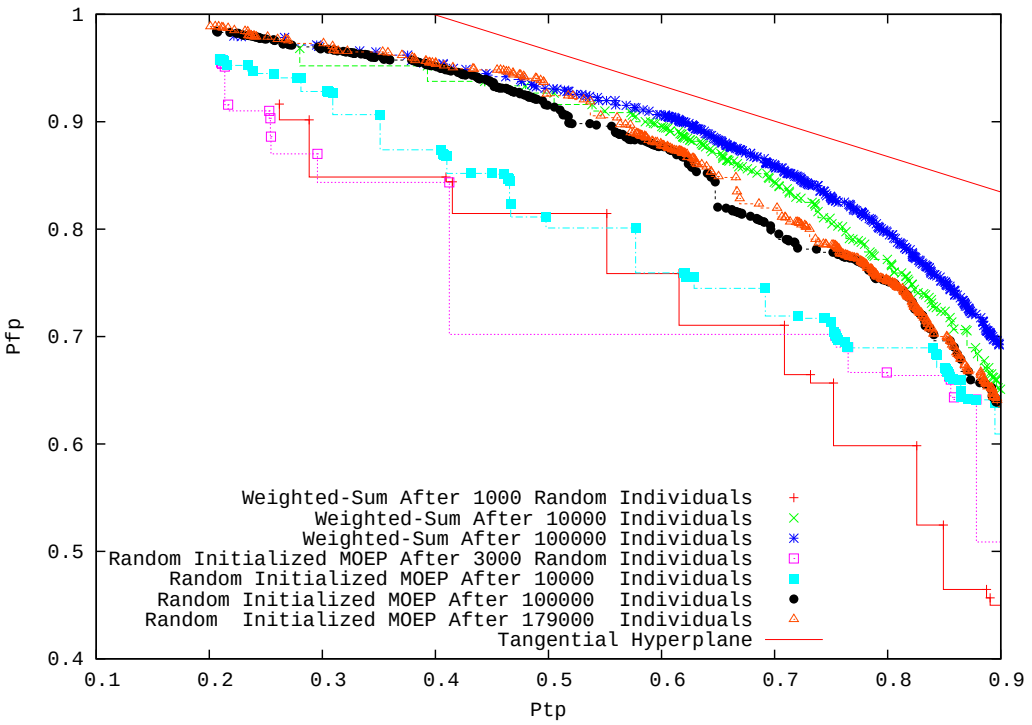

Figure 129: Pareto Fronts From Random Initialized Evolutionary Programming Compared To Weighted-Sum Results

demonstrate the inability to achieve results similar to the simple topology of the weighted-sum.

Figure 130 displays a comparison of the training and evaluation Pareto optimal front for the last available elite individuals, which occurred around 153, 000 individuals. For this figure individual 152, 777 shows the best performance with an error rate of 17.2 percent for training data and 16.68 percent for the evaluation data.

Figure 131 displays the topology for individual 152, 777. This individual utilizes a total of 102 processing blocks. Table 23 gives a summary of the number and types of processing blocks utilized. From this table, note that none of the *Math Functions* utilized the linear x function. Only 79 of the 105 possible input ports are utilized. There are a total of 474 inner-connections between all ports and elements. This large number of inner-connections, and the lack of regular patterns makes the interpretation of the block diagram very difficult.

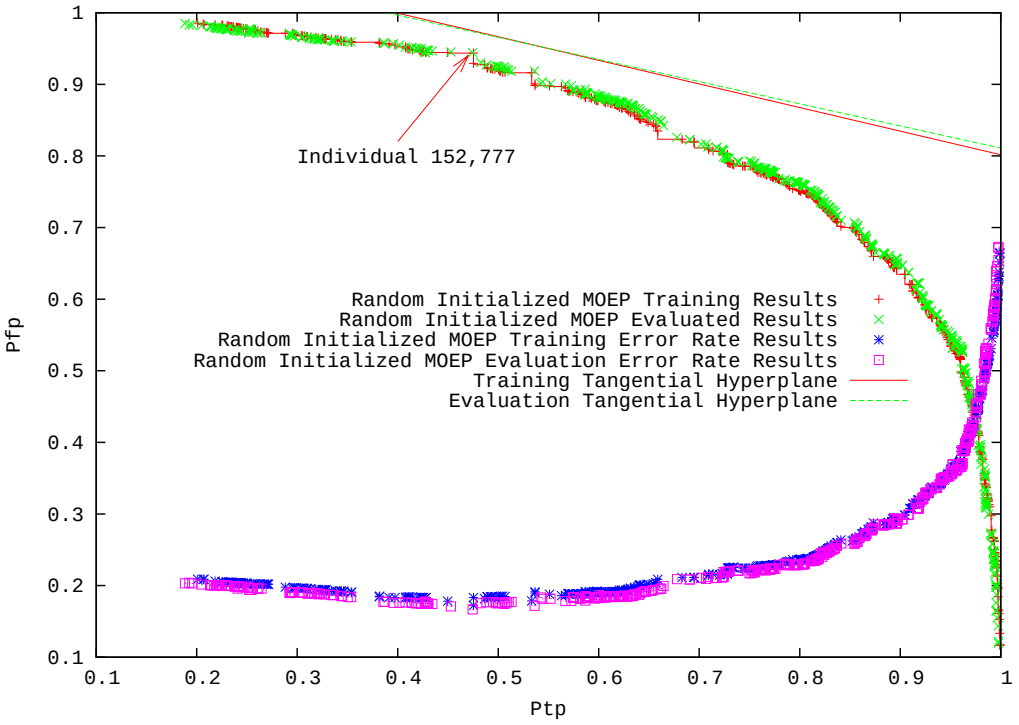

Figure 130: Comparison of Training and Evaluation Pareto Fronts and the Training and Evaluation Error Rates as defined in Equation 163 After 153, 000 Individuals For The Random Initialized Evolutionary Programming Technique

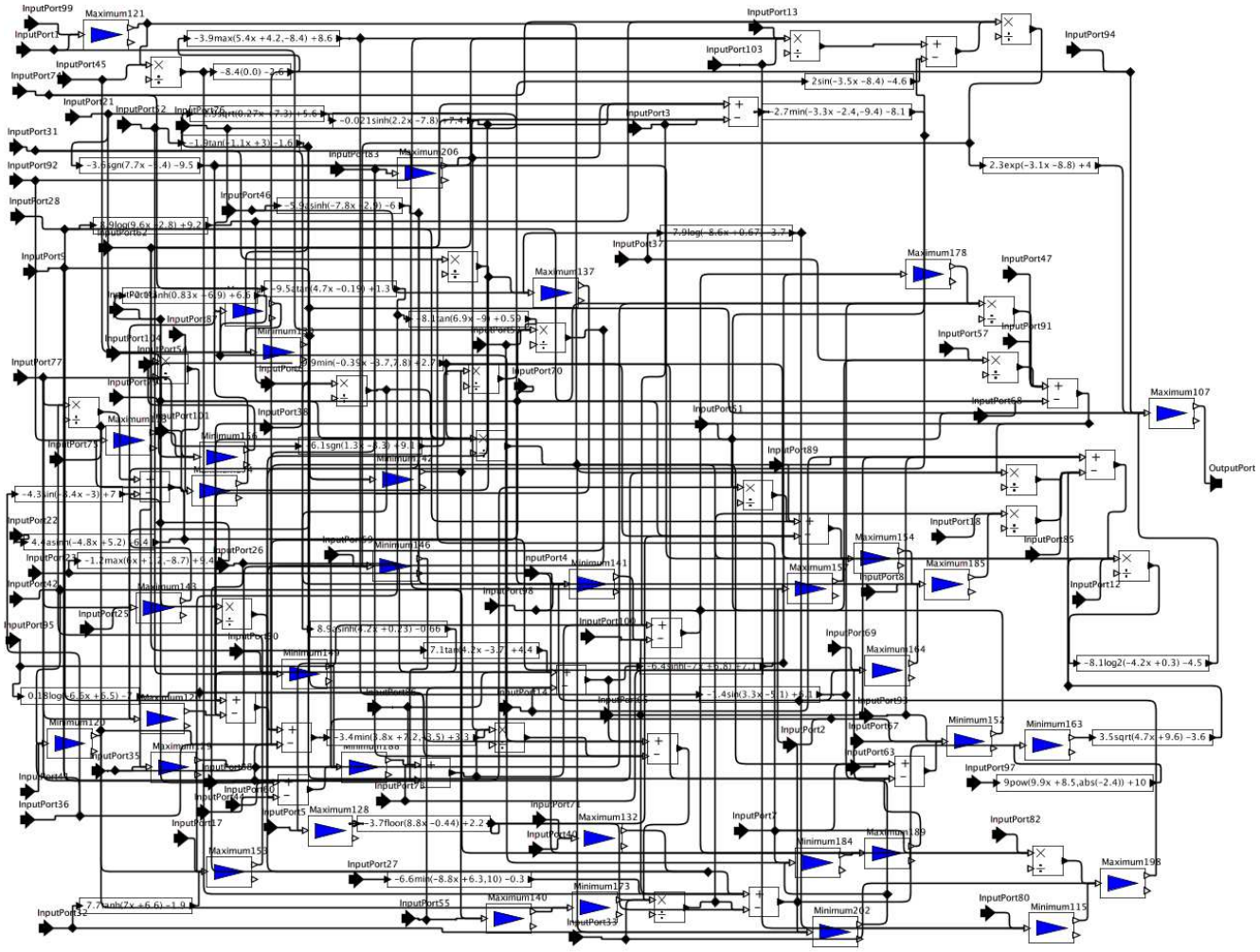

Figure 131: Individual 152, 777 From Random Initialized Evolutionary Programming. Error Rate of 17.15 Percent Training and 16.68 Percent Evaluation.

In summary, this experiment illustrates the capability of block diagram multiple objective evolutionary programming to find solutions to a pattern recognition problem. Unfortunately, evolution is much slower than that of a simple weighted-sum. The weighted-sum requires 6.09 seconds on average to evaluate an individual, compared with the average of 83.09 seconds for the random-initialized method evolutionary programming method. These extend run-times are due to the use of Ptolemy II, instead of a simply awk script. The random-initialized method is able to reach its best error rate of 17.15 percent with the evaluation of individual 178, 497, implying a total of 171.7 processing days. On the other hand, the weighted-sum method evaluates the same number of individuals in 12.6 processing days, and reaches the same error rate with individual 27, 905, which required only 2.0 processing days. *This implies that block diagram method is* 85 *times slower than that of a simple weighted-sum.* Section 7.6 will further discuss the run-time issues.

Table 23: Types of Functions Used for Individual 152, 777 of the Random Initialized Evolutionary Programming Method

Function	**Type**	**# Used**
Math Function		32
	asinh	3
	atan	1
	exp	1
	floor	1
	log	3
	log2	1
	max	2
	min	4
	sgn	2
	sin	3
	sinh	2
	sqrt	2
	tan	3
	tanh	2
	$pow(x, abs(p_1))$	1
	0.0	1
Add/Subtract	N/A	15
Multiply	N/A	20
Maximum	N/A	21
Minimum	N/A	15
Total	N/A	102

7.5.4 Cascaded Weighted-Sum Initialized Results

To eliminate the problems of the random initialized method in reaching the capabilities of the weighted-sum, this experiment is initialized with the 750 elite individuals resulting from the weighted-sum optimization after $50,000$ individuals. This initialization allows the faster running weighted-sum to be utilized to reach good performance. Then, the block diagram evolutionary programming method makes modifications in the topology and utilizes non-linear functions. The input descriptions are made using the scaling *Math Function* and *Add/Subtract* basis functions. The weights for each input are implemented with the scaling math function $g(x) = m_1(m_2x + b_2) + b_1$, with $m_2 = 1$ and $b_1 = b_2 = 0$ resulting in $g(x) = m_1x$. The sums are implemented with cascaded *Add/Subtract* basis function. Four plus and five minus ports are connected to scaling functions. The remaining plus port is connected to the output of the next Add/Subtract function block.

Figure 132 displays the resulting topology visualized using the Ptolemy II Vergil tool. This topology requires 12 *Add/Subtract* components, and 105 linear *Math Functions*. The average time for evaluating individuals increases from 84.9 seconds for the random individual to 119.8 seconds for this cascade topology.

Figure 133 displays the two measures of performance during training for this run along with the results for weighted-sum optimization for comparison. Note that the results for the cascade solution are shifted by $50,000$ individuals to indicate the initialization using the results from the weighted-sum optimization at this period. Note that even upon termination, the hypervolume shows a continued positive slope, indicating a population that is continuing to improve. Also note the rapid initial improvement in the measure as compared to that of the weighted-sum.

Examination of individual $29,076$ with an error rate of 15.41 percent training and 15.35 percent for evaluation in Figure 134 illustrates an example of the sources of this rapid initial improvement. From this figure note that the number of processing blocks increases from 117 to 130. The number of *Math Function* blocks increases from 105 to 112, but all but

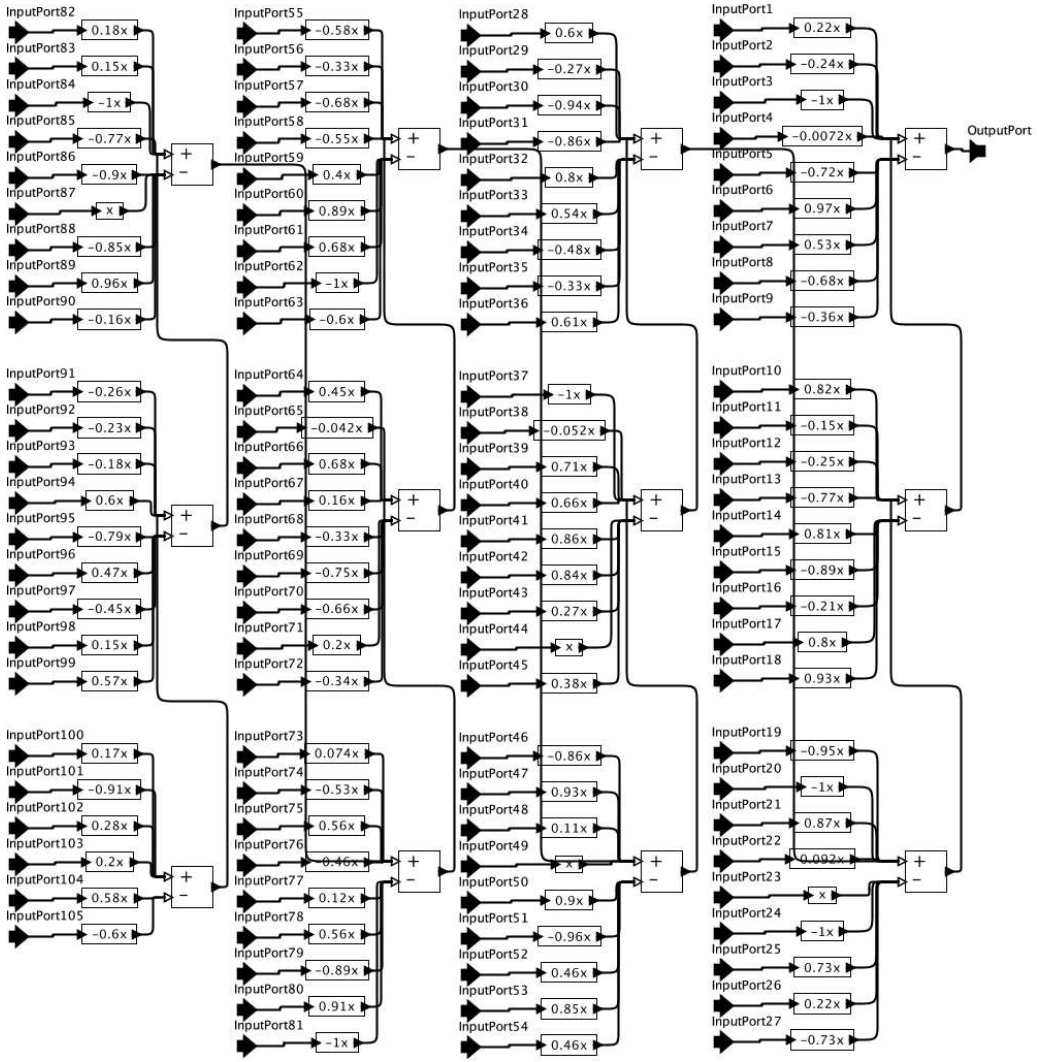

Figure 132: Topology for Individuals Initialized with Cascaded Weighted-Sum

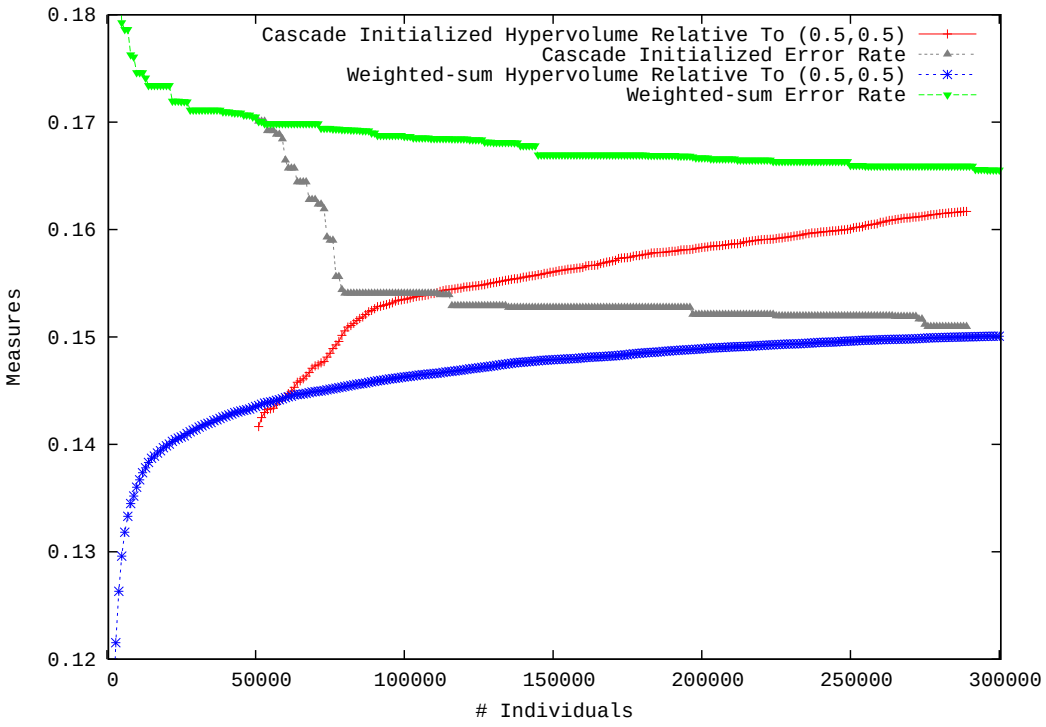

Figure 133: Measures of Performance During Training of Cascade Weighted-Sum Initialized Evolutionary Programming for Census Problem. Hypervolume calculated using Equation 3. Error rate calculated using Equation 163.

five of these functions are still linear functions. Input ports 45 and 65 are made useless via mutation to the function 0.0. Also added through mutation are the use of the atan, pow and sinc non-linear functions.

The overall topology remains much the same. The far left side of the figure displays some additional complexity connected to the final cascade *Add/Subtract Function* with the use of six additional *Add/Subtract* functions, and seven additional *Math Functions*. The complexity is probably attached to the final *Add/Subtract Function* because it is the only function block with additional ports available. Therefore, there is no loss in complexity to add additional elements at this location. In summary, individual 29, 079 illustrates that the rapid improvements are utilizing the ability to increase the complexity of the overall model through changes in topology and use of non-linear components.

Figure 135 displays a comparison of the training and evaluation Pareto optimal front for last available elite individuals, which occurred around 203, 000 individuals. Individual 174,557, with an error rate of 15.20 percent for training and 14.88 percent for evaluation, is illustrated in Figure 136. This individual reduces the total number of blocks utilized from 117 to 109. Much of the reduction occurs because of the loss of two of the twelve cascading *Add/Subtract Functions*, which results in the loss of the input ports, and associated connections for ports 82 to 99.

Of the 92 *Math Functions* utilized, 84 are linear functions. The non-linear functions used are floor, sin, sinc, clipping, asinh, tan and the zero function. The genetic dependence on individual 29, 079 can still be seen through the dependence of port 45 on the zero function. Also similar to individual 29, 079 is that much of the additional complexity is again added to the left-hand side of the diagram. Unfortunately, this displays a limitation on the original topology selected.

The cascade initialized evolutionary programming experiment illustrates two important features:

1. The ability to seed a block diagram with an existing set of Pareto optimal system

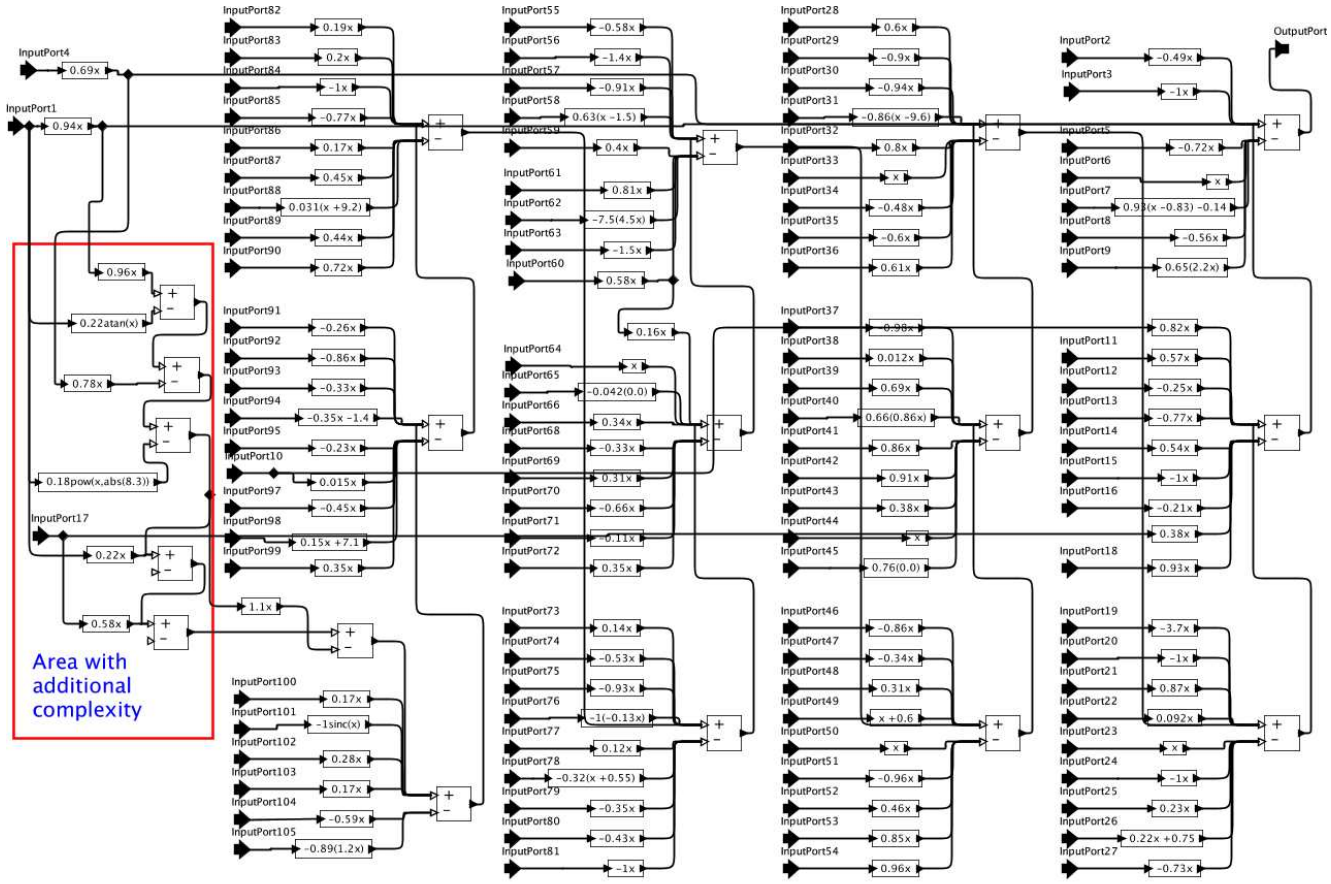

Figure 134: Individual 29,706 From Cascade Initialized Evolutionary Programming. Error Rate of 15.4 Percent.

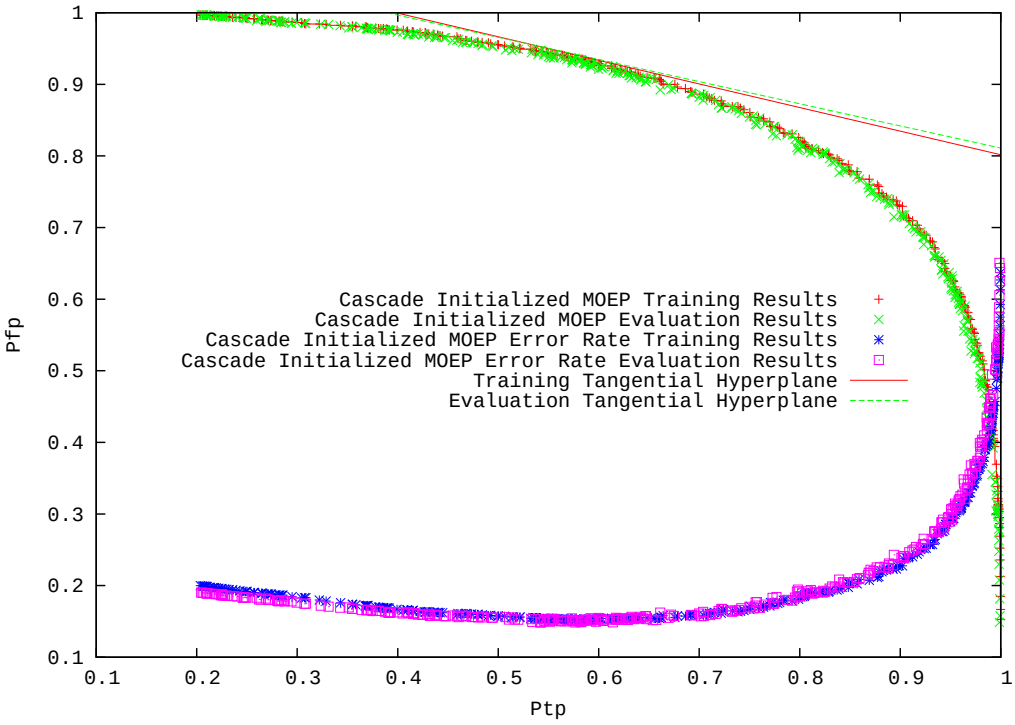

Figure 135: Comparison of Training and Evaluation Pareto Fronts and the Training and Evaluation Error Rates as defined in Equation 163 After 203, 000 Individuals For The Cascade Initialized Evolutionary Programming Technique

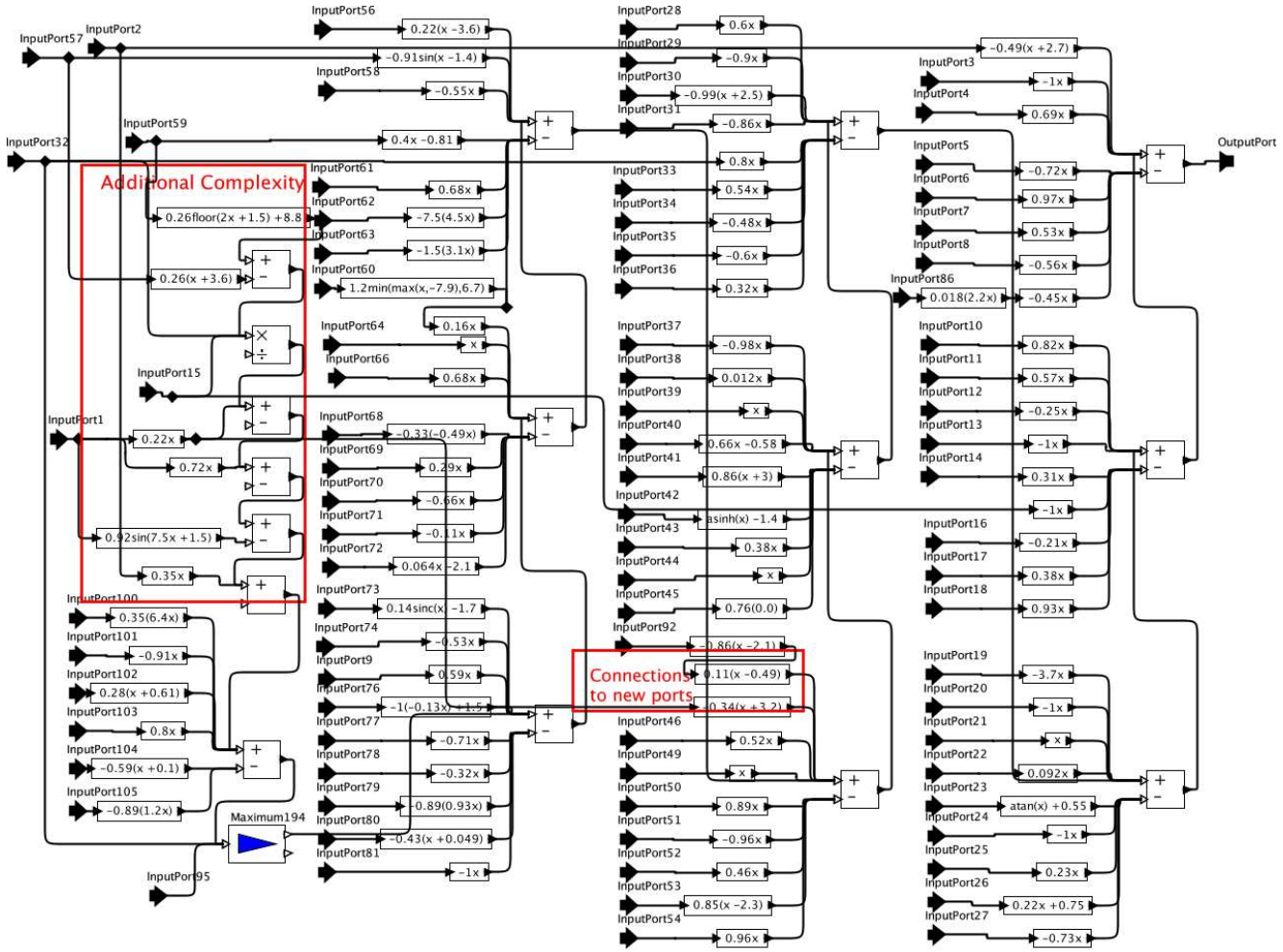

Figure 136: Individual 174,557 From Cascade Initialized Evolutionary Programming. Error Rate of 14.88 Percent.

designs.

2. The ability of the system to permute the design through changes in attributes and topology into a set of Pareto optimal systems with even better performance than the previous designs.

7.5.5 Hierarchical Weighted-Sum Initialized Results

The third experiment attempts to remove the dependence on the cascaded weighted-sum topology in several ways. First, a more hierarchical topology is created to remove the dependency of the first level of *Add/Subtract Functions* on the input of every subsequent input. Second, to allow insertion of additional components, the lowest level *Add/Subtract Functions* only receive input from three linear functions. Third, additional complexity between hierarchical components is seeded by using two cascaded linear functions.

Figure 137 displays a section of the topology. This method increases the number of components to 105 starting linear elements, 54 hierarchical *Add/Subtract Functions* and 108 linear elements between hierarchical *Add/Subtract Functions*. This results in a total of 267 components, which increases the average evaluation time from 119.8 seconds for the cascade design to 346.0 seconds. Again, the third run is initialized with 750 elite individuals resulting from the weighted-sum optimization after $50,000$ individuals.

An initial optimization resulted in very poor rate of performance improvements. This is hypothesized to originate from the increase in effective mutation due to the increased number of components. The cascade initialized method has twelve *Add/Subtract Functions* each with ten attributes, 105 linear elements each with eight attributes, and each element also has a function type attribute. With a 0.5 percent mutation rate for uniform and normal mutation, an individual has a probability of mutation of:

$$\begin{aligned} P_{m,cascade,individual} &= 1.0 - (1.0 - 0.005)^{2(12*11+105*9)} \\ P_{m,cascade,individual} &= 0.999998. \end{aligned} \tag{166}$$

But, the probability of mutating the hierarchical components is only dependent on the *Add/Subtract Functions*, resulting in:

$$\begin{aligned} P_{m,cascade,hierarchy} &= 1.0 - (1.0 - 0.005)^{2(12*11)} \\ P_{m,cascade,hierarchy} &= 0.733748. \end{aligned} \tag{167}$$

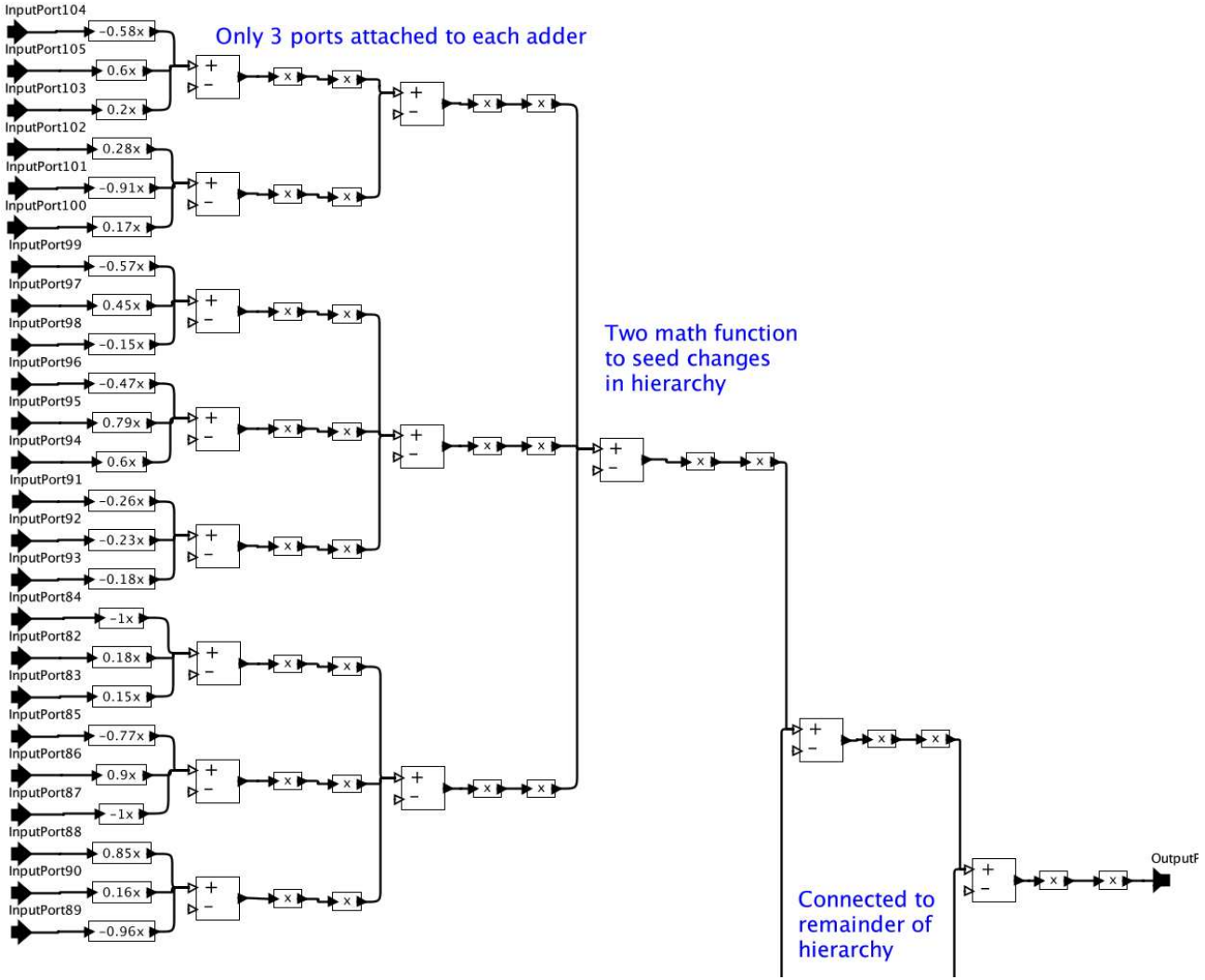

Figure 137: Topology for Individuals Initialized with Hierarchical Weighted-Sum

The hierarchical method uses 54 *Add/Subtract Functions*, 108 linear elements, resulting in a probability of mutating the hierarchical components of:

$$\begin{aligned} P_{m,hierarchical,hierarchy,original} &= 1.0 - (1.0 - 0.005)^{2(54*11+108*9)} \\ P_{m,hierarchical,hierarchy,original} &= 0.99999999. \end{aligned} \tag{168}$$

To reduce the deleterious effect of this high level of mutation, the probability of mutation for normal and uniform mutation is reduced by a factor of ten from 0.5 percent to 0.05 percent, resulting in:

$$\begin{aligned} P_{m,hierarchical,hierarchy,final} &= 1.0 - (1.0 - 0.0005)^{2(54*11+108*9)} \\ P_{m,hierarchical,hierarchy,final} &= 0.791203. \end{aligned} \tag{169}$$

This does leave a disparity in the mutation rate for the 105 input *Math Functions*, which cannot currently be rectified. The mutation rate for the input functions reduces from:

$$\begin{aligned} P_{m,cascade,input} &= 1.0 - (1.0 - 0.005)^{2(105*9)} \\ P_{m,cascade,input} &= 0.999941 \end{aligned} \tag{170}$$

for the cascade initialized MOEP experiment to

$$\begin{aligned} P_{m,hierarchical,input} &= 1.0 - (1.0 - 0.0005)^{2(105*9)} \\ P_{m,hierarchical,input} &= 0.621766 \end{aligned} \tag{171}$$

for the hierarchical initialized MOEP experiment.

Figure 139 displays a comparison of the training and evaluation Pareto optimal front for last available elite individuals, which occurred at 111, 750 individuals.

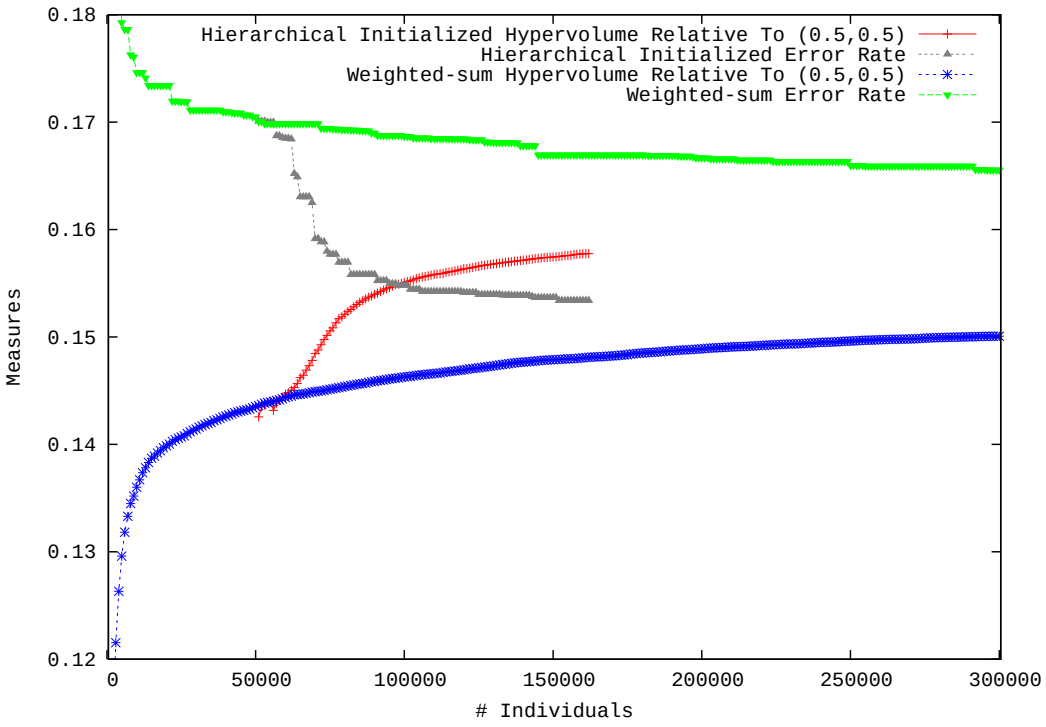

Figure 138: Measures of Performance During Training of Hierarchical Weighted-Sum Initialized Evolutionary Programming for Census Problem. Hypervolume calculated using Equation 3. Error rate calculated using Equation 163.

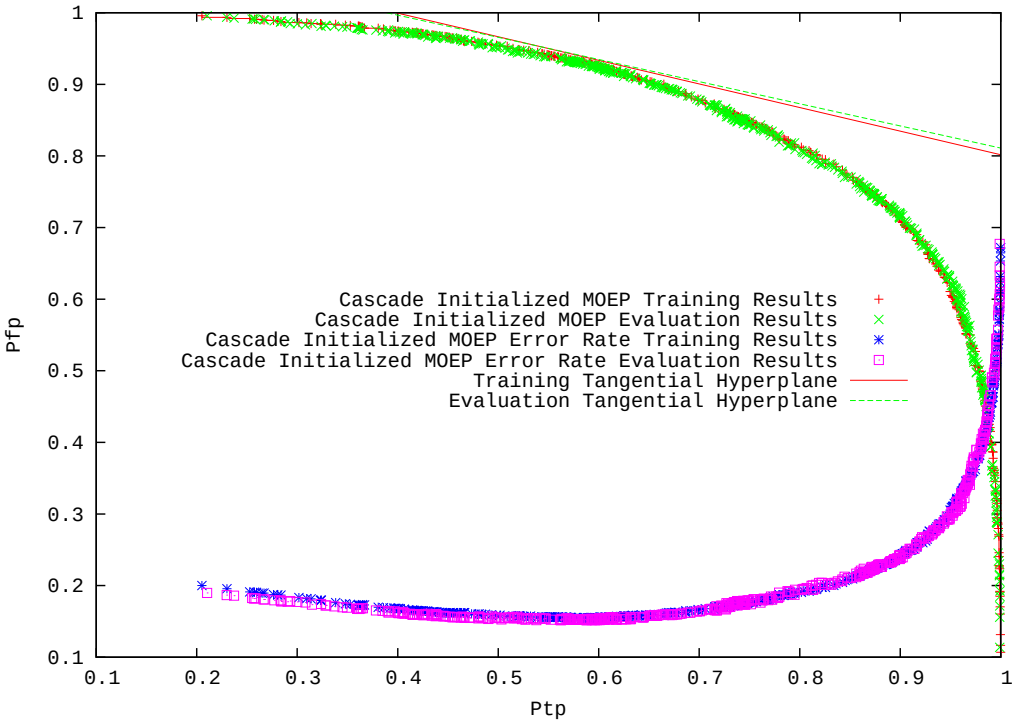

Figure 139: Comparison of Training and Evaluation Pareto Fronts and the Training and Evaluation Error Rates as defined in Equation 163 After 80,000 Individuals For The Hierarchy Initialized Evolutionary Programming Technique

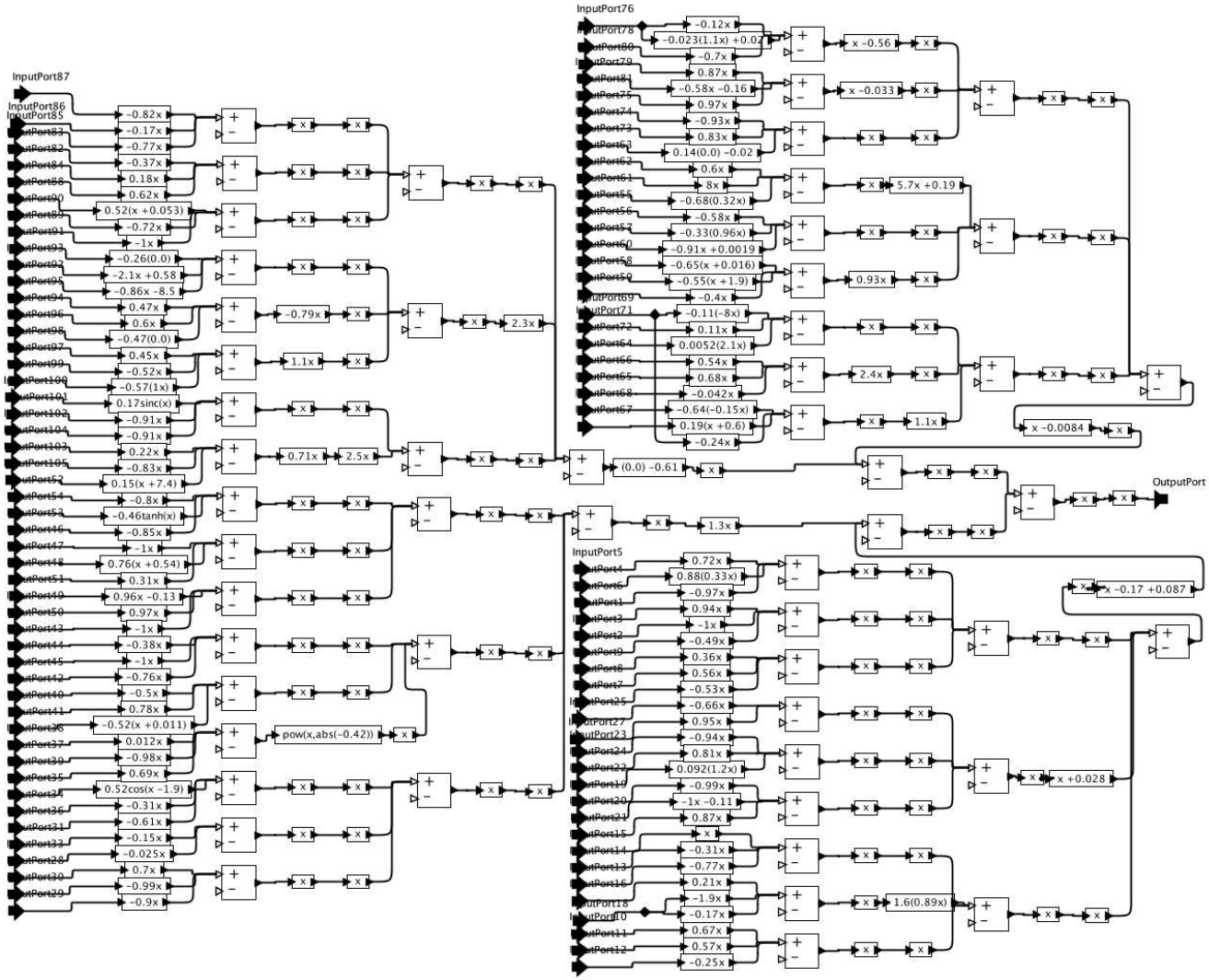

Figure 140: Individual 93, 348 From Hierarchy Initialized Evolutionary Programming. Error Rate of 15.37 Percent Training and 15.23 Percent Evaluation.

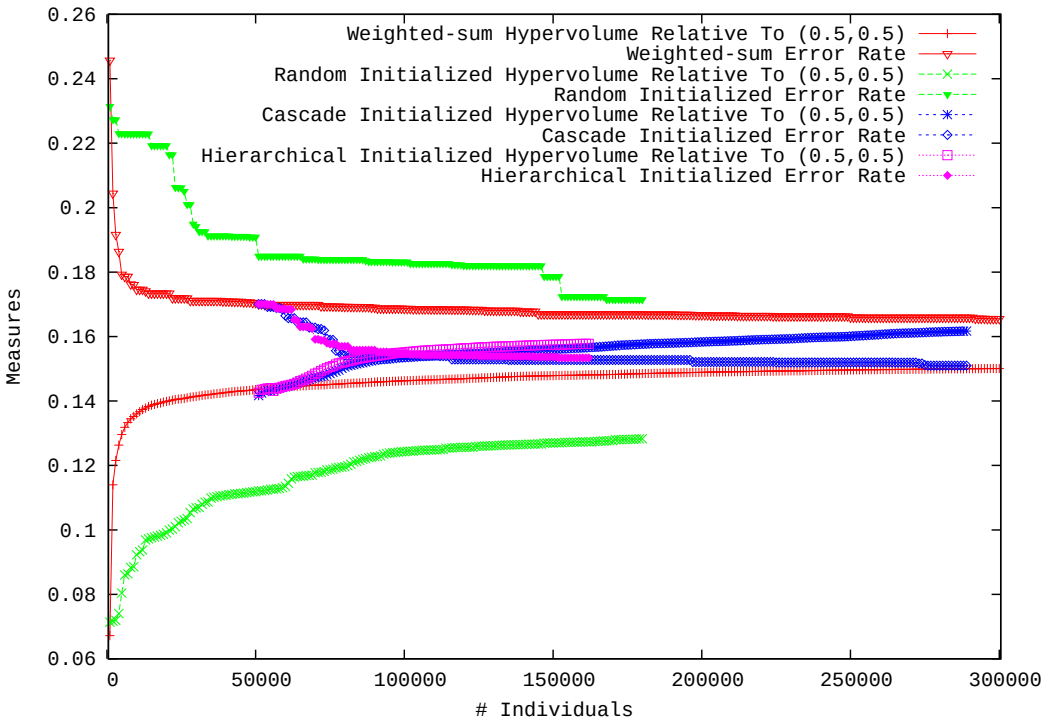

Figure 141: Evolution of Measures of Performance for Four Examined Methods for Census Problem. Hypervolume calculated using Equation 3. Error rate calculated using Equation 163.

7.5.6 Comparison of Census Problem Results

This section offers a comparison of the results of the optimization using the weighted sum, random initialized MOEP, cascade initialized MOEP, and hierarchical initialized MOEP methods. The two performance measures for the problem are the hypervolume in the desired area of solutions greater than $(0.5, 0.5)$ and the error rate. Figure 141 illustrates these two performance measures versus the number of individuals evaluated. Note that the cascade initialized MOEP method reaches the best performance measure after the solution of over $100,000$ individuals beyond the $50,000$ required to seed the execution.

The specifics of implementing each of the four methods affect the complexity of the solutions, which in turn affect the resulting run time. Figure 142 gives the minimum, average, and maximum runtime for each of the solution methods for each group of $10,000$ individuals. After the initial transition during the first $30,000$ individuals, there is relatively

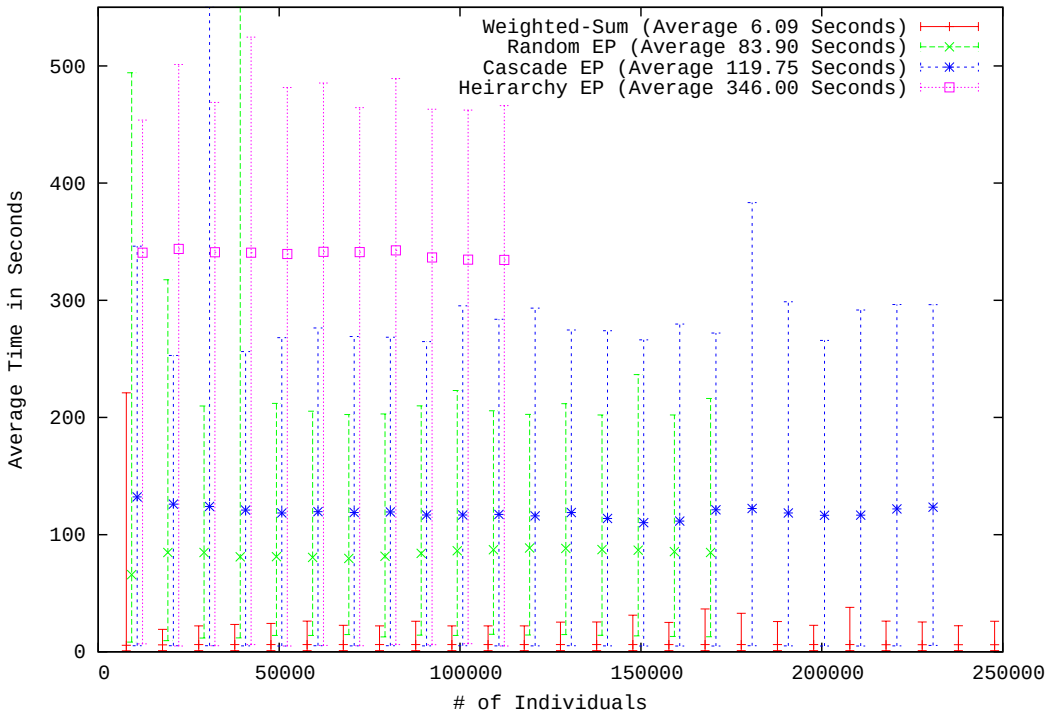

Figure 142: Minimum, Average and Maximum Evaluation Time Over Evolutionary Period for Four Examined Methods of Census Problem

little change in the average evaluation time. The best time evaluation performance is by far for the weighted-sum implementation, which is implemented using awk. The other runs implemented using the MOEP methods have the improved capability of including changes in topology, but these did come at a large run-time premium.

Of the three MOEP implementations, the random initialized individual has the lowest average runtime. But, note that the minimum time required is largest for the random initialized method. This increase for the minimum reflects an increase in the time required for the translation from the genome description to the Ptolemy II description. In comparison to the cascade or hierarchical initialized individuals, the random individuals contained many more impossible connections requiring more time in the recursive error detection and error correction procedures of the translation process as outlined in Section 7.2.2.

To allow a comparison of the run time required to reach the desired levels, Figure 143 changes the x-scale of Figure 141 to CPU days. This representation accentuates the

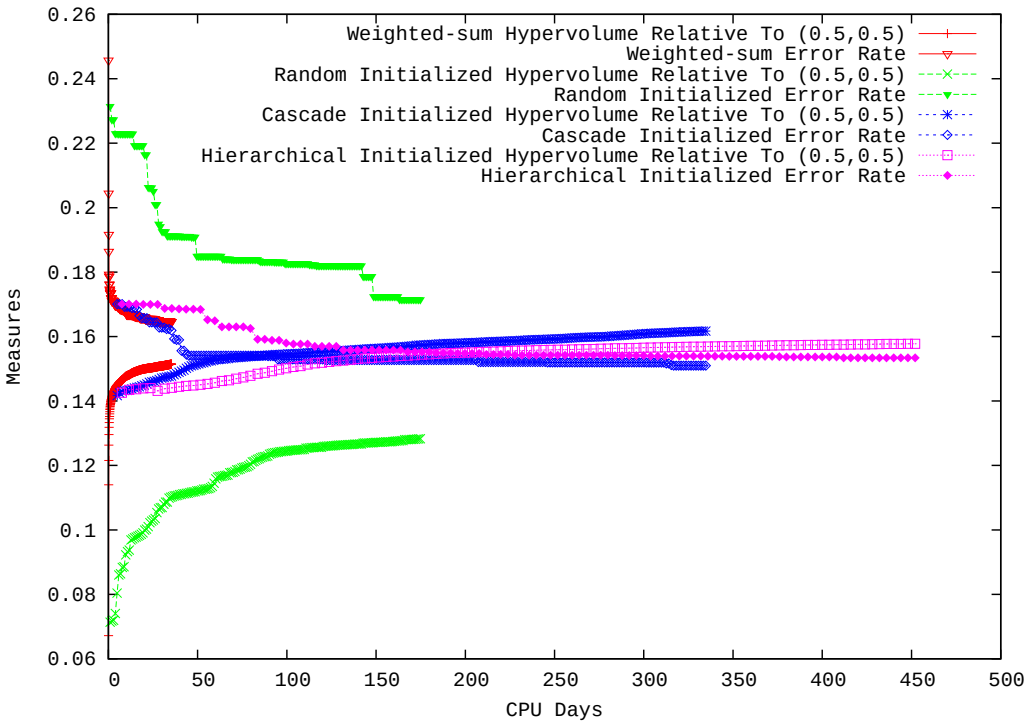

Figure 143: Measures of Performance versus CPU time for Four Examined Methods for Census Problem. Hypervolume calculated using Equation 3. Error rate calculated using Equation 163.

improvements made using the simple EA methods of the weighted-sum. Note that due to the relatively quick evaluation of the weighted-sum, the overall speed of evolution would like have been improved by using a later version of the weighted-sum optimization to seed the cascade and hierarchical optimization.

Also, note that the hypervolume for all three evolutionary programming techniques has a positive slope at the end of the run indicating that improvements are still being made. The limiting factor is receiving enough processor time. Other evolutionary programming techniques, such as that of Koza [37] require up to one million individuals to converge. Therefore, these methods have not reached their final potential.

Another aspect of this research is the use of dynamic objective thresholding. As detailed in Section 7.4, an effort was made to run a smaller, 5,000 data point, set of data for the first two objectives followed by a larger, 26,162 data point, set of data for the third and fourth

objectives. The effects of dynamic objective thresholding can be seen in Figure 142 by the large difference between the average and maximum run times, indicating the truncation of the evaluation of individuals after the evaluation of only the smaller set of data. Table 24 details the percent of individuals truncated.

Truncation simply indicates a poor-performing individual. Considering that the individual is the product of two good individuals, the poor performance must originate from either the crossover or mutation operations. To measure the effects of mutations, Table 24 also shows the calculation of the percent of individuals mutated based on the number of attributes available for mutation with a typical individual. There is a high correlation between mutation rate and the percentage of poor performing individuals.

Table 24: Effects of Dynamic Objective Thresholding for Census Problem. Percent of Individuals Truncated After Evaluation of the First Set of Training Data. Percent of Individuals Mutated.

Optimization Method	**Percent Truncated**	**Percent Individuals Mutated**
Weighted-Sum MOEA	9.3	$1.0 - 0.99^{2(105)} = 87.88\%$
Random Initialized MOEP	33.3	$1.0 - 0.99^{2(71*11+32*9)} = 99.9999\%$
Cascade Initialized MOEP	48.4	$1.0 - (0.995)^{2(12*11+105*9)} = 99.998\%$
Hierarchical Initialized MOEP	16.8	$1.0 - (0.9995)^{2(54*11+213*9)} = 91.8864\%$

Table 25 displays the results of the four approaches used in this research in comparison with the results of other pattern recognition methods. Note, none of the methods were able to outperform all of the previously designed pattern recognition methods.

Table 26 displays the error rates from each of the four methods for both training and evaluation. From this table, note each of the MOEP methods performs better in evaluation than for training. But, more importantly the evaluation of the weighted-sum performs much worse than for the training data. This is an indication that the weighted-sum method is over-trained. Therefore, the limits of the weighted-sum method have likely been reached. In contrast, the MOEP methods have not met these same limits.

Table 25: Error Rates for Census Pattern Recognition Problem for Various Algorithms, with Results of This Research **Emphasized**

Algorithm	**Error Rate (%)**
FSS Naive Bayes	14.05
NBTree	14.10
IDTM (Decision table)	14.46
C4.5-auto	14.46
HOODG	14.82
Cascade Initialized MOEP	**14.88**
C4.5 rules	14.94
OC1	15.04
Hierarchical Initialized MOEP	**15.23**
C4.5	15.54
Voted ID3 (0.6)	15.64
CN2	16.00
Naive-Bayes	16.12
Voted ID3 (0.8)	16.47
Random MOEP	**16.68**
T2	16.84
Weighted-Sum	**17.21**
1R	19.54
Nearest-neighbor (3)	20.35
Nearest-neighbor (1)	21.42

Table 26: Comparison of Training and Evaluation Error Rates

Optimization Method	**Training Error Rate**	**Evaluation Error Rate**
Weighted-Sum MOEA	16.49	17.21
Random Initialized MOEP	17.15	16.68
Cascade Initialized MOEP	15.20	14.88
Hierarchical Initialized MOEP	15.37	15.23

7.6 Conclusions

Overall, the results of this research are disappointing. None of the methods developed are able to outperform the top five pattern recognition methods. There is, however, knowledge that should not be lost. Two sub-sections are used to organize this knowledge. The first sub-section details the lessons learned from this research. The second sub-section provides future directions recommended for this line of research.

7.6.1 Lessons Learned

Several suppositions of this research have been confirmed. First, *the block diagram method is able to use the ability to modify topology to improve performance*. This capability is seen in all three of the MOEP methods investigated. Second, *changes in topology should support improved performance.* This capability is seen in both the cascade and hierarchical initialized MOEP methods, which are able to dramatically improve performance beyond that of the weighted-sum systems used to seed the evolution. Third, *the use of dynamic objective thresholding can be used to improve performance*. This improved performance is obtained through the truncation of individuals. As illustrated in Table 24, dynamic thresholding was able to eliminate 9 to 48 percent of the individuals after the evaluation of a smaller set of data.

Another interesting lesson learned is that *seeding of solutions with the results of other system optimization is able to dramatically improve performance*. This dramatic improvement is best seen in the comparison of the random initialized methods with those of the cascade initialized methods.

The downfalls of this research are mostly related to the time required to evolve solutions. The best solution requires over 300 CPU days to evolve. Although there is an obvious correlation between the complexity of the solutions and the runtime, as indicated in Figure 142, the large difference for implementation in Ptolemy II versus an equivalent solution in awk is unacceptable. In addition, awk is also known to be an inefficient interpretive

language as compared to compiled languages such as C++.

Ptolemy II does offer a very flexible simulation infrastructure. But, for many pattern recognition problems the full capabilities of this infrastructure are not needed. Therefore, *it is recommended to eliminate the use the Ptolemy II infrastructure for the MOEP optimization*.

7.6.2 Future Directions

The ability to translate a linear genome to a block diagram system already exists with the capabilities of the MoML generator discussed in Section 7.2.2. But, Ptolemy II in particular does not seem to be the best system description due to run time constraints. Therefore, *it is recommended to interface the capabilities of the MoML generator to a new evaluator*.

Two possible evaluators are worthy of pursuit. First, the freely available Matlab clone Octave [2] is rapidly maturing. It may be possible to modify the MoML generator to output an Octave system description for evaluation. Use of a tool such as Octave should provide a fast running infrastructure with a large number of developed and tested basis functions.

The second possible method is the development and integration of the necessary basis functions straight into the MoML generator. This has some appeal as the MoML generator must already create and allow recursion through the topology of a block diagram system. An obvious first basis function is the expression actor, which provides a configurable interface to a plethora of non-linear math functions. Interestingly with both of these methods, the existing MoML generator can be used to generate system description that can be visualized using the Vergil tool regardless of the infrastructure used to evaluate the system in the optimization process.

A final limitation that needs to be removed is the inability to assign different mutation characteristics to different parts of the system description. Currently, a single mutation rate is assigned to all attributes of the system description. But, as seen in the hierarchical optimization in Section 7.5.5 components may have different purposes, thus the need for different mutation rates. For example, different attributes are used to control the topology,

function type, and function attributes. Likewise, the mutation setup should be dependent on the purpose and type of attribute being mutated, e.g., normal mutation is applicable for most variable types, but it is much less applicable to enumerated attributes. Although the GTMOEA configuration allows different mutation configurations for different named components, the PRESTO description assigns the same name to all attributes. This problem should be easily eliminated through a slight redesign of the genome used to describe the PRESTO description.

CHAPTER VIII

CONCLUSIONS

In summary, *this research augments current Multiple Objective Evolutionary Algorithms with methods that dramatically reduce the time required to evolve toward a region of interest in objective space*. The research also demonstrates the effectiveness of these methods with optimization in three complex objective domains of flare pattern design, missile warning receiver algorithm optimization, and block diagram Multiple Objective Evolutionary Programming optimization.

The major original contributions of this thesis are:

1. Development of a hierarchical search space description that allows association of crossover and mutation settings with elements of the genotypic description.

2. Development of a method for parallel evaluation of individuals that removes the need for synchronization delays at the end of each generation.

3. Dynamic evolution of thresholds for objectives to allow partial evaluation of objectives for individuals.

4. Dynamic objective orderings to minimize the time required for unnecessary objective evaluations.

5. Application of MOEAs to the computationally expensive flare pattern design domain.

6. Application of MOEAs to the optimization of fielded missile warning receiver algorithms.

7. Development of a new method of using MOEAs for automatic design of pattern recognition systems.

This chapter is divided into two sections. The first, Section 8.1, provides a summary of each of the original contributions. The second, Section 8.2, provides recommendations for future directions of this research.

8.1 Summary

This section provides subsections that summarizes each of the research contributions itemized above.

8.1.1 Hierarchical Search Space Description

Section 3.1.1 detailed the simple description of the genome encoding that is used to define a hierarchical search space description. The hierarchical search space description supports the definition of the name, base data type (sampled, double precisions floating point, or enumerated), and the acceptable range of values for each of the search space dimensions. In addition, search space dimensions can be combined together into logical groups, which improves understanding and enables quick duplication of defined search space dimensions.

A description of the genome that is created externally to the software allows tailoring to new domains without recompilation of the software. Providing a generic system description also hides the underlying complexity of the evolutionary operators from the end-user, e.g., the end-user does not need to worry about how many bits are required to encode various attributes or how they are stored or communicated. These capabilities were utilized in every application of this research including:

1. 0/1 Knapsack optimization of Section 4.1.
2. Debs T function optimization of Section 4.2.
3. Flare pattern design optimization of Chapter 5.
4. AAR-44 missile warning receiver algorithm optimization of Chapter 6.
5. MOEA algorithm optimization of the census problem of Section 7.4.

6. MOEP block diagram optimization of the census problem of Section 7.5.

By providing a name to each attribute, EA operators such as mutation and crossover can be associated with each attribute. This capability was especially important for assigning standard deviations to normal mutation operators during the asexual reproduction phase required to seed the evolution of the AAR-44 missiles warning receiver algorithms in Section 6.2.

8.1.2 Removal of Generation Synchronization

Section 2.1.6 outlined the current methods for single and multiple gene pool parallel objective evaluations. That section also analyzed the effects of the large synchronization times required at the end of each generation. These synchronization times are most detrimental for the ICEO class of problems where:

1. The computation time for objective evaluations is large in comparison to the time required to perform the EA operations.

2. The evaluation time is dependent on the genome being evaluated.

3. Processes are running on many machines which may have various processing speeds due to their processing speed and loading.

To implement this approach, each processor is given an individual to evaluate. Remaining individuals in the current generation, C, are given to a processor for evaluation as they finish the evaluation of their current individual. When there are no more individuals in the current generation that need evaluating, or are not in the process of being evaluated, then the software creates the next generation based on the current population and the elite population. Without generation synchronization, the software also becomes more fault tolerant, i.e., if a processor goes down for some reason, the individual being evaluated is lost, but the entire evolutionary process is not halted as it would be with generation synchronization. Fault tolerance is especially important for ICEO optimization problems, which require long

optimization times, and thus have a probability of processor or system malfunction that is not negligible.

The removal of generation synchronization was shown in Section 5.2.5 to reduce the time required for flare pattern optimization an average of forty-six percent. These methods were then utilized on the AAR-44 optimization of Chapter 6 and the census problem optimization of Chapter 7.

8.1.3 Dynamic Objective Thresholding

As described in Section 3.4.1, dynamic objective thresholding is used to terminate the evaluation of an individual before all objectives are evaluated if it is determined on the basis of already-evaluated objectives that the individual is not likely to improve performance. An individual that does not have all objectives evaluated still has a fitness value. The fitness value is obtained with the remaining objectives values set to the minimum value for the objective. It is only by not evaluating all objectives for every individual that speed improvements can be made.

Section 4.1.3 illustrated that dynamic objective thresholding is able to reduce the number of objectives evaluated, especially for problems with many objectives. Section 5.2.1 showed the capability of dynamic objective thresholding to reduce the time required for evolution by forty-one percent for the flare pattern design problem. Dynamic objective thresholding was especially effective with the block diagram method, as detailed in Table 24, which shows the truncation of 9.3 to 48.4 percent of individuals. Dynamic objective thresholding was also utilized with the AAR-44 missile warning receiver algorithm optimization. As listed in Table 19, up to twenty-three to seventy-two percent of individuals were truncated, depending on the setup conditions and number of objectives.

8.1.4 Dynamic Objective Ordering

Objective ordering allows the user to preset the order and the software to automatically modify the order in which objectives are evaluated. This capability feeds two purposes.

First, the software can dynamically order the objectives creating objective orders that further improve speed performance beyond that of dynamic objective thresholding with the default objective ordering alone. Second, the user can force different orderings of objectives for different demes, enabling the different demes to focus on different desired attributes of the final individual. Note that either method requires the dynamic objective thresholding algorithm of the previous section to eliminate the evaluation of objectives. *Without dynamic objective thresholding, the objective ordering can not provide any performance improvement.*

There are currently two dynamic objective ordering algorithms: hypercube distance and auto ordering. The hypercube distance ordering algorithm orders objectives based on their average distance from the region of interest. This allows concentration on objectives that are furthest from the region of interest. The auto ordering method takes into account the time to evaluate each objective, and tries to order objectives in an order that reduces the time to evaluate a total population.

The capability of the HCD dynamic ordering method with the 0/1 Knapsack problem is given in Sections 4.1.3.3, 4.1.3.4 and 4.1.3.5. These results show that dynamic objective ordering is especially effective for problems with more than two objectives, and is in some circumstances able to outperform the best static ordering.

The results of HCD dynamic ordering with the flare optimization problem of Section 5.2.2 show that on average it requires thirty-one percent more time with HCD objective ordering than is required without it. Problems with dissimilar run-times are best served by the auto ordering method, which showed an average sixteen percent decrease in the time required for optimization for the flare pattern design problem.

With the optimization of the AAR-44 missile warning receiver, much was known about the run-times, and relative difficulty of the objectives before optimization began. As described in Section 6.3.1, this problem utilizes multiple objective orderings to build multiple demes that focused on different design criteria.

8.1.5 Flare Pattern Optimization

Chapter 5 illustrates that flare patterns for the defeat of imaging seekers from multiple angles and ranges can be found using MOEA methods. In fact, the ICEO methods are arguably the most important for this particular problem for the following reasons:

1. The run-times for each objective are large, requiring two to six minutes.
2. The run-times are both objective and genome dependent.
3. The solutions are not intuitively obvious.
4. The entire Pareto front is not desired, only those solutions that reside near or inside the region of interest.

8.1.6 Missile Warning Receiver Algorithm Optimization

The purpose of this research was to improve the performance of the AAR-44A OFP. The AAR-44A is a missile warning receiver utilized on United States Air Force Special Operation Forces aircraft. With the aid of government technical input, four measures of performance were selected: probability of declaration, false alarm performance, time to intercept minimum, and time to intercept average. These objectives were further divided into threat groups and regions of performance to create a 22-dimensional objective space. By inspection of the source code, 156 constants within the OFP were identified as having the possibility of improving system performance. These constants, along with their associated limits, defined a 156-dimensional search space.

To optimize the OFP performance against false alarms, live-fire data, and simulated data, MOEAs were applied to find the set of Pareto optimal individuals in the 22-dimensional objective space consisting of the various threat and region groupings. The objective space was then reduced back to a four-dimensional objective space mentioned above using a weighted-sum. The results of this optimization showed dramatic improvements in all four dimensions.

The research in optimization of the AAR-44A resulted in several conclusions that were applicable to MOEA optimization in general. First, the combinations of previous MOEA algorithms coupled with the new ICEO methods of dynamic objective thresholding, hypercube distance scaling, and multiple demes were able to improve the performance of a very complex system. Even the evaluation of $576,929$ individuals was achieved using parallel evaluation and objective thresholding within one month of calendar time.

Second, tuning a system is different from designing a new system. As such, the ability to start from a single known solution is required for tuning a system. Tuning an existing system requires a short period of high mutation using normal distributions to create a genotypicly and phenotypicly diverse population. This high mutation period is then followed by a longer period of lower rate mutation to provide evaluation of different combinations of the new genetic material.

Third, although identification and removal of variables from the search space does dramatically reduce the size of the search space, it does not dramatically reduce the time required to explore the search space when tuning an existing system. Therefore, bad individuals will only be created by the rate of the mutation P_m for that attribute, which typically is around one to two percent. Therefore, it is better to add another variable to the search space than to miss one important variable. As illustrated by this optimization, 156 variables are selected, and sensitivity analysis at the end shows only 74 of the variables are important.

8.1.7 Block Diagram Multiple Objective Evolutionary Programming

The block diagram multiple objective evolutionary programming technique was detailed in Section 7.2. The block diagram method was able to use the ability to modify topology to improve performance. This capability was seen in both the cascade and hierarchical initialized MOEP methods, which were able to dramatically improve performance beyond that of the weighted-sum systems used to seed the evolution.

Another interesting lesson learned from this research is that *seeding of solutions with the results of other system optimization was able to dramatically improve performance*. This dramatic improvement was best seen in the comparison of the random initialized methods with those of the cascade initialized methods.

The disappointments of this research were mostly related to the time required to evolve solutions. With over 300 CPU days required to evolve, the use of Ptolemy II is computationally prohibitive. Ptolemy II does offer a very flexible simulation infrastructure. But, for many pattern recognition problems the full capabilities of this infrastructure are not needed. Therefore, *it is recommended to eliminate the use the Ptolemy II infrastructure for the MOEP optimization*.

8.2 Future Directions

This section provides an overview of the many future directions this research can take. The areas for directions are divided into three subsections. The first subsection discusses possible improvements in the GTMOEA software application for ICEO problems. The second subsection proposes improvements to the PRESTO software application for MOEP problems. The third subsection proposes the application of this research to other domains.

8.2.1 Improvements in Georgia Tech Multiple Objective Evolutionary Algorithm Software

GTMOEA provides a modular infrastructure for integration, testing, and production use of new and previously developed MOEA concepts. The following subsections provide concepts for improvements to this software package that would potentially improve its application to ICEO problems. In all cases, promising methods can be evaluated using 0/1 Knapsack, 0/1 Deb's T function, and Census problems. The results for these problems given in Chapters 4 and 7 will provide a good basis for comparison to measure the improvement in performance.

8.2.1.1 Software Infrastructure Improvements

As discussed in Section 3.1.1, a DTD exists that defines the hierarchy, name, data type, and range of all dimensions of the search space. Since the original development of the software, methods such as the XML schema have matured. These methods should be able to provide the same information as the existing DTD in a more standard format. It is therefore proposed to replace the current search space description with a more standard interface.

In addition, many other concepts have been developed for the standard evolutionary algorithms operators of selection, crossover, mutation, and elitism. Specific concepts include averaging crossover methods [46, 47], and dynamic mutation rates. These methods should be examined for applicability to ICEO problems. Those methods with applicability should be added to the software and then evaluated for performance improvement.

As discussed in Section 2.1.6 and shown to be effective in Section 5.2.5, the innovation for the removal of the need for generation synchronization improves performance by keeping processors busy evaluating individuals instead of waiting for an individual to be evaluated. One possible wait condition still exists. This wait condition occurs while the master process, which is responsible for aggregating the results from many function evaluator processors, is busy creating the next generation of individuals, evaluating dynamic thresholds, or creating dynamic objective orders instead of keeping the external evaluators busy. This wait condition is most likely to manifest itself when many external processors are used for function evaluators. Figure 39 illustrated the current processing. Figure 144 illustrates a proposed approach using three threads to remove the major processing elements from the thread responsible for servicing the function evaluator processors.

8.2.1.2 Historical Information

The concepts of dynamic objective thresholding and dynamic objective ordering both utilize historical information in order to improve the speed of evolution. The idea of using past

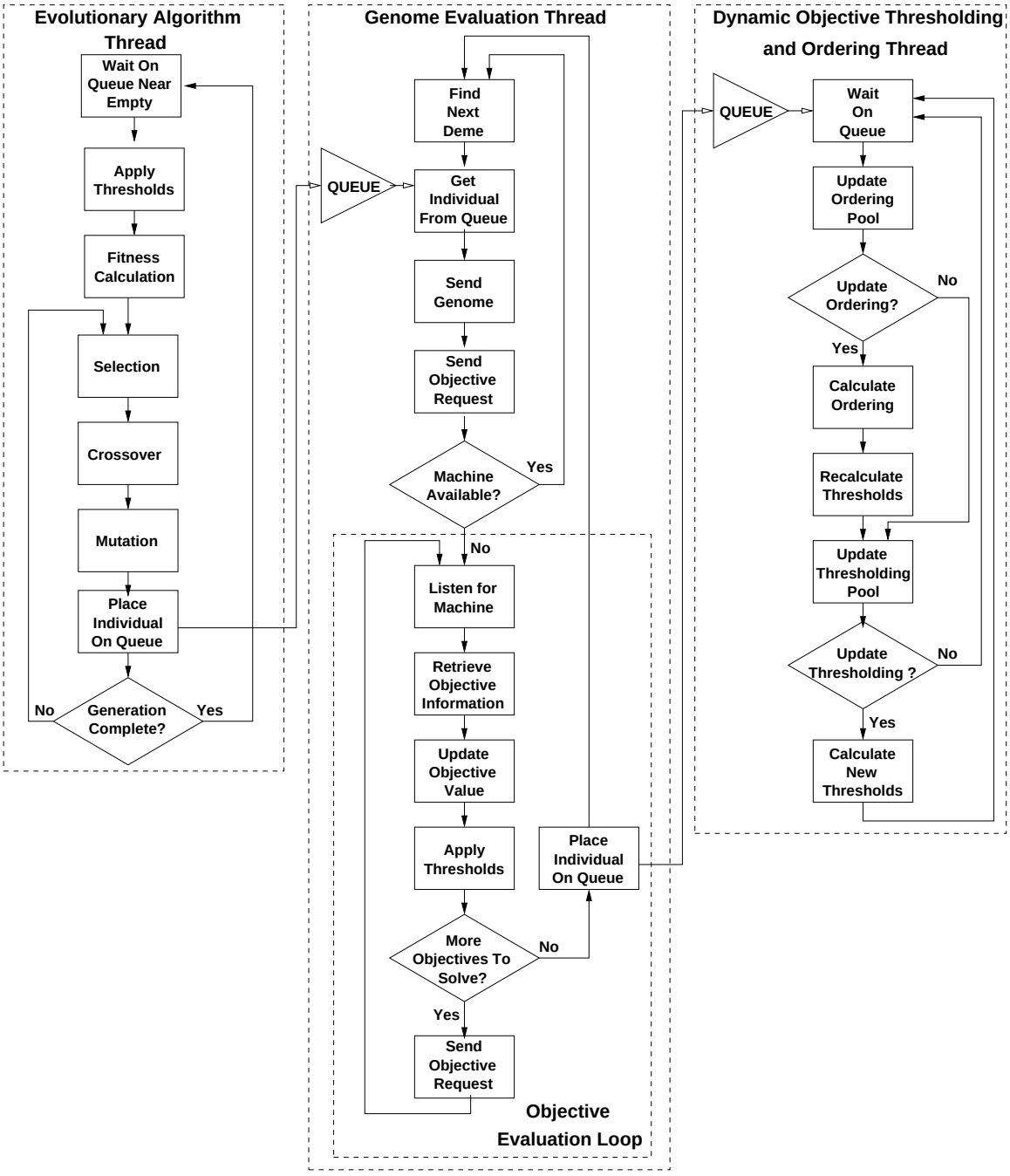

Figure 144: Proposed Architecture for Multi-Threaded Software Design

information to improve future performance could be applied to other EA operators as well. For example, by using information about the crossover locations and mutation operations that result in improved performing individuals, these operators can be tuned to increase the likelihood of crossover at specific locations or mutation of specific variables.

The first step of the research would be to implement the recording of the EA operators utilized to create new individuals. This data should be analyzed to find the "family tree" of individuals. Any trends in operations that are more common in good performing individuals than in poor performing individuals become candidates for dynamic setting from population information. The census problem is an excellent candidate for evaluation of the technique because it contains non-trivial, real-valued attribute values whose sensitivity to mutation should be attribute dependent.

8.2.1.3 Distance From Pareto Front as Performance Indicator

This thesis research used dynamic objective thresholds to greatly reduce the number of objectives required. Division of training data into smaller subsets was also used to allow dynamic thresholds to eliminate poor performing individuals after the evaluation of only a few objectives. This division of data should result in sets of objectives that are highly correlated. If the data sets are highly correlated, then the individuals that are Pareto optimal in the lower dimensional objective space from one set of objectives should be highly correlated with the Pareto optimal individuals in the objective space defined by the other set of objectives.

Therefore, in addition to the dynamic thresholds which push solutions toward a region of interest, the minimum distance of individuals from the Pareto optimal front in the lower dimension could also be used to provide a means of truncating evaluation of poor performing individuals. Figure 145 illustrates the concept in two dimensions, where C is the current population and E is the elite population.

Let E_P be the Pareto individuals in the elite population, then the minimum distance to

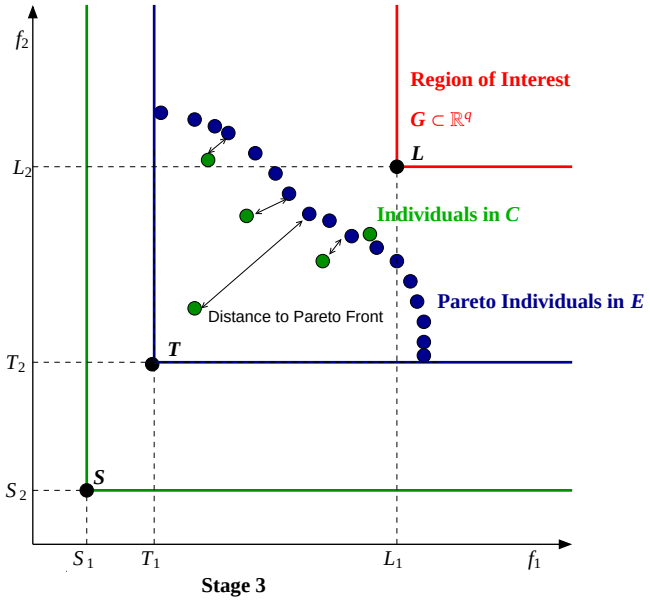

Figure 145: Distance From Pareto Front as Performance Indicator

E_P for an individual i in C is

$$d_{min}(i, E_P) = min_{j \in E_P} \left[\left(\sum_{k=1}^{q} (|f_k(i) - f_k(j)|)^p \right)^{\frac{1}{p}} \right], \tag{172}$$

where p is the Holder coefficient. Because of the normalization of objective values the range is

$$d_{min}(i, E_P) \in [0, q^{\frac{1}{p}}]. \tag{173}$$

An individual i would only be allowed to transition to the next set of objectives if it exceeds the dynamic threshold $\boldsymbol{T}$, and either dominates the current solutions in E or has a $d_{min}(i, E_P) > \beta$, where β is some constant in the valid range given in Equation 173.

Usefulness of the measurement of the distance from the current Pareto front is predicated on the correlation of the objectives. The correlation between objectives can be measured from those objectives that are solved. The value of beta can then be increased or decreased based on the amount of correlation between the sets of objectives.

8.2.2 Multiple Objective Evolutionary Programming Improvements

The block diagram approach demonstrated the ability to evolve solutions that improved solutions through changes in topology and use of non-linear components. A problem with this approach was the ability to evaluate enough individuals to investigate the effects of changes in input architecture, basis functions, and MOEA settings. This problem was due to the poor run-time performance of the Ptolemy II simulation. It is anticipated that the other function evaluators could achieve up to 100 times faster evaluation.

As discussed in Section 7.6.2, two possible evaluators are worthy of pursuit. First, the freely available Matlab clone Octave is rapidly maturing. It may be possible to modify the MoML generator to output an Octave system description for evaluation. Use of a tool such as Octave should provide a fast running infrastructure with a large number of developed and tested basis functions. The second possible method is the development and integration of the necessary basis functions directly into the MoML generator. This has some appeal as the MoML generator must already create and allow recursion through the topology of a block diagram system.

In addition, a final limitation that requires removal is the inability to assign different mutation characteristics to different parts of the system description. Currently, a single mutation rate is assigned to all attributes of the system description. But, as seen in the hierarchical optimization in Section 7.5.5, components may have different purposes, and thus need different mutation rates. Although the GTMOEA configuration allows different mutation configurations for different named components, the PRESTO description assigns the same name to all attributes. This problem can be easily eliminated through a slight redesign of the genome used to describe the PRESTO description.

8.2.3 Application of Multiple Objective Evolutionary Algorithms to Additional Domains

This section describes possible applications of the MOEA to additional ICEO domains.

8.2.3.1 Application to Additional Missile Warning Receiver

The MOEA techniques were successfully applied to the AAR-44 missile warning receiver, see Section 6, implying these techniques would also be very applicable to similar warning receiver systems. At the time of this writing, GTRI is already under contract with the Air Force for optimization of the AAR-47 missile warning receiver algorithms using MOEA techniques. The results of the AAR-44 optimization have also been presented to BAE Systems, the developers of the AAR-57 Common Missile Warning System.

8.2.3.2 Application to Erythema Detection

GTRI is also currently under contract with the National Institute of Health to aid in multi-spectral detection of erythema. The MOEA and MOEP methods will be utilized in optimizing the detection processes.

8.2.3.3 Use of Pareto Optimal solutions for Extended Operation Conditions

As seen during the optimization of the AAR-44 algorithms in Section 6, the operating scenarios and conditions can be used to multiply the number of objective space dimensions; e.g., the objectives may be TP, FP and TTI for threat A and the TP, FP and TTI for threat B. Other operating scenarios and conditions include the state of the sensor platform (e.g. position and velocity), and the state of the environment (e.g. visibility, ozone concentration, and sun angle).

If optimization is performed in this large objective space, then a large Pareto-optimal set of possible algorithms is created. The best possible solution for use is then dictated from a combination of sensor and operator inputs. Sensor inputs such as weather condition and sensor platform state can be used to reduce the size of the objective space. Human operators can then use mission requirements to determine the relative importance of each of the possible threats, the acceptable FP performance, and the acceptable TTI. This combination of constraints and weights from sensor and operator inputs is then used to find the

single best algorithm from the set of Pareto optimal solutions that is optimal for the current conditions. These methods allow the algorithms to adapt to extended operating conditions, (EOCs), based on the solutions found from a single MOEA/MOEP optimization.

Several EOC events will make continued MOEA optimization advantageous. New training data of an existing system may become available for a variety of reasons including additional testing of the system, use of the system in combat conditions, improved modeling of the system, introduction of new targets, and emergence of new countermeasure methods. Replacement of training data may be required due to drastic changes in sensor noise, sensitivity, and resolution. Other EOC events such as the introduction of a new target processor or the development of novel basis algorithms may also provide the ability to improve performance via continued optimization.

APPENDIX A

GEORGIA TECH MULTIPLE OBJECTIVE EVOLUTIONARY SOFTWARE DATA TYPE DESCRIPTION

```
<!ELEMENT MOEA ( Seed, Evaluator, Terminate,
                 DebugPareto?, Deme+ )>
  <!--*************************************************-->
  <!-- Seed for random number generator                -->
  <!ELEMENT Seed ( #PCDATA )>
  <!--*************************************************-->
  <!-- Evaluator defines interface to remote           -->
  <!-- evolution                                       -->
  <!ELEMENT Evaluator ( Name, TmpDir?, GeneDescription,
      Header*, Trailer*, ObjectiveList, MachineList,
      Debug?)>
    <!ELEMENT Name              ( #PCDATA )>
    <!ELEMENT TmpDir            ( #PCDATA )>
    <!ELEMENT GeneDescription   ( #PCDATA )>
    <!ELEMENT Header            ( #PCDATA )>
    <!ELEMENT Trailer           ( #PCDATA )>
    <!ELEMENT ObjectiveList     ( (Name, MinValue?,
      Header*, Trailer*)+ )>
    <!ELEMENT MachineList       ( Name+ )>
    <!ELEMENT Debug             ( #PCDATA )>
  <!--*************************************************-->
  <!-- Terminate node determines when to stop          -->
  <!-- executable                                      -->
  <!ELEMENT Terminate ( MaxFitness?, MaxObjectives?,
                        MaxGenome?, InsideRegion? ) >
    <!ELEMENT MaxFitness        ( #PCDATA )>
    <!ELEMENT MaxObjectives     ( #PCDATA )>
    <!ELEMENT MaxGenome         ( #PCDATA )>
    <!ELEMENT InsideRegion      ( #PCDATA )>
  <!--*************************************************-->
  <!-- Optional DebugPareto is used to provide         -->
```

```
<!-- Pareto optimal solutions while running          -->
<!ELEMENT DebugPareto                ( Delta, Type, Name )>
  <!ELEMENT Delta                    ( #PCDATA )>
  <!ELEMENT Type                     ( #PCDATA )>
<!--*********************************************-->
<!-- Multiple Demes are used to define evolution   -->
<!-- of parallel populations                       -->
<!ELEMENT Deme ( Name, Fitness, Output, Init,
                 Selection, Crossover, Mutation )>
  <!--*********************************************-->
  <!-- The Fitness section defines the mapping     -->
  <!-- from multiple objectives to a single        -->
  <!-- fitness value                               -->
  <!ELEMENT Fitness ( (ParetoRank1|ParetoRank2|SPEA1|
                       SPEA2|SPEA3|WeightedSum),
      (HCDScale|Niching|Transform|Normalize|
           SPEA2Density)*, Thresholding?, Ordering? )>
    <!--*********************************************-->
    <!-- There are seven fitness calculation        -->
    <!-- methods: ParetoRank1, ParetoRank2, SPEA1,  -->
    <!-- SPEA2, SPEA3, WeightedSum, HCD -->
    <!ELEMENT ParetoRank1            ( RankLimit )>
      <!ELEMENT RankLimit            ( #PCDATA )>
    <!ELEMENT ParetoRank2            ( RankLimit )>
    <!ELEMENT WeightedSum            ( Weight* )>
      <!ELEMENT Weight               ( Name, Value )>
        <!ELEMENT Value              ( #PCDATA )>
    <!ELEMENT SPEA1                  ( HolderCoef, MaxNum)>
      <!ELEMENT HolderCoef           ( #PCDATA )>
      <!ELEMENT MaxNum               ( #PCDATA )>
    <!ELEMENT SPEA2                  ( HolderCoef )>
    <!ELEMENT SPEA3                  ( HolderCoef, MaxNum)>
    <!--*********************************************-->
    <!-- There are five fitness scaling methods:    -->
    <!-- HCDScale, Niching, Transform, Normalize,   -->
    <!-- SPEA2Density -->
    <!ELEMENT HCDScale ( HolderCoef, Minimize?, Offset? )>
    <!ELEMENT Niching  ( NumNiches, HolderCoef,
                         Minimize?)>
      <!ELEMENT NumNiches            ( #PCDATA )>
      <!ELEMENT Minimize             ( #PCDATA )>
    <!ELEMENT Transform ( Point, Point, Point*)>
      <!ELEMENT Point    ( In, Out)>
        <!ELEMENT In                 ( #PCDATA )>
        <!ELEMENT Out                ( #PCDATA )>
```

```
  <!ELEMENT Normalize          ( #PCDATA )>
  <!ELEMENT SPEA2Density ( HolderCoef )>
  <!--*****************************************-->
  <!-- Objective Thresholding is used to limit  -->
  <!-- the number of objective calculations     -->
  <!ELEMENT Thresholding       ( (Gauntlet|
     FitnessThresh|FixedRate), NumHistory?,
     NumDelta?, Threshold+ )>
    <!ELEMENT Gauntlet         ( #PCDATA )>
    <!ELEMENT FitnessThresh    ( #PCDATA )>
    <!ELEMENT FixedRate        ( #PCDATA )>
    <!ELEMENT NumHistory       ( #PCDATA )>
    <!ELEMENT NumDelta         ( #PCDATA )>
    <!ELEMENT Threshold ( Name, MinValue, MaxValue,
                          MaxWeight )>
      <!ELEMENT MinValue       ( #PCDATA )>
      <!ELEMENT MaxValue       ( #PCDATA )>
      <!ELEMENT MaxWeight      ( #PCDATA )>
  <!--*****************************************-->
  <!-- Objective Ordering is used to change the -->
  <!-- order in which objectives are calculated. -->
  <!-- Coupled with objective thresholding,     -->
  <!-- this can further reduce the time for     -->
  <!-- objective evaluation.                    -->
  <!ELEMENT Ordering ( Force*, RandomInit?,
      CheckObjectives?, NumDelta?, NumHistory?,
      MinN?, Debug?,
      (Correlation|HCDOrder|AutoOrder) )>
    <!ELEMENT Force            ( #PCDATA )>
    <!ELEMENT RandomInit       ( #PCDATA )>
    <!ELEMENT CheckObjectives  ( #PCDATA )>
    <!ELEMENT MinN             ( #PCDATA )>
    <!ELEMENT Correlation      ( MinN, MaxR, MinT )>
      <!ELEMENT MaxR           ( #PCDATA )>
      <!ELEMENT MinT           ( #PCDATA )>
    <!ELEMENT HCDOrder         ( HolderCoef )>
    <!ELEMENT AutoOrder ( MaxDiff,
                          (ParetoBest|FitnessBest) )>
      <!ELEMENT MaxDiff        ( #PCDATA )>
      <!ELEMENT ParetoBest     ( #PCDATA )>
      <!ELEMENT FitnessBest    ( #PCDATA )>
<!--*****************************************-->
<!-- This section allows output of individual -->
<!-- data during the evolution process        -->
<!ELEMENT Output ( (CreatedOut|NewObjectiveOut|
```

```
                    CompletedOut)*)>
  <!ELEMENT CreatedOut          ( Type, Name )>
  <!ELEMENT NewObjectiveOut     ( Type, Name )>
  <!ELEMENT CompletedOut        ( Type, Name )>
<!--*****************************************-->
<!-- The Init section details how the        -->
<!-- populations are initialized             -->
<!ELEMENT Init ( GenSync?, NumPerGen,
    PreventDuplicates?, Elite?, RandomBits?,
    RandomValues?, File* )>
  <!ELEMENT GenSync             ( #PCDATA )>
  <!ELEMENT NumPerGen           ( #PCDATA )>
  <!ELEMENT PreventDuplicates ( #PCDATA )>
  <!ELEMENT Elite ( Num, CheckFitness?,
    CheckObjectives?, SPEA2Prune?, MinFitness?,
    MaxRank?, Pareto?, Clustering?, Thresholds?,
    Minimize?, DebugPool? )>
    <!ELEMENT Num               ( #PCDATA )>
    <!ELEMENT CheckFitness      ( #PCDATA )>
    <!ELEMENT SPEA2Prune        ( HolderCoef )>
    <!ELEMENT MinFitness        ( #PCDATA )>
    <!ELEMENT MaxRank           ( #PCDATA )>
    <!ELEMENT Pareto            ( #PCDATA )>
    <!ELEMENT Clustering        ( HolderCoef,
          PreserveTop, TopFitnessNum)>
      <!ELEMENT PreserveTop     ( #PCDATA )>
      <!ELEMENT TopFitnessNum ( #PCDATA )>
    <!ELEMENT Thresholds        ( #PCDATA )>
    <!ELEMENT DebugPool         ( GenDelta, Type,
          Name, Num )>
      <!ELEMENT GenDelta        ( #PCDATA )>
  <!ELEMENT RandomBits          ( #PCDATA )>
  <!ELEMENT RandomValues        ( #PCDATA )>
  <!ELEMENT File                ( Type, Name )>
<!--*****************************************-->
<!-- The selection method details how natural/ -->
<!-- artifical selection is modeled          -->
<!ELEMENT Selection ( (Roulette|Tournament|Stud)+ )>
  <!ELEMENT Roulette ( Rate, Gamma, Offset,
                       Minimize? )>
    <!ELEMENT Rate              ( #PCDATA )>
    <!ELEMENT Gamma             ( #PCDATA )>
    <!ELEMENT Offset            ( #PCDATA )>
  <!ELEMENT Tournament ( Rate, Num?, Minimize?,
                         Replacement? )>
```

```
    <!ELEMENT Replacement      ( #PCDATA )>
  <!ELEMENT Stud ( Rate , Name, Selection )>
<!--*****************************************-->
<!-- The crossover method determines how genome -->
<!-- are combined during the mating process     -->
<!ELEMENT Crossover ( NumParents?, RemoveParents?,
                      (NPoint|Location|Clone)+ )>
  <!ELEMENT NumParents        ( #PCDATA )>
  <!ELEMENT RemoveParents     ( #PCDATA )>
  <!ELEMENT NPoint ( Rate , Number, Boundaries? )>
    <!ELEMENT Number          ( #PCDATA )>
    <!ELEMENT Boundaries      ( #PCDATA )>
  <!ELEMENT Location          ( Rate , (Name)* )>
  <!ELEMENT Clone             ( Rate )>
<!--*****************************************-->
<!-- Mutation method                         -->
<!ELEMENT Mutation  ( Rate , (Bit|Normal|Uniform)+ )>
  <!ELEMENT Bit     ( Rate , BitRate , (On|Off)? )>
    <!ELEMENT BitRate         ( #PCDATA )>
    <!ELEMENT On              ( #PCDATA )>
    <!ELEMENT Off             ( #PCDATA )>
  <!ELEMENT Normal ( Rate ,NameRate , StdDev ,(Name)* )>
    <!ELEMENT StdDev          ( #PCDATA )>
    <!ELEMENT NameRate        ( #PCDATA )>
  <!ELEMENT Uniform     ( Rate , NameRate , (Name)* )>
```

APPENDIX B

MODELING MARKUP LANGUAGE DATA TYPE DESCRIPTION

```
<!ELEMENT class (class | configure | deleteEntity |
            deletePort | deleteRelation | doc |
            entity | group | link | port |
            property | relation | rename | unlink)*>
<!ATTLIST class name CDATA #REQUIRED
                extends CDATA #IMPLIED
                source CDATA #IMPLIED>
<!ELEMENT configure (#PCDATA)>
<!ATTLIST configure source CDATA #IMPLIED>
<!ELEMENT deleteEntity EMPTY>
<!ATTLIST deleteEntity name CDATA #REQUIRED>
<!ELEMENT deletePort EMPTY>
<!ATTLIST deletePort name CDATA #REQUIRED>
<!ELEMENT deleteProperty EMPTY>
<!ATTLIST deleteProperty name CDATA #REQUIRED>
<!ELEMENT deleteRelation EMPTY>
<!ATTLIST deleteRelation name CDATA #REQUIRED>
<!ELEMENT doc (#PCDATA)>
<!ATTLIST doc name CDATA #IMPLIED>
<!ELEMENT entity (class | configure | deleteEntity |
            deletePort | deleteRelation | doc |
            entity | group | link | port |
            property | relation | rename | unlink)*>
<!ATTLIST entity name CDATA #REQUIRED
                 class CDATA #IMPLIED
                 source CDATA #IMPLIED>
<!ELEMENT group ANY>
<!ATTLIST group name CDATA #IMPLIED>
<!ELEMENT link EMPTY>
<!ATTLIST link insertAt CDATA #IMPLIED
               port CDATA #REQUIRED
               relation CDATA #REQUIRED
               vertex CDATA #IMPLIED>
<!ELEMENT port (configure | doc | property | rename)*>
```

```
<!ATTLIST port class CDATA #IMPLIED
              name CDATA #REQUIRED>
<!ELEMENT property (configure | doc | property |
             rename)*>
<!ATTLIST property class CDATA #IMPLIED
                   name CDATA #REQUIRED
                   value CDATA #IMPLIED>
<!ELEMENT relation (configure | doc | property |
            rename | vertex)*>
<!ATTLIST relation name CDATA #REQUIRED
                  class CDATA #IMPLIED>
<!ELEMENT rename EMPTY>
<!ATTLIST rename name CDATA #REQUIRED>
<!ELEMENT unlink EMPTY>
<!ATTLIST unlink index CDATA #IMPLIED
                 insideIndex CDATA #IMPLIED
                 port CDATA #REQUIRED
                 relation CDATA #IMPLIED>
<!ELEMENT vertex (configure | doc | property | rename)*>
<!ATTLIST vertex name CDATA #REQUIRED
                 pathTo CDATA #IMPLIED
                 value CDATA #IMPLIED>
```

REFERENCES

[1] "Web-site for MLC++," 2004. http://www.sig.com/tech/mlc/index.html. 7.3

[2] "Web-site for Octave," 2004. http://www.octave.org. 7.6.2

[3] AHO, A. V., KERNIGHAN, B. W., and WEINBERGER, P. J., *The AWK Programming Language*. Reading, MA: Addison-Wesley, 1988. 3.2.1

[4] ANDRE, D. and KOZA, J. R., "A Parallel Implementation of Genetic Programming that Achieves Super-Linear Performance," *Information Sciences*, vol. 106, pp. 201–218, May 1998. 2.1.6.2

[5] BACK, T., *Handbook of Evolutionary Computation*, ch. C3.2.1, pp. C3.2:1–C3.2:2. New York, NY: IOP Publishing Ltd. and Oxford University Press, 1997. 2.1.4.1

[6] BLICKLE, T., *Handbook of Evolutionary Computation*, ch. C2.3, pp. C2.2:1–C2.3:4. New York, NY: IOP Publishing Ltd. and Oxford University Press, 1997. 2.1.2.2

[7] BLINN, B., *Portable Shell Programming*. Upper Saddle River, NJ: Prentice Hall, 1996. 3.2.1

[8] BOOKER, L. B., *Handbook of Evolutionary Computation*, ch. C3.3.1, pp. C3.3:2–C3.3:3. New York, NY: IOP Publishing Ltd. and Oxford University Press, 1997. 2.1.3.2

[9] BOSMAN, P. A. N. and THIERENS, D., "The Balance Between Proximity and Diversity in Multiobjective Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 174–188, April 2003. 3.1.5

[10] CANTU-PAZ, E. and GOLDBERG, D., "On the scalability of parallel genetic algorithms," *Evolutionary Computation*, vol. 7, pp. 429–49, Winter 1999. 2.1.6.1, 2.1.6.2, 2.3

[11] CATHCART, M. J., *Test Report Document for Task 1.1 - AAR-44 Re-Hosting of the AAR-44A Genetic Algorithm and Genetic Programming Project A-6706-110*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, January 2002. UNCLASSIFIED. 6.1.3.1

[12] CATHCART, M. J. and HARBERT, S. D., *Test Design Document for Task 1.2 - Infrastructure to Modify Parameters of the AAR-44A Genetic Algorithm and Genetic Programming Project A-6706-120*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, April 2002. UNCLASSIFIED. 6.1.3.2

[13] Cathcart, M. J. and Harbert, S. D., *Test Report Document for Task 1.2 - Infrastructure to Modify Parameters of the AAR-44A Genetic Algorithm and Genetic Programming Project A-6706-120*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, May 2002. UNCLASSIFIED. 6.1.3.2

[14] Cathcart, M. J., Rohling, G. A., and Harbert, S. D., *Requirements and Design Document for Task 1.2 - Infrastructure to Modify Parameters of the AAR-44A Genetic Algorithm and Genetic Programming Project A-6706-120*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, March 2002. UNCLASSIFIED. 6.1.3.2

[15] Coello-Coello, C. A., *An Empirical Study of Evolutionary Techniques for Multiple Objective Optimization in Engineering Design*. PhD thesis, Tulane University, 1996. Unpublished. 2.2.2

[16] Darwin, C., *The Origin of Species by Means of Natural Selection*. Baltimore, Maryland: Penguin Books, 1859. 1.2

[17] Deb, K., "Multi-objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems," *Evolutionary Computation*, vol. 7, pp. 205–230, November 1999. 1.5, 4, 4.2

[18] Dougherty, D., *Sed and Awk*. Sebastopol, CA: OReilly and Associates, 1992. 3.2.1

[19] Edwards, S. A. and Lee, E. A., "The Semantics and Execution of a Synchronous Block-Diagram Language," *Science of Computer Programming*, 2003. 7.2.1

[20] Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, *Users Manual for GT-MOEA Georgia Tech Multiple Objective Evolutionary Algorihtms for Independent Computationally Expensive Objectives*, June 2002. UNCLASSIFIED. 3.1.5

[21] Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, *Imaging Seeker and Missile Simulation Users Manual*, April 2004. UNCLASSIFIED. 5.1.2

[22] Fogel, D. B., *Handbook of Evolutionary Computation*, ch. C3.2.2, pp. C3.2:2–C3.2:5. New York, NY: IOP Publishing Ltd. and Oxford University Press, 1997. 2.1.4.2, 2.1.4.3

[23] Fonseca, C. and Fleming, P., "Multiobjective Optimal Controller Design with Genetic Algorithms," in *International Conference on Control '94 (Conf. Publ. No.389)*, vol. 1, pp. 745–9, 1994. 3.4.1

[24] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley Publishing Company, Inc., 1989. 1, 2.2.4

[25] Gradshteyn, I. and Ryzhik, I., *Table of Integrals, Series, and Products*. New York, New York: Academic Press, first ed., 1965. 2.2.4.1, 2.2.5.1, 2.2.5.1

[26] Grefenstette, J., *Handbook of Evolutionary Computation*, ch. C2.2, pp. C2.2:1–C2.2:7. New York, NY: IOP Publishing Ltd. and Oxford University Press, 1997. 2.1.2.1

[27] Gwo, J., Hoffman, F., and Hargrove, W., "Mechanistic-Based Genetic Algorithm Search on a Beowulf Cluster of Linux PCs.," in *Proceedings of the High Performance Computing 2000 (HPC2000) Conference, Washington, DC.*, pp. 148–153, April 2000. 2.1.6.1

[28] Harbert, S. D., Rohling, G. A., and Cathcart, M. J., *Requirements and Design Document for Task 1.6 Automation of Measure Extraction of the AAR-44A Genetic Algorithm and Genetic Programming Project A-6706-160*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, May 2002. UNCLASSIFIED. 6.1.3.3

[29] Holland, J. H., *Adaptation in Natural and Artifical Systems: An Introductory Analysis with Application to Biology, Control, and Artifical Intelligence*. Ann Arbor, MI: University of Michigan Press, 1975. 1.2, 2.1.2.1

[30] Horn, J., Nafpliotis, N., and Goldberg, D. E., "A Niched Paerto Genetic Algorithm for Multiobjective Optimization," in *Proceedings of the First IEEE Converence on Evolutionary Computations. Orlando, FL*, vol. 1, pp. 82–7, May 1994. 1.5, 2.2.4, 2.2.4.2, 2.2.4.2

[31] Howard, D., Roberts, S., and Brankin, R., "Target detection in SAR imagery by genetic programming," *Advances in Engineering Software*, vol. 30, pp. 303–11, May 1999. 7.1.5.1, 7.2, 7.2.3

[32] Kohavi, R., "Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996. 7, 7.3

[33] Kohavi, R. and Becker, B., "Web-site for 1994 census data," 2004. http://www.sig.com/tech/mlc/db/adult.names. 7.3

[34] Koza, J., *Genetic Programming : On the Programming of Computers by Means of Natural Selection*. Cambridge, Mass.: MIT Press, 1992. 7

[35] Koza, J., Andre, D., III, F. B., and Keane, M., "Use of Automatically Defined Functions and Architecture-altering Operations in Automated Circuit Synthesis with Genetic Programming," in *Proceedings of Genetic Programming 1996 Conference, Stanford, CA, USA*, pp. 132–40, July 1996. 7.1.5.2, 7.2

[36] Koza, J. and et al., *Genetic Programming III : Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann, 1999. 1.5, 7, 7.2

[37] Koza, J., III, F. B., D. Andre, D., and Keane, M., "Automated WYWIWYG Design of Both the Topology and Component Values of Electrical Circuits Using Genetic Programming," in *Proceedings of Genetic Programming 1996 Conference, Stanford, CA, USA*, pp. 123–31, July 1996. 7.1.5.2, 7.2, 7.5.6

[38] Lamm, D. R., *Requirements Analysis for Task 1.4.1 Offline Simulation Sensor Data*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, April 2002. UNCLASSIFIED. 6.1.4.2

[39] Lamm, D. R. and Shelton, R. J., *Simulation Procedures for Task 1.4.1 Offline Simulation Sensor Data*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, March 2002. UNCLASSIFIED. 6.1.4.2

[40] Lee, E. A., *Overview of the Ptolemy Project, Technical Memorandum UCB/ERL M03/25,* . University of California, Berkeley, CA, 94720, USA, July 2003. 7

[41] Lee, E. A., Neuendorffer, S., and Wirthlin, M. J., "Actor-Oriented Design of Embedded Hardware and Software Systems," *Journal of Circuits, Systems, and Computers*, vol. 12, no. 3, pp. 231–260, 2003. 7.2.1

[42] Martello, S. and Toth, P., *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, New York: J. Wiley and Sons, 1990. 1.5, 4, 4.1, 4.1.1

[43] Martin, W., Lienig, J., and Cohoon, J. P., *Handbook of Evolutionary Computation*, ch. C6.3, pp. C6.3:3–C3.3:6. New York, NY: IOP Publishing Ltd. and Oxford University Press, 1997. 2.1.6.2

[44] Murata, T. and Ishibuchi, H., "MOGA: Multi-Objective Genetic Algorithms," in *Proceedings of 1995 IEEE International Conference on Evolutionary Computation, Perth, WA, Australia*, vol. 1, pp. 289–94, Nov. 1995. 2.2.2, 2.2.3

[45] Neuendorffer, S. and Lee, E. A., "Hierarchical Reconfiguration of Dataflow Models," June 2004. 7.2.1

[46] Obayashi, S., Takahashi, S., and Takeguchi, Y., "Niching and Elitist Models for MOGAs," in *Parallel Problem Soving from Nature, PPSH V., Ansterdam, Netherlands*, pp. 260–9, Sept. 1998. 2.1.3, 3.1.3, 8.2.1.1

[47] Obayashi, S., Sasaki, D., Takeguchi, Y., and Hirose, N., "Multiobjective Evolutionary Computation of Supersonic Wing-Shape Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 4, pp. 182–187, July 2000. 2.1.3, 3.1.3, 8.2.1.1

[48] Patterson, E. M., *Data Division into Training and Evaluation Groups for Task 1.3 of the AAR-44A Genetic Algorithm and Genetic Programming Project A-6706-130*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, September 2002. UNCLASSIFIED. 6.1.4.1, 6.1.4.3

[49] Patterson, E. M., *Flight Data Requirements for Task 1.3 of the AAR-44A Genetic Algorithm and Genetic Programming Project A-6706-130*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, September 2002. UNCLASSIFIED. 6.1.4.1, 6.1.4.3

[50] Rohling, G. A. and Lamm, D. R., *Classified Supplement for Final Scientific and Technical Report for Task 1, Optimization of the AAR-44A Opertionaly Flight Program Using Multiple Objective Evolutionary Algorithms*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, September 2003. UNCLASSIFIED. 6

[51] Rohling, G. A. and Lamm, D. R., *Final Scientific and Technical Report for Task 1, Optimization of the AAR-44A Opertionaly Flight Program Using Multiple Objective Evolutionary Algorithms*. Electro-Optics, Environment, and Materials Laboratory, Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA 30332, September 2003. UNCLASSIFIED. 6

[52] Schaffer, J. D., *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, 1984. Unpublished. 2.2.2

[53] Selby, S. M. and Girling, B., *CRC Standard Mathematic Tables,*. Cleveland, Ohio: The Chemical Rubber Co., fourteenth ed., 1965. 2.2.4.1, 2.2.5.1, 2.2.5.1

[54] Stephens, W. R., *UNIX Network Programming*. Englewood Cliffs, NJ: Prentice Hall, 1990. 3.2.2.1

[55] Stephens, W. R., *Advanced Programming in the UNIX Environment*. Reading, MA: Addison-Wesley, 1992. 3.2.2.1

[56] Uesaka, K. and Kawamata, M., "Synthesis of Low-Sensitivity Second-Order Digital Filters Using Genetic Programming with Automatically Defined Functions," *IEEE Signal Processing Letters*, vol. 7, pp. 83–5, April 2000. 7.1.5.3, 7.1.5.3, 7.2

[57] Veldhuizen, D. A. V., *Multiobjective Evolutionary Algorithms: Classifications, Analyses and New Innovations* . PhD thesis, Air Force Institute of Technology, 1999. Unpublished. 1.2, 1.2.2, 3.1.2

[58] Veldhuizen, D. A. V., Zydallis, J. B., and Lamont, G. B., "Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 144–173, April 2003. 2.1.6

[59] Wolpert, D. H. and Macready, W. G., "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 67–82, April 1997. 1.2

[60] Xu, D. and Daley, M., "Design of optimal digital filter using a parallel genetic algorithm," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, pp. 673–5, Oct. 1995. 2.1.4

[61] Zitzler, E., *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology Zurich, 1999. Unpublished. 1.3.2, 1.3.3, 1.5, 2.2.5, 4.1, 4.1.1, 4.1.2.5

[62] Zitzler, E., Deb, K., and Thiele, L., "Comparison of Multiobjective Evolutionary Algorithms: Emperical Results," *Evolutionary Computation*, vol. 8, pp. 173–195, June 2000. 1.5, 4.2

[63] Zitzler, E., Laumann, M., and Thiele, L., *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. ETH Zentrum, Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001. 2.2.6, 4.1, 4.1.2.5

[64] Zitzler, E. and Thiele, L., "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strenth Pareto Approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 257–271, November 1999. 2.2.5, 4.1

VITA

Greg Rohling completed his Bachelors in Electrical Engineering from Georgia Tech in 1988, and his Masters also in Electrical Engineering from Georgia Tech in 1989. After graduation he began work with Georgia Tech Research Institute (GTRI), in the development of simulations for the countermeasures for missile threats. From 1992 to 1995 he worked for a small company, TimePlus, in the development of real-time data acquisition for radar data. In 1995 he rejoined GTRI continuing in variety of programs requiring simulation of IR threats and countermeasures. He enrolled as a part-time student at Georgia Tech in 1997. In 1999 he discovered the area of Genetic Algorithms that ultimately resulted in a non-orthogonal alignment of his thesis vector with his work vector.