Neural Enhancement for Multiobjective Optimization

Except where reference is made to the work of others, the work described in this dissertation is my own or was done in collaboration with my advisory committee. This dissertation does not include proprietary or classified information.

Aaron Garrett

Certificate of Approval:

John Hamilton
Associate Professor
Computer Science and Software Engineering

Gerry Dozier, Chair
Associate Professor
Computer Science and Software Engineering

Cheryl Seals
Associate Professor
Computer Science and Software Engineering

Joe F. Pittman
Interim Dean
Graduate School

Neural Enhancement for Multiobjective Optimization

Aaron Garrett

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama
May 10, 2008

Neural Enhancement for Multiobjective Optimization

Aaron Garrett

_____

Signature of Author

_____

Date of Graduation

Vita

Aaron Lee Garrett, son of Jerry Wayne Garrett and Sheila (Reece) Garrett, was born August 11, 1977, in Rome, Georgia. He graduated from Spring Garden High School as salutatorian in 1995. He attended Jacksonville State University in Jacksonville, Alabama, and graduated summa cum laude with a Bachelor of Science degree in Mathematics in December, 1999. He immediately entered into graduate school at Jacksonville State University and received a Master of Science degree in Computer Science in August, 2002. He was then hired by the Computer Science department at Jacksonville State University, where he currently holds a faculty position. He married Ashley Lynn Tawbush, daughter of John and Patricia (Carr) Tawbush, on May 31, 2003. In September of 2004, he was admitted into the Ph.D. program in Computer Science at Auburn University. And on January 17, 2008, he and his wife Ashley became the proud parents of their son Ian Bishop Garrett.

Dissertation Abstract

Neural Enhancement for Multiobjective Optimization

Aaron Garrett

Doctor of Philosophy, May 10, 2008
(M.S., Jacksonville State University, 2002)
(B.S., Jacksonville State University, 1999)

219 Typed Pages

Directed by Gerry Dozier

In this work, a neural network approach is applied to multiobjective optimization problems in order to expand the set of optimal solutions. The network is trained using results obtained from existing evolutionary multiobjective optimization approaches. The network is then evaluated based on its performance against those same approaches when given more processing time. The results are collected from a set of well-known benchmark multiobjective problems, and its performance is evaluated using various indicators from the multiobjective optimization literature.

Preliminary experiments reveal the viability of this approach for expanding the set of solutions to multiobjective problems. Further experiments prove that it is possible to train the neural network in a reasonable time using heuristic methods. The results of this training approach are shown to be very competitive with the underlying evolutionary multiobjective optimization approach that was used to produce the training set. Additional experiments reveal the applicability of this approach across existing multiobjective optimization approaches.

Style manual or journal used Journal of Approximation Theory (together with the style known as "auphd"). Bibliograpy follows van Leunen's *A Handbook forScholars*.

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `auphd.sty`.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimization problems are commonly found in real world applications. However, many optimization problems involve multiple, often conflicting, objectives [1]. These multiobjective optimization problems are often much more difficult to solve than single-objective problems. Typically, solutions to such problems are actually sets of solutions, each of which represents a particular trade-off for the objectives in question [2]. The more elements in such a set, the more options that are available as possible solutions [2].

Many multiobjective optimization techniques have been developed over the years [3], and, most recently, evolutionary computation approaches have been applied with great success [1]. Approaches like the vector evaluated genetic algorithm [1], the multiobjective particle swarm optimizer [4], the nondominated sorting genetic algorithm [5, 6], and ParEGO [7] have all been shown to be effective. However, each of these approaches only produce a small number of nondominated solutions in a given number of function evaluations. Increasing the size of this solution set has proven to be an extremely difficult problem [8]. In this work, a neural network system is described that can, in essence, learn the areas where nondominated solutions are found. In this way, many such solutions can be generated without the need for a large number of function evaluations.

1.1 Multiobjective Optimization

A multiobjective optimization problem is defined as follows [1]:

$$\text{Minimize} \quad [f_1(\vec{x}), f_2(\vec{x}), \cdots, f_k(\vec{x})]$$

subject to the m inequality constraints

$$g_i(\vec{x}) \leq 0 \quad i = 1, 2, \ldots, m$$

and the p equality constraints

$$h_i(\vec{x}) = 0 \quad i = 1, 2, \ldots, p$$

where k is the number of objective functions and $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$. The vector $\vec{x} = [x_1, x_2, \ldots, x_n]$ is referred to as the vector of *decision variables* [1]. We wish to find the values $x_1^*, x_2^*, \ldots, x_n^*$ that yield the optimum values for all the objective functions.

Even with such a clean, mathematical definition, the difficulties that arise in multiobjective optimization are obvious. First and foremost, it is extremely unlikely (except in a few, handpicked problems) that a single assignment of values for the decision variables will yield an optimum value for all the objective functions. Instead, decision variable assignments will represent "trade-offs" in the objective function values. The goal in multiobjective optimization is to present the decision maker with a wide range of possible trade-offs so that the best choice can be made for the situation at hand [2].

1.1.1 An Example

In order to understand the complexities of multiobjective optimization, an example is in order. Suppose the following functions are to be minimized:

$$f_1 = x^2$$

$$f_2 = (x-2)^2$$

$$x \in [0, 2]$$

These objective functions are depicted in Figure 1.1. The best trade-off appears to occur at $x = 1.0$, even though each objective function, if considered individually, would produce minimum values at $x = 0.0$ and $x = 2.0$, respectively. In this case, intuition may be somewhat misleading, depending on our definition of optimality in the context of multiple objectives. When there are multiple possible trade-offs, it is important to clearly define what it means for a solution to be *optimal*. The following subsections discuss three different approaches for such a definition.

1.1.2 Aggregation-based Approach

Single-objective optimization methods have existed for hundreds of years [3]. In order to leverage those methods, one approach for multiobjective optimization is to convert such a problem into a single objective. This is called an aggregation-based approach [2] or an aggregating function [1]. Typically, a weighted sum of the objective values is used as the new scalar optimization function. For example, given objective functions $f_i(x)$ for $i = 1, 2, \ldots, k$,

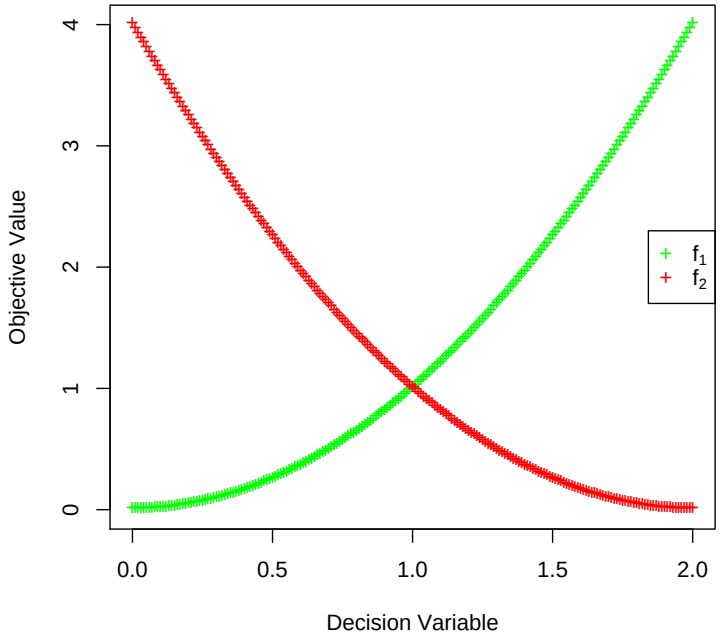


Figure 1.1: Example Multiobjective Problem

the aggregating function $F(x)$ would be

$$F(x) = \sum_{i=1}^{k} w_i f_i(x)$$

where

$$\sum_{i=1}^{k} w_i = 1.$$

Such an approach produces only one solution for a given choice of weights. Therefore, to produce multiple solutions (as is typically required for multiobjective optimization

problems), the process must be repeated many times. Likewise, since good choices for the weight values are often unknown, those values are usually varied before each new run [1].

One of the greatest benefits of using aggregating functions, as mentioned above, is that there are many different single objective optimization methods that can be applied to the transformed problem. However, there are drawbacks. One is, of course, the difficulty of choosing appropriate weights for each objective. An even greater difficulty is that these weighted sums can only generate portions of the Pareto optimal front that are convex [9]. If the Pareto optimal front is non-convex, then a weighted-sum approach will fail to find the points that lie on the concave portion of the front [9]. Additionally, an even spread of weights (for multiple runs of the optimizer) will often fail to produce an even spread of points on the Pareto optimal front [9]. In fact, only curves fitting a specific shape will produce an even spread of solutions given an even spread of weights [9]. There are also non-linear aggregation-based approaches that have been used for multiobjective optimization. These approaches are typically not limited by the convexity of the Pareto optimal front but have been criticized by the multiobjective community [10], possibly because of the current emphasis on Pareto preference.

1.1.3 Criterion-based Approach

Another approach to turn multiobjective problems into single objective problems is to consider only one objective at a time. This is referred to as a criterion-based approach [2] or lexicographic ordering [10]. In the simplest case, the objectives are ranked in order of importance. The one that is considered most important is optimized first as a single objective problem. Then, the next objective is optimized without decreasing the quality

of the first. This process is repeated until all objectives have been considered. In a more complex evolutionary approach, criterion based methods can be used to vary which objective function will determine the selection of an individual [2]. In such cases, the choice of objectives can be random or probabilistic according to a user-defined set of probabilities, or those probabilities can be varied during the run of the algorithm [2].

1.1.4 Pareto-based Approach

Since solutions to multiobjective optimization problems actually represent trade-offs between some objectives in favor of others, there is no clear way to define what is meant by a "good" solution. To overcome this difficulty, another type of metric is used, known as *Pareto preference* [10]. Using this metric, a solution s_1 is said to *dominate* another solution s_2 if s_1 is no worse than s_2 in any objective and if s_1 is strictly better than s_2 in at least one objective. In mathematical terms, solution s_1 is said to dominate solution s_2 if and only if

$$\forall i, 1 \leq i \leq k, f_i(s_1) \leq f_i(s_2)$$

and

$$\exists p, 1 \leq p \leq k, f_p(s_1) < f_p(s_2).$$

The set of all globally nondominated solutions for a particular multiobjective problem is referred to as the *Pareto optimal set* or the *efficient set* [10]. The image of the Pareto optimal set under the objective functions is called the *Pareto optimal frontier* (or often just the *Pareto optimal front*) [10]. Finally, since the global Pareto optimal set is often unknown, most multiobjective optimization approaches actually produce a *Pareto set approximation*

[2]. In this work, the term "Pareto optimal front" is used to refer both to the Pareto optimal frontier as well as the approximation to the Pareto optimal frontier, where the particular meaning will be clear from the context.

1.2 Evolutionary Computation

The beginnings of evolutionary computation (EC) can be traced back as far as the late 1950's, but it was the last two decades that saw truly exponential increases in the amount of attention given to the field [11]. EC can be thought of more as a problem solving strategy than a one-size-fits-all tool [11, 12]. At its core, an EC attempts to mimic the biological process of evolution in order to solve a given problem [13]. This, consequently, suggests that there should be a simulated analog to the biological process of reproduction and some measure of or mechanism for survivability. Indeed, these must be defined for any EC. However, before these processes can be defined, it is necessary first to define what makes up an *individual*.

An individual is simply a *candidate solution*. As such, it can be represented in any way that the designer chooses, and most representations are problem-specific. The manner in which a candidate solution is represented is known as the *genotype* [11] (in keeping with the biological analogy). Since two or more solutions must be compared with one another in order to determine which is "better", it is necessary to also have some measure of "goodness", which in the EC literature is known as the *fitness* of an individual [11]. The fitness is a measure of how well one candidate solution solves a problem, sometimes in comparison to the performance of other candidate solutions. Usually, candidate solutions cannot be compared (and thus their fitness values cannot be assessed) by simply looking at the genotype values.

Instead, the candidate solution undergoes a transformation to convert its genotype into the corresponding *phenotype* [11], which can be thought of as its representation in the fitness space.

To make this discussion more clear, an example is in order. Suppose we wish to find the real value for $-10 \leq x \leq 10$ such that it minimizes the function

$$y = f(x) = x^2.$$

(Clearly, the minimum value will occur when $x = 0$.) In the simplest case, we would say that the x value represents the genotype of the candidate solution, and the set of all candidate solutions is simply the line from -10 to 10. The phenotype for a given x value, in this case, is also that same x value. (In other words, the mapping function between genotype space and phenotype space is simply the identity function.) The phenotype space is simply all of the possible values for x, which in this case is the curve representing $f(x) = x^2$. The fitness for a candidate solution x is simply the value of y at that point, $y = x^2$. These concepts are illustrated in Figure 1.2.

Once the representation is chosen, the *evolutionary operators* must be specified. These operators define the mechanisms of *variation* and *selection* [11]. Variation determines how new solutions are created from existing solutions (i.e., reproduction) and how existing solutions are modified to explore new areas of the search space (i.e., mutation). Selection determines how solutions are chosen to remain viable candidates or to participate in the creation of new solutions. In some cases, one or more of these operators are simply identity functions, which means that they have no actual function. However, when viewed in this way, all of the EC approaches are simply variations on this basic theme [11].

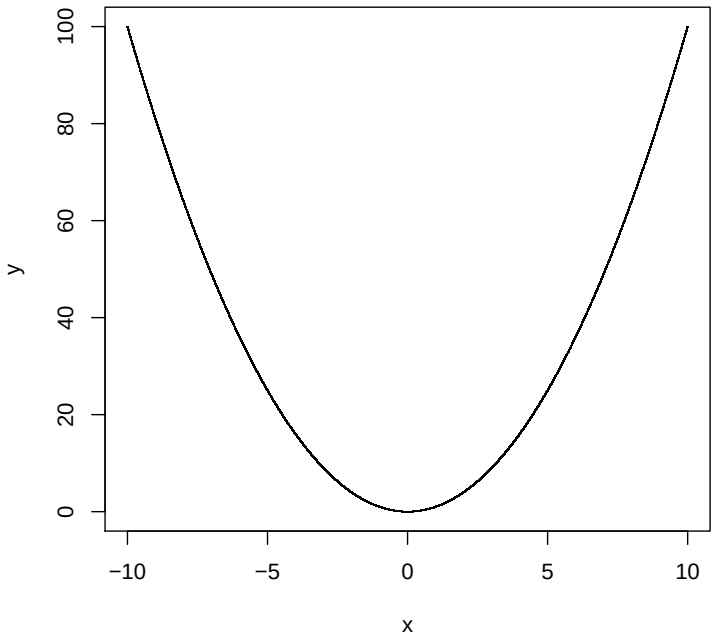


Figure 1.2: Example Minimization via Evolutionary Computation

The basic EC is presented in Listing 1.1. Here, P_t is a set of candidate solutions (the *population*) at time t. Typically, the population is of a fixed size. The `Evaluate()` function performs the mapping from genotype to phenotype space and assigns appropriate fitness values to each individual in the population. The `Variation()` function takes the current population and produces a set of new individuals, P'_t. This function need not make use of all of the individuals in the current population. Often, only a subset are chosen (internal to the function) and used for variation. Finally, the `Select()` function performs the selection of individuals (chosen from the union of the original and modified individuals) which will make up the population in the next generation.

```
ALGORITHM EvolutionaryComputation()
   t ← 0
   Initialize(P_t) // P_t represents the population at time t
   Evaluate(P_t)
   functionEvaluations ← Size(P_t)
   WHILE functionEvaluations < MAX_FUNCTION_EVALUATIONS LOOP
      P'_t ← Variation(P_t) // P'_t represents the offspring
                            // of the population at time t
      Evaluate(P'_t)
      functionEvaluations ← functionEvaluations + Size(P'_t)
      P_{t+1} ← Select(P_t ∪ P'_t)
      t ← t + 1
   END LOOP
```

Listing 1.1: Basic Evolutionary Computation

An important point of which to take note here is that the number of *function evaluations* is critical when comparing two evolutionary computation algorithms. A function evaluation is simply one mapping from genotype to fitness. (The mapping is often called the *evaluation function* or *fitness function*, so the term "function evaluations" is appropriate.) That is the typical processing unit used in the EC literature when referring to the efficiency of a particular algorithm.

According to Bäck et al [11], the majority of current evolutionary computation implementations come from three different, but related, areas – genetic algorithms [14–17], evolutionary programming [11, 18, 19], and evolution strategies [19, 20]. However, as stated earlier, each area is defined by its choice of representation and/or operators. These choices are discussed in the following sections.

1.2.1 Genetic Algorithms

Genetic algorithms were first proposed by John Holland in the 1960's [14] in an attempt to leverage the power of natural selection to solve difficult optimization problems. In the canonical genetic algorithm, sometimes referred to as the *simple genetic algorithm* (SGA) [15, 17], the genotype for a candidate solution is represented as a binary string. The evaluation function is then a mapping from a binary string to some real-valued measure of fitness (as determined from the phenotype). As in all EC approaches, the fitness function is entirely problem-dependent.

The selection operator used in an SGA is *fitness proportionate selection* [15], sometimes called "roulette wheel" selection. In this type of selection, the probability of selecting a particular individual increases proportional to that individual's fitness. In some sense, the size of the individual's "roulette wheel section" increases with its fitness. In this way, the more fit an individual, the more likely it is to survive. However, this operator is stochastic, so it is possible that even the strongest individual will not survive (and, likewise, the weakest individual may survive).

A similar yet slightly more complex selection procedure is *remainder stochastic selection* [15]. In this type of selection, any individual with an above average fitness is guaranteed to participate in creating the next generation of individuals, and the remaining slots are randomly assigned to individuals in a manner proportional to how well their fitness values compare to the average fitness. In this setup, the objective fitness for an individual (as determined by the fitness function) is modified to produce a relative fitness value as follows:

$$f_i^r = \frac{f_i}{\bar{f}}$$

where f_i^r is the relative fitness of individual i, f_i is the objective fitness of individual i, and $\bar{f}$ is the average fitness of the current population.

Another popular selection method is *tournament selection* [15]. In this type of selection, a subset of candidate solutions is chosen from the existing population against each of which the individual in question is compared (as if that individual is competing in a single-elimination tournament against all individuals in the subset). If the individual's fitness is better than all candidate solutions in the subset, then that individual is selected. The size of the subset (often called the *tournament size* [15]) can be used to control the *selection pressure* [15].

For instance, suppose that a particular individual has a relative fitness of 2.36. Then that individual automatically gets 2 copies (from the whole part of the relative fitness) that are allowed to participate in the reproduction phase. Additionally, the individual has a 36% chance of being selected again to fill any remaining slot. In contrast, and individual with a relative fitness of 0.75 would have no guaranteed copies in the reproduction phase, but it would have a 75% chance to fill any remaining slots.

After selection is performed, the participating individuals undergo variation through crossover and mutation. Each of these operators is performed according to some probability of occurrence (typically denoted p_c and p_m, respectively) that must be specified as parameters to the system. The variation operators used in an SGA are single-point crossover [15] and bit-flip mutation [15]. In single-point crossover, two individuals (i.e., binary strings) are chosen, along with a single recombination point that determines the position in each string that will be "cut". The individuals are then recombined at that point to form two new individuals. This can be understood more clearly in the following example (where the

vertical bar represents the recombination point):

Parent A:	XXXXXXX | XX
Parent B:	YYYYYYY | YY
Child 1:	XXXXXXXYY
Child 2:	YYYYYYYXX

This operation is applied to randomly selected parents with probability p_c, which is typically set to be a fairly high (e.g., 0.75) value. Bit-flip mutation simply means that each bit in a newly created binary string is changed to the opposite value with probability p_m, which is typically set to be a very low (e.g., 0.01) value.

The resultant population is made up entirely of the newly-created offspring. This is known as *generational replacement* [11], which means that no individuals from the previous generation are allowed to survive to the succeeding generations. This type of selection strategy (where here we mean *survivor* selection instead of *parent* selection) can be augmented with *elitism* [11], which would allow some proportion (as determined by system parameters) of the most fit individuals to survive into the next generation. Additionally, some genetic algorithms make use of *steady-state replacement* [11], in which only one offspring is created in a given generation, and this offspring always replaces the least-fit individual in the current population.

1.2.2 Evolutionary Programming

In the early 1960's, Lawrence Fogel attempted to use simulated evolution, which he called Evolutionary Programming (EP), to create artificial intelligence [18, 19]. In this

seminal work, finite state machines (FSMs) were evolved to predict future symbols from a given input stream [19]. Using a FSM representation of the individuals in the population required novel variation operators. The following operators were used in the work: changing an output symbol, changing a state transition, adding a state, deleting a state, and changing a state. The fitness of a given FSM was determined by how accurate its predictions were, given the sequence of input symbols. More recently, EP approaches have been applied to real-valued, continuous optimization problems, but these approaches are similar to the approaches used in Evolutionary Strategies [19], which are discussed below.

1.2.3 Evolution Strategies

At the same time that Holland was developing the genetic algorithm, Rechenberg was independently discovering a technique for using natural selection for optimization problems, which he termed *evolution strategies* [20]. The simplest version of an evolution strategy (ES) is what is known as a *two-membered ES* [20] or, more commonly, a (1+1)-ES. In this scenario, a single individual, represented as a vector of real values, comprises the population. At each generation, that individual is mutated (the variation operator) along each dimension using a Gaussian distribution with zero mean and a given variance (provided as a parameter to the system) to produce an offspring. The fitness values for both the parent and the offspring are compared, and the better of the two individuals is allowed to become the parent in the next generation.

It was discovered [20] that online adjustment of the mutation rate (i.e., the variance of the normal distribution) could provide better performance. This online adjustment is known as the $\frac{1}{5}$ *success rule* [20], which states that around $\frac{1}{5}$ of the mutations should be

successful. If the actual number of successful mutations is greater than $\frac{1}{5}$, increase the variance. If the number is less than $\frac{1}{5}$, decrease the variance.

In addition to online adjustment of the variance, more sophisticated versions of evolutionary strategies can also include the particular variance as a part of the genotype to be evolved [20]. It is also possible to evolve and use a different variance along each dimension of the problem [20], thus allowing the search for a solution to conform more appropriately to the topology of the search space. When variances are included in the genotype, an additional parameter is needed to serve as the variance used to mutate the evolved variances.

The (1+1)-ES did not truly make use of the idea of a population of individuals, so this concept was generalized and extended to yield the $(\mu + 1)$-ES [20]. In this system, a population of μ individuals is maintained in each generation. Additionally, a reproduction operator is included that selects two (or more) individuals from this population and recombines them to form a new individual. This recombination is simply a random selection of each component from the parents. Once the new individual is created, it undergoes mutation as mentioned above. Finally, each offspring is added to the population if it is better than the least fit individual, producing the new population for the next generation. This approach can be and has been [20], of course, extended to a $(\mu + \lambda)$-ES, in which μ individuals produce λ offspring. The best μ individuals of the $\mu + \lambda$ individuals are then chosen to survive.

It is also possible to provide somewhat of an analog to the generational replacement of a GA within an ES. This approach is known as a (μ, λ)-ES (where λ must be greater than or equal to μ) [20]. In such a scenario, the μ individuals are used to create λ offspring, and

from those offspring only, μ individuals are chosen to comprise the population in the next generation.

1.2.4 Particle Swarm Optimization

In addition to the evolutionary computation techniques described above, another nature-inspired optimization algorithm has also been applied to multiobjective optimization problems. A technique called Particle Swarm Optimization (PSO) was developed by Kennedy and Eberhart in 1995 [21]. Inspired by the movement of bird flocks and insect swarms, they attempted to develop a model of swarm behavior that could be used to solve optimization problems. To create the analogy, they imagined a flock of birds flying around in search of a corn field. Each bird was capable of remembering the best location it had found, and each was capable of knowing the best location that any of the birds had found. The birds were allowed to fly around while being pulled toward both their individual best locations and the flock's best location. Kennedy and Eberhart found that their simulation of this analogy produced very realistic-looking behavior in their virtual flocks [21].

In the PSO model presented in [21] and expanded in [22], each particle is composed of three vectors: x, p, and v. These represent the particle's current location, best location found, and velocity, respectively. These vectors are each of the same dimensionality as the search space. Additionally, each particle maintains a two values – one corresponding to the fitness of the x vector and the other to the fitness of the p vector.

As the particles in the swarm move around the search space, their velocities are first updated according to the following equation:

$$v_{id} = v_{id} + \varphi_1 R_1(p_{id} - x_{id}) + \varphi_2 R_2(g_{id} - x_{id})$$

In this equation, v_{id} is the velocity of the i^{th} particle along the d^{th} dimension. The g vector represents the best location found by the flock, and R_1 and R_2 are uniform random values chosen from the interval $[0, 1]$. Finally, φ_1 and φ_2 are two constants that control the influence of the personal best and the global best locations, respectively, on the particle's velocity. These values are often referred to as *cognitive* and *social* learning rates, respectively [22].

After the velocity vector is updated, the particle's location is updated by applying the following equation:

$$x_{id} = x_{id} + v_{id}$$

At this point, the new location's fitness is evaluated and compared to the fitness of the particle's personal best location. If the new location is better, then it also becomes the new personal best location for the particle.

The *topology* for a swarm refers to the structure of the neighborhood for each particle. In a *star topology*, all the particles exist in the same neighborhood, so the global best vector represents the best location found by any particle in the swarm. In contrast, a *ring topology* arranges the particles into overlapping neighborhoods of size η. The global best vector in this type of topology represents the best location found by any particle in that particle's neighborhood.

In 1999, Maurice Clerc introduced an improvement to the equation for updating the velocity of a particle [23]. He introduced a constant to be multiplied to the new velocity before updating the location of the particle. He called this constant the *constriction coefficient* [23]. The calculation of this coefficient is as follows:

$$K = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}$$

In this equation, $\varphi = \varphi_1 + \varphi_2$ and $\varphi > 4$. The constriction coefficient is used to restrain the velocity vector of each particle so that it does not grow unbounded.

Finally, various other models have been proposed as alternatives to the so-called full model presented above [24]. The *cognitive-only* model sets φ_1 to 0, while the *social-only* model sets φ_2 to 0. A *selfless* model was also developed which was identical to the social-only model except that a particle's personal best was not included in the search for that particle's neighborhood's global best [24].

1.2.5 Application to Multiobjective Optimization

The first application of evolutionary computation to the problem of multiobjective optimization was Schaffer's Vector Evaluated Genetic Algorithm (VEGA) [25] in 1985 [26]. Since then, evolutionary multiobjective optimization has been an incredibly active area of research [26]. This is due to the natural union of evolutionary computation and multiobjective optimization, which stems from the fact that EC algorithms are generally very good optimizers and they work simultaneously on a set of candidate solutions (i.e., the population) [26]. Since multiobjective optimization problems typically require a set of solutions, rather than just a single solution, the final population from an EC algorithm provides just such a set. The next chapter will discuss in detail several of the most popular evolutionary multiobjective (EMO) algorithms.

1.3 Neural Computation

Also inspired from natural phenomena, neural computation attempts to model the connectionist learning that occurs within the brain in order to produce a computational learning/prediction system [27]. Examples of such systems include simple perceptrons [28, 29], backpropagation feed-forward artificial neural networks [30], radial basis function networks [31], and general regression neural networks [32]. Neural network approaches can accomplish unsupervised tasks, such as clustering and data mining, as well as supervised tasks, such as function approximation and prediction.

1.3.1 Unsupervised Learning

Unsupervised learning [27] simply refers to learning in which the system is not given the "right answer." For example, it is not trained on pairs consisting of an input and the desired output. Instead, the system is given a set of input patterns and is left to find interesting patterns, regularities, or clusterings among them. In these systems, the inputs are clustered together into categories according to how similar they are to the other inputs in the dataset. Inputs that are similar enough (this similarity threshold is generally controlled by parameters to the system) to one another are grouped into the same category. The output from an unsupervised system for a given input is generally the category into which the input was assigned. For this reason, an unsupervised system can be thought of as a mapping from inputs to categories. There are a number of neural network systems that perform unsupervised learning. Two examples include self-organizing maps [33] and Adaptive Resonance Theory neural networks [34].

1.3.2 Supervised Learning

Supervised learning [27] occurs when the learning algorithm is provided with a set of inputs along with the corresponding correct outputs. The learning in this case involves an algorithm that compares the system's output with the correct (supervised) output. Hence, the system is provided with a measure of error. Via many different learning approaches, the system can be modified so as to minimize this error in order to produce correct outputs for given inputs. Many neural network systems, such as backpropagation networks [30] and general regression neural networks [32], make use of supervised learning.

1.3.3 Neural Network Basics

There are millions of neurons within the human brain [35], and each of these neurons is connected to countless others. Thus, among the millions of neurons there are billions of connections. With this arrangement, humans have the ability to learn enormous amounts of information without forgetting things previously learned. In light of the current state of artificial intelligence, this is a truly remarkable feat of evolution and biology. In an effort to simulate such a learning mechanism, computer scientists have developed what are referred to as *neural networks*, which are simply mathematical models, in the hopes that software agents that embed this behavior can perform similarly to their biological counterparts. To more adequately understand the model, however, it is important first to understand the biological mechanism of learning.

Biological Neurons

Each neuron has three basic parts – the *axon*, the *soma*, and the *dendrites* [35]. The axon of a neuron is a long fibrous connector that carries the output of the neuron. The soma is the cell body of the neuron. The dendrites are the receivers of the input from other neurons. Axons are connected to the dendrites of many other neurons. Each axon is separated from its connecting dendrite by a small gap known as a *synapse* [35]. Within the axon, chemicals called *neurotransmitters* are stored [35]. When a signal is to be passed from one neuron to another, a signal is sent from the soma down the axon, which triggers the release of these neurotransmitters. They then flow across the synapse where they bind to receptors on the dendrites of other neurons. If sufficient neurotransmitters are collected by the synapse, a signal is sent to the soma of the receiving neuron [35]. It is important to note that the firing pattern of neurons is "all or none", which means that their output can be considered binary.

When novel inputs trigger learning within neurons, those neurons undergo a change. The efficiency of transmission of signals between neurons is increased or decreased in order to minimize the error [35]. This modulation of the transmission efficiency is accomplished by increasing or decreasing the amount of neurotransmitter found in the synapse between neurons. To illustrate this idea, imagine that each neuron, rather than conducting electrochemical signals, actually conducted water. Each neuron would receive water through its dendrites, and it would transmit water through its axon. (Imagine that there are tiny valves at the end of the axon that connect the neuron to the dendrites of its neighbors.) When learning occurs within a neuron, the flow of water from that neuron to its neighbors is modulated by changing the state of those valves. The more open the valve is, the more

water will be passed on to the neighboring neurons. In this manner, learning may occur within the brain by the changing the strength of signal transmission between neurons.

Artificial Neurons

If each neuron is considered to be an information-processing unit, then the human brain can be thought of as a massively parallel computing machine [35]. Neural networks are software implementations of the neural dynamics and connectivity within the human brain [35]. A neural network is made up of neurons and the weighted connections between them. These weighted connections provide the capability of modulating the signal transmission between neurons. If the weight is high (e.g., the valve is completely open), then the signal transmission from that neuron will be very strong. On the other hand, if the weight is low (e.g., the valve is nearly closed), then the signal transmission from that neuron will be very weak. Thus, by modifying these weights, the network is able to learn to perform a given task.

Neural networks have two major components [35] – a function that determines at which point a neuron should fire, known as an *activation* (or *transfer*) *function*, and a method for updating the weights, often called the *learning rule* for the network. The activation function defines how the neuron responds to the *net input*, which is simply the weighted sum of all inputs into the neuron. For instance, it determines whether or not the neuron should fire given a particular input. The learning rule takes the inputs, the output, and the expected output as parameters. If the expected output is not known, as it would be for unsupervised networks, it is estimated in some way. The function then updates the weights in such a way as to associate the given input with the expected (or estimated) output.

1.4 Neural Enhancement

To perform the neural enhancement [8], a neural network is created using some of the decision variables in the Pareto optimal set to predict the values for the remaining decision variables in the set. In essence, the neural network is used to learn the mapping, in the Pareto optimal set, from some subset of the decision variables to the remaining decision variables. To more clearly understand the approach, consider the following analogy.

In the deserts of the southwestern United States, there are many plateaus and canyons. In the deepest spots of these canyons lay hidden wealth that needed only to be mined. Once news of these riches was released, thousands of people began to flock into the desert in search of their fortunes. Each one knew that wealth was only found in the deepest spots, so they quickly and carefully made their way down from the plateaus into the canyons. When they were done exhausting each mine in an area, they would begin the process again, moving downward if possible and moving towards their neighbors if not. This, as we will see, is somewhat similar to how the EMO approaches operate. Multiple people are searching for wealth in parallel, but each is unable to truly take advantage of the terrain information.

One day, word of the new gold rush reached a young man named Nemo, who also was interested in making his fortune. He studied the reports, paying attention to the locations in which mines had previously been found. After a little consideration, he realized that, perhaps, all of the mines lay along some natural formation. More importantly, the reports had given him enough information to make a crude map of the formation that he could follow. So he set off for the desert and began to search. However, rather than begin on the plateau and work his way down, he began in the canyons (the formation he had discovered) and followed his own map. Since the map was not completely accurate,

he found himself climbing hills from time to time, but overall he was able to search the canyons more efficiently *because he knew how to follow them.* This is similar to the manner in which the neural enhancement system discussed in this work searches for solutions to multiobjective problems.

Chapter 2

Literature Review

This chapter provides the relevant background material from the evolutionary multiobjective optimization (EMO) and neural computation literature.

2.1 Evolutionary Multiobjective Optimization

There have been many different evolutionary approaches to multiobjective optimization proposed in the past two decades [36]. A small subset of these approaches has been chosen for inclusion here. The algorithms presented below either have historical significance or are commonly used at present as EMO techniques. For a more comprehensive survey of EMO approaches, the reader should refer to [1, 2, 10, 26, 36–38].

2.1.1 Vector Evaluated Genetic Algorithm

The first application of evolutionary computation to multiobjective optimization was Schaffer's Vector Evaluated Genetic Algorithm (VEGA) [25]. His approach was to modify the simple-GA to allow for multiple objective values. To do this, the selection procedure was modified so that, given q objective functions, a population of N individuals would generate q subpopulations of size $\frac{N}{q}$, each using a particular objective function as its fitness measure. Then, these subpopulations would be shuffled together to form the new population for the next generation. This approach has been shown [39] to be equivalent to an linear aggregation approach where the weighting coefficients depend on the current population [37]. Additionally, the shuffling of the subpopulations tends to encourage what Schaffer called

"speciation," which means that the algorithm prefers individuals who are particularly good at one or more of the objectives, rather than individuals that represent a balanced trade-off.

2.1.2 Multiobjective Genetic Algorithm

Inspired by the original VEGA concept, Fonseca and Fleming [37] developed a Multiobjective Genetic Algorithm (MOGA) that performs selection based on "rank," where the rank of an individual is defined to be one more than the number of individuals that dominate it. (Thus, all nondominated individuals have rank 1.) Additionally, the authors introduce an application of *fitness sharing* [40] for multiobjective optimization in order to maintain diversity. Fitness sharing in this context means that individuals who are "near" one another must share the resources (i.e., the fitness value) for that area. This implies that highly fit individuals in less crowded areas will be preferred. In order to determine "nearness", a distance metric is employed (usually Euclidean) and a user-defined parameter, σ_{share}, is used to determine at what distance two individuals are considered to be "near enough" to share the resources.

2.1.3 Niched Pareto Genetic Algorithm

In 1994, Horn et al. [41] developed a genetic algorithm for multiobjective optimization that they termed the Niched Pareto Genetic Algorithm (NPGA). It uses a tournament-type selection where two individuals are compared based on whether or not each dominates or is dominated by a set of other individuals in the population. If there is a tie, fitness sharing is used to determine the winner. Increasing or decreasing the size of this comparison set allows the selection pressure to be tuned similar to the tournament size in regular tournament selection. The niching in this algorithm is done through fitness sharing, where a

distance function is used along with a neighborhood kernel (much like that used in a General Regression Neural Network [32]). The fitness of an individual is inversely proportional to the number of individuals "near" it. The algorithm was compared against VEGA [25] on Schaffer's F2 function [25], and it was also applied to a real-world problem of optimum placement of well locations to detect groundwater contamination.

2.1.4 Nondominated Sorted Genetic Algorithm

The nondominated sorting genetic algorithm (NSGA) [5] was first introduced in 1994 by Srinivas and Deb as an evolutionary multiobjective optimizer. The NSGA algorithm operated like a simple genetic algorithm except for the selection mechanism. There, the individuals were ranked according to Pareto preference using multiple passes(i.e., the first pass ranks all truly nondominated points, the second pass ranks all nondominated points from the remainder, etc.). In each pass, the nondominated points are assigned a dummy fitness value and points with the same fitness value undergo fitness sharing.

In 2002, Deb et al. introduced an improvement to NSGA that they termed NSGA-II [6]. NSGA-II uses a sorting routine that runs in $O(MN^2)$ time, rather than $O(MN^3)$ (where M is the number of objectives and N is the population size). This routine is made possible by the use of a new domination operator that relies on the idea of *crowding distance* instead of using fitness sharing. This crowding distance ensures that the solutions adequately fill the objective space. Finally, NSGA-II uses elitism, which allows good individuals to continue to survive and reproduce.

2.1.5 Pareto Archived Evolution Strategy

In [42], the authors present the Pareto Archived Evolution Strategy (PAES) algorithm for multiobjective optimization. The PAES is a (1+1)-ES that uses an archive of Pareto optimal solutions to aid in selection, just like the tournament selection of the NPGA. The archive maintains a list of all current nondominated solutions and stores them in the form of a d-dimensional adaptive grid, where d is the number of objectives. The grid is subdivided recursively and is encoded by a tree structure. In this way, less crowded areas of the grid are favored in the event of a tie (to maintain diversity).

The algorithm is compared with NPGA on five test functions [42], and is shown to compare well, even though it is arguably the simplest algorithm for multiobjective optimization. The results given are primarily qualitative (in the form of graphs for visual comparison), but PAES tends to show less bias than NPGA towards any particular area of the Pareto optimal front, though it is more noisy. The authors describe the conception of the idea as an extension of local search approaches, such as tabu search [43] and simulated annealing [44].

2.1.6 Strength Pareto Evolutionary Algorithm

In 1999, Zitzler and Thiele [45] developed the Strength Pareto Evolutionary Algorithm (SPEA). Their approach differs from those previously discussed in two major ways. First, they use an external archive, much like in PAES, to determine the fitness of individuals in the population. They define the *strength* of a nondominated solution i in the archive to be the number of solutions in the population that are covered (i.e., dominated) by i. Using this definition of the strength of an archive solution, they then define the fitness of all archive solutions to simply be their strengths, and the fitness values of solutions in the population

are calculated by the following function:

$$f_j = 1 + \sum_{i, i \succ j} s_i.$$

Here, f_j represents the fitness of individual j in the population, the operator $\succ$ represents Pareto dominance (so $i \succ j$ means that i dominates j), and s_i represents the strength of archive solution i. Note that in Zitzler's and Thiele's work low fitness values correspond to high reproductive probabilities. This definition of fitness also inherently creates a form of fitness sharing that does not require a niching parameter (like σ_{share} in MOGA). Since solutions that are dominated by only a few points are preferred over those that are dominated by many, the population tends to converge to the Pareto optimal front with a more uniform spread [45].

Zitzler and Thiele also used a clustering approach to keep the size of the external archive bounded and to maintain a uniform distribution of points on the Pareto optimal front (which, as mentioned above, causes the population to produce more uniformly distributed solutions). This was necessary given that the archive is used as part of the selection mechanism. If it were allowed to grow unbounded, then the selection pressure may suffer. The clustering algorithm used in their work was the *average linkage method* [46]. Using this approach, initially each point in the archive represents a cluster. Then, the closest pair of clusters is combined to form a new cluster, thus reducing the number by one. This process is repeated until the desired number of clusters is reached. Afterwards, the individual in each cluster that is nearest the centroid is selected to be retained as that cluster's representative.

The authors compare the performance of the SPEA with that of VEGA, NPGA, NSGA, and Hajela's and Lin's genetic algorithm (HLGA) [47], which uses weighted-sum aggregation

where the weights are encoded into the genotype of the individual. Zitzler and Thiele show that NSGA outperforms the remaining three algorithms (VEGA, NPGA, and HLGA) on the 0/1-knapsack problem, but that SPEA outperforms all approaches, covering 100% of the solutions found by the other approaches [45].

In 2002, Zitzler et al. produced an enhanced version of this algorithm, which they called SPEA2 [48]. In this algorithm, the strength of a solution is calculated as in SPEA, except that it is calculated for all members of both the population and the archive. Additionally, the *raw fitness* of an individual is calculated based on the sum of the strengths of its dominators in both the population and the archive (as opposed to SPEA, where only the strengths of the archives were used). The raw fitness for an individual is summed with the density information about that individual (i.e., how crowded the area around that individual is), which is obtained using a variant of the k-nearest neighbor method [49]. This sum represents the individual's fitness.

The updating of the archive is also modified in SPEA2. First, all nondominated individuals in the current generation are placed in the archive for the next generation. If the archive is not yet full, then the dominated individuals in the current generation are added to the archive in a best-fitness-first manner until it is full. If the archive contains too many individuals after the nondominated solutions are included, then a truncation procedure is used (instead of the clustering approach used in SPEA). This procedure essentially iteratively removes the individual that has the minimum distance to some other individual (as defined using the k-nearest neighbor metric) at each stage.

2.1.7 Pareto Evelope-based Selection Algorithm

Corne et al. developed the Pareto Envelope-based Selection Algorithm (PESA) [50] in 2000. This approach also maintains an external archive of nondominated solutions. This archive is to select individuals that will become parents as a part of the variation operator. With probability p_c (a user-defined parameter), two parents are selected from the archive to undergo crossover to produce one offspring, which is then mutated. With probability $(1 - p_c)$, one parent is selected to undergo mutation to create an offspring. The offspring make up the new population, which is incorporated into the external archive at the beginning of each generation. Binary tournament selection is used to select the participating parents according to the lowest *squeeze factor* [50]. A grid system is used to break the objective space into hypercubes, and the squeeze factor of an individual is defined to be the number of individuals in the same hypercube. This provides a method for maintaining diversity in the population, favoring less crowded areas of the Pareto optimal front.

The squeeze factor is also used to maintain a bound on the size of the external archive. Whenever this bound is exceeded, old individuals are removed from the archive according to a highest-squeeze-factor-first strategy. The new nondominated individuals from the population always replace existing members in the archive in the case of an excess. If the squeeze factors for two individuals are the same, ties are broken randomly.

The authors compared their PESA algorithm against SPEA and PAES on six test functions taken from [51]. The results show that PESA was able to outperform the other algorithms on most of the test functions at 1000, 5000, and 20000 function evaluations. While not conclusive, these results suggest that PESA has the capability to compete with some of the best EMO approaches.

2.1.8 Micro-Genetic Algorithm for Multiobjective Optimization

Coello Coello and Pulido [52] developed the first micro-GA for multiobjective optimization. A micro-GA is defined by a small population size (which was between 2 and 5 individuals in [52]) and small numbers of generations per run. After each run, the micro-GA is started over again with a seeded population from some replaceable memory. This process is then repeated as many times as desired.

The micro-GA system in [52] uses an external archive of nondominated points so that no results from any of the runs are lost. They also use a population memory, which represents a smaller archive that has both a non-replaceable and a replaceable portion (with sizes that are user defined). The population memory is used to seed the initial population for each run of the GA. The GA itself also uses elitism such that one nondominated solution in each generation is allowed to survive into the next generation. When a run of the GA is completed, at most two nondominated solutions from the GA's final population are moved into the external archive. If this archive is full, the new solutions replace older solutions using a grid technique similar to that used in PAES [42]. Finally, two (possibly the same two) nondominated solutions are moved into the replaceable portion of the population memory.

Coello Coello and Pulido have shown good results [52] using very small population sizes (approximately 5) and function evaluations (150 to 1500) on five different test problems from the literature. For each problem, they generated the actual Pareto optimal front using exhaustive enumeration and compared it to the results of their micro-GA. Their results were all qualitative (in the form of graphs of the generated fronts), but the micro-GA appeared to perform very well.

2.1.9 Multiobjective Particle Swarm Optimization

Several approaches to multiobjective optimization have made use of particle swarm optimization approaches. In 2002, Coello Coello and Lechuga [4] developed a PSO for multiobjective optimization that operates normally except that the global (or neighborhood) best particle is chosen from a repository of nondominated solutions. The repository is divided into hypercubes (using the values of the objective functions for a particular solution as its coordinates, much like in PAES [4]), and the probability of choosing a particular hypercube (using roulette wheel selection) is inversely proportional to the number of nondominated solutions located there (to help maintain diversity). After a hypercube is selected, a solution from that area is selected randomly from those available in the hypercube. The performance of the MOPSO was compared with PAES and NSGA-II on three test problems from the EMO literature. The authors state that the multiobjective PSO (MOPSO) remained competitive with the other approaches, even though it wasn't necessarily better in all cases, and it required lower computational time to converge to a set of solutions.

In 2004, Coello Coello et al. [53] developed a modified version of their original MOPSO that included a constraint-handling mechanism and a mutation operator to aid in search space exploration. Their approach also used the same adaptive grid technique that was used for the micro-GA [52]. In order to deal with constrained optimization problems, the concept of dominance was modified just like with the micro-GA so that a feasible solution dominates an infeasible solutions, and, if both are infeasible, then the solution with the fewest constraint violations dominates. The additional mutation operator essentially performs random mutation on each dimension of the particle according to a specified mutation rate. However, similar to simulated annealing approaches [44], this operator is applied with

much less frequency as the swarm develops. In this way, the initial swarm is able to more effectively explore the search space while the later swarm is allowed to converge.

Also in 2004, Mostaghim and Teich [54] used a particle swarm approach similar to that used by Coello Coello et al. [4] to determine an initial Pareto optimal front. Once the initial front is generated, each point in the archive is used as the local guide (or neighborhood best) in a subswarm generated around that point. The subswarms are used to fill out the Pareto optimal front to cover any gaps left by the initial swarm. This approach makes use of a post-processing step that uses subswarms to provide a more fully defined Pareto optimal front. The subswarm approach was compared to a Hybrid MOEA [55] on a real-world antenna design problem, and the subswarm was shown to cover the Pareto optimal front much faster than the Hybrid MOEA.

2.1.10 ParEGO

In 2005, Knowles presented a new EMO technique called ParEGO [7], which uses efficient global optimization EGO [56] as a learning system to produce a model of the fitness landscape. This model is necessary when dealing with expensive multiobjective functions (such as very time-consuming experiments or experiments that must be performed using specialized machinery that require trained operators). The EGO system attempts to learn the model of the multiobjective function given some points on the Pareto optimal front. Then, this model can be used in conjunction with an evolutionary algorithm to find minima, which can then be checked using the expensive actual multiobjective optimization function. Any multiobjective evolutionary algorithm can be used in this system, but Knowles described the particular algorithm used to be a population-based steady-state EA using both

crossover and mutation. He compared the results with those produced by NSGA-II on eight different test functions. Only 250 function evaluations were used for each, but ParEGO was shown to perform very well, outperforming NSGA-II in all cases.

2.2 Neural Computation

The following subsections describe in greater detail several modern neural network techniques. Note that this list is in no way exhaustive. For a more comprehensive list of neural network and machine learning approaches, the reader should refer to [49, 57].

2.2.1 Perceptron

Many different types of neural networks have been designed and implemented by researchers over the last 50 years [58]. Each has their particular strengths and weaknesses. Perhaps the easiest feed-forward network to understand is the perceptron neural network [28]. The defining characteristic of a perceptron is that it contains only two layers of neurons – the input layer and the output layer. The simplest form of a perceptron is one that has some finite number of input neurons and only one output neuron. In this neural network, there exist connections from each input to the output. Figure 2.1 displays a perceptron with three input neurons and one output neuron. The output of the j^{th} output neuron of a perceptron is defined as follows [28]:

$$O_j = A\left(\sum_i I_i w_{ij}\right)$$

where O_j is the output of the j^{th} output neuron, $A(\cdot)$ is the activation function, I_i is the i^{th} input to the network, and w_{ij} is the weight between the i^{th} input to the j^{th} output.

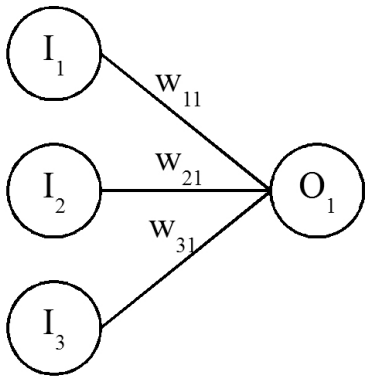


Figure 2.1: Illustration of a Basic Perceptron

The activation function for the perceptron is a simple threshold function [27]. Such a function takes the weighted input to the neuron (i.e., an L_1 magnitude of the product of the input and the weight vector) and determines whether or not this magnitude lies above some threshold value. If so, the neuron fires. Otherwise, the neuron generates no output. While the threshold value could be set by the network designer, in actual neural networks, the value is determined by what is known as a *bias* neuron. A bias neuron is a neuron in each layer of a network that always has a constant input of 1.0. When a bias is used, each neuron is treated as if its threshold value is 0.0. In this way, the *actual* threshold value for any neuron connected to the bias will be determined by the weight from the bias to that neuron. For instance, in Figure 2.1, I_1 could represent the bias neuron, which would mean that it would have a constant input of 1.0. (Note that if such a system is used, the perceptron in Figure 2.1 now only has two genuine input neurons.) In this case, the value of weight w_{11} would represent the threshold value, and this weight would be allowed to change during the learning phase.

To carry out the learning phase, a learning rule must be specified. The learning rule for a perceptron (often called the *Perceptron Learning Rule*) is defined as follows [28]:

$$w_{ij} = w_{ij} + I_i(T_i - O_i)$$

where w_{ij} represents the weight from input i to output j, I_i represents the i^{th} input, T_i represents the correct output (teaching signal) of node i, and O_i represents the actual output of node i. Clearly, this learning rule will move the weight vector closer to the desired output vector given the appropriate inputs.

While the simple perceptron is capable of learning many different types of inputs, its effectiveness has been mathematically proven to be extremely limited [29]. It can only represent vectors that are *linearly separable*, which means that it must be possible to separate different output classes with straight lines (or hyperplanes in higher dimensions). An example of a function that is not linearly separable is the exclusive-or (XOR). This function is displayed in Figure 2.2. In order for this function to be linearly separable, it must be possible to separate the points for which the output is 0 from those where the output is 1 by using only a straight line. The graphical representation in Figure 2.2 makes it clear that no such straight line exists.

2.2.2 Backpropagation Feed-forward Networks

It was discovered [30] that the limitations of the perceptron could be resolved by the introduction of at least one additional layer, a so-called *hidden layer*, between the input and output layers. Figure 2.3 illustrates such a neural network with three inputs, two neurons in the hidden layer (typically referred to as *hidden neurons*), and one output neuron.

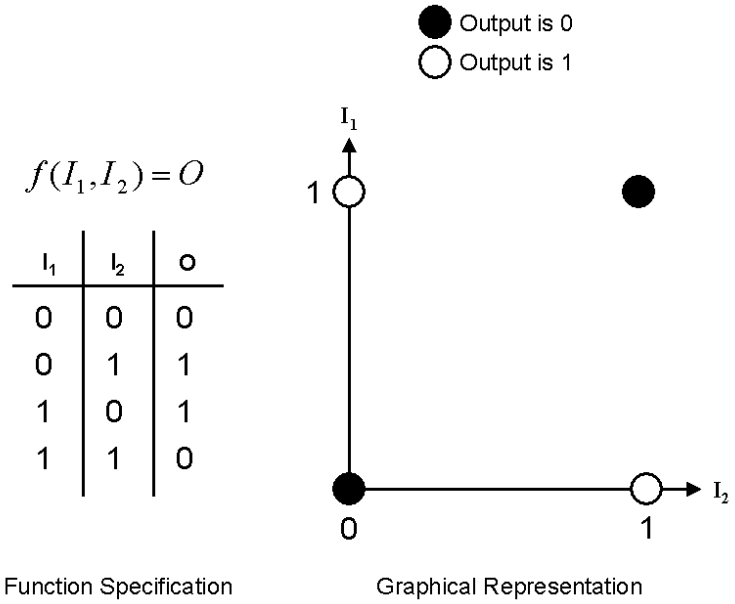


Figure 2.2: Illustration of the Exclusive-Or Function

Because of this additional layer, these types of networks are often referred to as Multi-Layer Perceptrons (MLPs) [49].

However, the introduction of this layer posed problems of its own because the creation of a learning rule for such a network is a non-trivial matter. A learning rule had to be found that could distribute weight changes appropriately across all those connections that led to the error. The most popular such network in use today is known as the *backpropagation network* [59, 60], because the learning rule it applies in essence propagates the blame back through the network to each of the contributing weights as a function of their contribution to the resulting error.

The backpropagation learning rule relies on gradient information in order to manipulate the weight values so as to minimize the prediction error of the network. Since gradient

information is required, it is important that a differentiable activation function be used. In many cases, the *sigmoid* function is used. This function is defined as [60]

$$F(x) = \frac{1}{1 - e^{-x}},$$

and whose first derivative is easily discovered to be [60]

$$F'(x) = F(x)(1 - F(x)).$$

For the output layer, the learning rule is [60]

$$w_{ji} = w_{ji} + \alpha \; a_j \; g'(in_i)(T_i - O_i)$$

where w_{ji} is the weight from hidden node j to output node i, α is the learning rate, a_j is the output of the j^{th} hidden neuron, $g(\cdot)$ represents the activation function, in_i represents the net input to the i^{th} output neuron, T_i represents the correct output (teaching signal) of node i, and O_i represents the actual output of node i.

For the hidden layer, the learning rule is [60]

$$w_{kj} = w_{kj} + \alpha \; I_k \; g'(in_j) \sum_i \left(w_{ji}(T_i - O_i) \; g'(in_i) \right)$$

where w_{kj} is the weight from the k^{th} input neuron to the j^{th} hidden neuron, α is the learning rate, I_k is the input to the k^{th} input node, $g(\cdot)$ represents the activation function, in_i represents the net input to the i^{th} output node, $\sum_i$ is the summation across all neurons

i which make up the layer above (i.e., the output layer) the j^{th} layer, T_i represents the correct output (teaching signal) of node i, and O_i represents the actual output of node i.

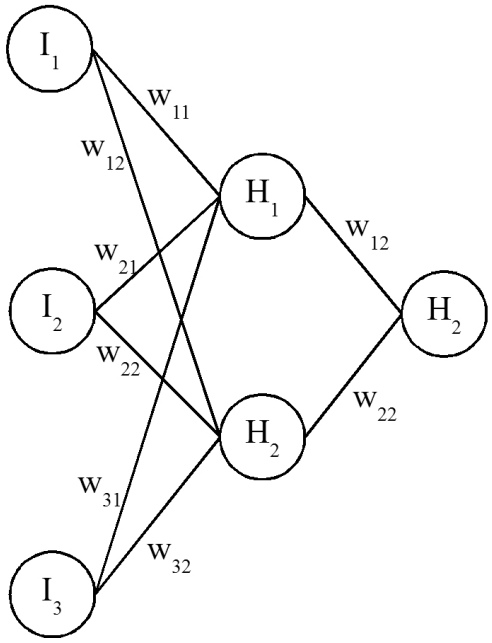


Figure 2.3: Illustration of a Backpropagation Neural Network

2.2.3 Radial Basis Function Networks

Radial basis function networks (RBFNs) [31, 49, 61] are often used as alternatives to backpropagation neural networks [61]. A radial basis function network has three layers – an input layer, a hidden layer, and an output layer. The input layer behaves just like the input layer in an MLP. There are no weights connecting the input and hidden layers in an RBFN. The hidden layer, however, is formed by clustering the input data into some user-defined

number of clusters, where each node of the hidden layer represents one cluster's centroid. A localized receptive field [61] is centered on each of the cluster centroids, which simply means that a significant response is generated from a cluster if the input falls "near" its center in the input space. Finally, the output layer performs a linear combination of this hidden layer activation and the weights between the hidden and output layers.

To make this discussion more precise, consider the following notation. Let $H_i(\cdot)$ denote the receptive field of the i^{th} hidden node, which is centered on c_i. Let w_{ij} denote the weight from hidden node i to output node j. The j^{th} output of the RBFN on a given input x is then

$$y_j = \sum_i (H_i(x) w_{ij})$$

Note that, just as with the perceptron and MLP, a bias node $H_0(\cdot)$ is used which always produces an output of 1 for all inputs.

The most commonly used hidden layer function (i.e., the localized receptive field) is the Gaussian kernel function:

$$H_i(x) = e^{\frac{-||x-c_i||^2}{2\sigma_i^2}}$$

Here, $||\cdot||$ represents Euclidean distance, and σ_i is a smoothing parameter that specifies the "width" of this receptive field. This function is radially symmetric, meaning that the function produces an identical output for each input that lies the same distance from the kernel center. This radial symmetry gave rise to the name "Radial Basis Function Network". (The kernel functions are sometimes called "basis" functions.)

Training an RBFN occurs in two stages [49]. First, the cluster centroids for the hidden layer must be found. This is typically accomplished through an unsupervised clustering

of the input data using a system such as k-means clustering [61]. Once the hidden layer centroids are found, the σ_i value for each cluster is often set to the average distance between the center and the training patterns in the cluster [61]. The second stage of the learning process involves finding the value of the hidden-output weights that minimize the error. This is often done using the Least Mean Squares (LMS) algorithm [61], of which the backpropagation learning rule is an extension.

2.2.4 General Regression Neural Networks

First introduced by Specht in 1991 [32], general regression neural networks (GRNNs) can be viewed as an extension to the k-nearest neighbor approximation using a distance-weighted average and a global neighborhood. A GRNN can also be viewed as Parzen window method [61]. GRNNs are lazy learners [57] since they do not form a model of the training set, choosing instead to store all training instances for future reference. The general form for the output of a GRNN given n training inputs (x_i, y_i), $1 \leq i \leq n$, is

$$F(x) = \frac{\sum_{i=1}^{n} (y_i d(x, x_i))}{\sum_{i=1}^{n} d(x, x_i)}$$

where $d(\cdot)$ is a weighting function, x_i is a training input, and y_i is the desired output for input i.

Commonly, a Gaussian weighting function is used, just as with RBFNs:

$$d(x, y) = e^{\frac{-||x-y||^2}{2\sigma^2}}$$

Once again, $||\cdot||$ represents Euclidean distance, and σ is a smoothing parameter that specifies the neighborhood size. For the GRNN, it is possible for a different smoothing parameter to be used for each training input, but this approach often leads to overfitting (as well as an explosion of parameter values that must be determined). Therefore, typically a single value for σ is used for all training inputs. Training a GRNN consists of merely finding the value of the smoothing parameter σ that provides the minimum error.

2.3 Applications of Learning to Multiobjective Optimization

The author is unaware of any research from the literature that attempts to predict the values of a subset of decision variables in the Pareto optimal set given the remaining decision variables. However, Hatzakis and Wallace [62] used prediction in order to determine the next location to search on the Pareto optimal set in order to find new optimal solutions for time-dependent (i.e., dynamic) multiobjective optimization problems. They used a predictor based on an autoregressive model to predict the next "likely" area to find optimal solutions once the fitness landscape shifts. Their predictor determined two anchor points on the Pareto optimal front using the time-series information of the previous anchor points. The authors mention that "[o]ne might also fit an analytic curve which describes the Pareto optimal set (the locus of the Pareto optimal front in variable space) and subsequently forecast changes in the parameter values of this curve, instead of using specific points".

In similar fashion, Knowles developed ParEGO [7], which uses efficient global optimization (EGO) as a learning system to produce a model of the fitness landscape. In this work, the system attempted to learn the model of an expensive multiobjective function given some points on the Pareto optimal front. This allowed the use of the model to find minima,

which could then be checked using the expensive actual multiobjective function. The results were compared against those produced by NSGA-II on eight different test functions, and ParEGO was shown to perform extremely well, even though only 250 function evaluations were used.

Finally, Gaspar-Cunha and Vieira [63] use a neural network to perform an inverse mapping from objective values to decision variables. In their work, they use an feedforward artificial neural network (using backpropagation) to learn the relationship between the values of the objective functions and the decision variables. Their network is trained using the Pareto optimal front found in the preceding generation, and new points are selected (in the objective space) using a user-defined "offset" from the existing points in the Pareto optimal front. These new points are passed through the neural network to produce new values for the decision variables, which are then tested using the actual multiobjective function. Their results on five test functions from the EMO literature show that this approach can produce a Pareto optimal front using up to 40% fewer function evaluations than a typical EMO approach. (However, in this work, they use their own EMO algorithm, called the Reduced Pareto Set Genetic Algorithm (RPSGA) [64], rather than any of the more popular EMO approaches.)

The work presented in this paper is an extension of Yapicioglu et al.'s work from [65]. It differs from the previously mentioned approaches in that prediction is used to model the Pareto optimal set using a subset of the decision variables. In this way, the most promising regions of the optimal set can be explored in order to more quickly discover nondominated solutions. Rather than being concerned solely with dynamic or expensive multiobjective

problems, this approach is applicable for all multiobjective problems and can be used with any pre-existing set of nondominated solutions, no matter how they are generated.

Chapter 3

Neural Enhancement for Multiobjective Optimization

The neural enhancement approach developed in this work, called Neural Enhancement for Multiobjective Optimization (NEMO), seeks to learn the function that maps a given subset of decision variables in the Pareto optimal set to the remaining decision variables. The basic operation of the NEMO approach is first to allow an existing multiobjective optimization approach to produce a set of solutions to a given problem. Then, that set of solutions is used to train NEMO to learn the mapping among the decision variables. Finally, NEMO can then be used to relieve the burden of producing additional solutions from the underlying multiobjective optimization technique, which is often computationally expensive.

The experiments conducted in this chapter are preliminary experiments to assess the viability of the approach using a well-known evolutionary multiobjective optimization approach to produce the initial set of optimal solutions. For this work, NSGA-II [6] was used to produce a preliminary set of optimal solutions for four benchmark problems from the multiobjective optimization literature, as well as one real-world multiobjective optimization problem. NSGA-II was used because it has been shown [6, 7, 53] to be a very effective evolutionary multiobjective optimization approach. NEMO was then trained using these sets, and the results were compared against the performance of NSGA-II on the same problems using the same number of function evaluations. The following sections describe the experimental setup in more detail.

3.1 Test Suite

In order to provide a thorough test of NEMO, a subset of four multiobjective benchmark problems was taken from [7]. These problems were OKA2, DTLZ1a, DTLZ2a, and DTLZ4a. Finally, the semi-desirable facility location problem (SDFLP) was included from [65] (where it was cited as test case 1.1).

3.1.1 OKA2

The OKA2 test function uses three decision variables to minimize two objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= x_1 \\
f_2 &= 1 - \frac{1}{4\pi^2}(x_1 + \pi)^2 + |x_2 - 5\cos(x_1)|^{\frac{1}{3}} + |x_3 - 5\sin(x_1)|^{\frac{1}{3}} \\
x_1 &\in [-\pi, \pi] \\
x_2, x_3 &\in [-5, 5]
\end{aligned}
$$

The Pareto optimal set lies on a 3D spiral curve.

3.1.2 DTLZ1a

The DTLZ1a test function makes use of six decision variables to minimize two objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= \frac{1}{2}x_1(1+g) \\
f_2 &= \frac{1}{2}(1-x_1)(1+g) \\
g &= 100\left[5+\sum_{i=2}^{6}\left((x_i-0.5)^2-\cos(2\pi(x_i-0.5))\right)\right] \\
x_i &\in [0,1], \quad i \in \{1,\ldots,6\}
\end{aligned}
$$

The Pareto optimal set consists of all decision variables set to 0.5 except the first value, which should come from $[0,1]$.

3.1.3 DTLZ2a

The DTLZ2a test function uses eight decision variables to minimize three objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= (1+g)\cos(x_1\frac{\pi}{2})\cos(x_2\frac{\pi}{2}) \\
f_2 &= (1+g)\cos(x_1\frac{\pi}{2})\sin(x_2\frac{\pi}{2}) \\
f_3 &= \sin(x_1\frac{\pi}{2}) \\
g &= \sum_{i=3}^{8}(x_i-0.5)^2 \\
x_i &\in [0,1], \quad i \in \{1,\ldots,8\}
\end{aligned}
$$

The Pareto optimal front for this function is one-eighth of a sphere centered at the origin with a radius of 1. The Pareto optimal set consists of all decision variables set to 0.5 except the first value, which should come from $[0, 1]$.

3.1.4 DTLZ4a

The DTLZ4a test function uses eight decision variables to minimize three objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= (1+g)\cos(x_1^{100}\frac{\pi}{2})\cos(x_2^{100}\frac{\pi}{2}) \\
f_2 &= (1+g)\cos(x_1^{100}\frac{\pi}{2})\sin(x_2^{100}\frac{\pi}{2}) \\
f_3 &= \sin(x_1^{100}\frac{\pi}{2}) \\
g &= \sum_{i=3}^{8}(x_i - 0.5)^2 \\
x_i &\in [0,1], \quad i \in \{1,\ldots,8\}
\end{aligned}
$$

As with the previous problem, the Pareto optimal front for this function is one-eighth of a sphere centered at the origin with a radius of 1. The Pareto optimal set consists of all decision variables set to 0.5 except the first value, which should come from $[0, 1]$.

3.1.5 SDFLP

The semi-desirable facility location problem deals with positioning a facility (such as an airport or landfill) that is necessary for members of the community. However, the facility itself produces an undesirable by-product (such as traffic or pollution) that the community does not desire. The balance of desirable and undesirable effects leads to the multiobjective

problem.

$$f_1 = \sum_{i=1}^{7} (w_{1i} d(\vec{x}, \vec{a_i}))$$

$$f_2 = \sum_{i=1}^{7} v_i(\vec{x}, \vec{a_i})$$

$$v_i(\vec{x}, \vec{a_i}) = \begin{cases} 200 & \text{if } w_{2i} d(\vec{x}, \vec{a_i}) < 10 \\ 200 - w_{2i} d(\vec{x}, \vec{a_i}) & \text{if } 10 \leq w_{2i} d(\vec{x}, \vec{a_i}) < 30 \\ 0 & \text{if } 30 \leq w_{2i} d(\vec{x}, \vec{a_i}) \end{cases}$$

$$x_1, x_2 \in [-20, 40]$$

$$\vec{a} = ((5, 20), (18, 8), (22, 16), (14, 17), (7, 2), (5, 15), (12, 4))$$

$$\vec{w_1} = (5, 7, 2, 3, 6, 1, 5)$$

$$\vec{w_2} = (1, 1, 1, 1, 1, 1, 1)$$

Since the SDFLP is a real-world problem, its Pareto optimal set is unknown.

3.2 Performance Indicator

To evaluate the performance of NEMO, an indicator was developed that represents the relative sizes of the resultant optimal sets. The *yield ratio* is simply the ratio of the NEMO's solutions versus NSGA-II's solutions. It is a single unit-less value that represents the proportion of NEMO solutions found compared to those of NSGA-II.

3.3 NEMO — The Neural Enhancer

To perform the neural enhancement for each test problem, a general regression neural network (GRNN) [32] was created using a subset of the decision variables in the Pareto set approximation to predict the values for the remaining decision variables in the set. As discussed previously, the GRNN is used to learn the mapping, in the Pareto set approximation, from some subset of the decision variables to the remaining decision variables.

For instance, given a multiobjective optimization problem with 5 decision variables, $d_1, \ldots, d_5$, NEMO first partitions the decision variables into two sets, I and O. For instance, suppose that I contains d_1 and d_3, while O contains d_2, d_4, and d_5. Then, a GRNN is created using elements of I as inputs and producing elements of O as outputs. The GRNN is trained using the elements from the original Pareto set approximation (as found by NSGA-II). Then, values, call them i_1 and i_3, are generated uniformly from the range of each of the elements of I. These values are then passed into the GRNN to produce the corresponding values o_2, o_4, and o_5 in set O. Finally, the generated solution $< i_1, o_2, i_3, o_4, o_5 >$ is passed to the multiobjective function to discover the objective values associated with it. These values determine whether the generated solution is nondominated.

A steady-state genetic algorithm (SSGA) [15] was used to evolve the subset of decision variables to be used as inputs to the GRNN, as well as the value of the GRNN's σ parameter. Each individual in the population was represented by an array of binary values, one for each decision variable representing its inclusion/exclusion as an input to the learning system, and a single real-number value, representing the σ smoothing parameter for the GRNN. The values available for the σ component of the individual was taken from the interval [0.01,

20.0]. The parameters chosen for the SSGA were shown experimentally to provide decent performance and are detailed in Table 3.1.

Parameter	Value
Population Size	100
Recombination Operator	Uniform Crossover
Crossover Usage Rate	1.0
Mutation Operator	Bit-flip (binary) and Gaussian (real)
Mutation Usage Rate	0.1 for both types
Gaussian Mutation Range	1.0
Function Evaluations	1500

Table 3.1: Parameters for Genetic Algorithm for Training NEMO

To evaluate each candidate solution, a GRNN was constructed using the appropriate decision variables and σ value. The GRNN was then trained using the Pareto optimal set for the given test problem. Then, the neural network was presented with 10 randomly generated "inputs" and was asked to predict the corresponding "outputs", which together constituted a possible point in the Pareto optimal set. Each point was then evaluated using the given test problem to produce values for the objective functions. Those values were compared to the values in the Pareto-optimal front to determine whether the point should be added, and, if so, whether it replaced any existing points in the Pareto-optimal front. The fitness for the candidate solution (i.e., the GRNN parameters) was then taken to be

$$fitness = 2 \cdot replaced + added.$$

Here, *replaced* represents the number of solutions in the original Pareto-optimal front that were dominated by the 10 solutions generated by the neural network. Likewise, *added*

represents the number of solutions generated by the neural network that were nondominated in the original Pareto-optimal front. The intention of this fitness measure was to more heavily favor neural networks capable of producing nondominated solutions in the existing Pareto-optimal front. The GA was given a maximum of 1500 function evaluations in order to find the best parameters for the GRNN.

After the appropriate parameters were found, the GRNN was used to predict 10000 values in the Pareto optimal set. To accomplish this, equally spaced points were chosen along the entire range of the GRNN's input dimensions and were passed to the neural enhancer to determine the values along the remaining dimensions. (The distance between points was chosen so that no more than 10000 values would be created.) Each value in the predicted Pareto optimal set was then evaluated using the multiobjective test function. The results are described below.

The implementation used for NSGA-II was taken from the EMOO repository [66] and was implemented in C by Kalyanmoy Deb et. al. It was used without modification except to add the problems from the test suite. It was successfully compiled under Borland version 5.5 and was executed on a Pentium 4 system running Windows XP. For each problem, NSGA-II was initialized with the default parameters as detailed in the software — a population size of 256, a crossover probability of 0.9, a mutation probability of $\frac{1}{n_d}$ where n_d is the number of decision variables, a distribution index for crossover of 10, and a distribution index for polynomial mutation of 50. NSGA-II was allowed to run for 100 generations for each problem in the test suite.

3.4 Results

The following subsections describe the results of applying NEMO to the test suite. These results are summarized in Table 3.2. In this table, the "NEMO Added" column displays the number of solutions that were added by NEMO to the original Pareto optimal front (produced by NSGA-II). Similarly, the "NEMO Replaced" column holds the number of NEMO solutions that replaced at least one solution from the original Pareto-optimal front. Finally, the "Yield Ratio" determines the relative effectiveness of NEMO versus NSGA-II, as described previously.

Problem	NSGA-II Front	NEMO Added	NEMO Replaced	Merged Front	Yield Ratio	Input-Output Assignment	σ
DTLZ1	200	6926	2592	9638	47.6	IOOOOO	8.04
DTLZ2	481	7849	1486	9572	19.4	IIOOOOOO	1.41
DTLZ4	631	9991	9	10626	15.9	OIOOOOOO	3.78
OKA2	15	84	99	39	12.2	OOI	0.01
SDFLP	1259	616	279	2015	0.71	OI	0.61

Table 3.2: Summary of NEMO Results

3.4.1 DTLZ1a

For the DTLZ1a problem, the Pareto optimal front found using NSGA-II, consisting of 200 points, is shown in red in Figure 3.1. A NEMO was created using the first decision variable as input to predict the remaining 5 variables. The σ value was set to 8.04. The results of generating 10000 points using the NEMO are shown in green in Figure 3.1. The GRNN created 9999 non-dominated solutions in the predicted Pareto optimal front (which means that only one of the predicted points was dominated by other predicted points).

These predicted values were merged with the original values found by NSGA-II, allowing the predicted values to dominate or be dominated by the original values, which yielded a Pareto optimal front consisting of 9638 points.

3.4.2 DTLZ2a

For the DTLZ2a problem, the Pareto optimal front found using NSGA-II, consisting of 481 points, is shown in red in Figure 3.2. A NEMO was created using the first and second decision variables as input to predict the remaining 6 variables, using a σ of 1.41. The results of generating 10000 points using the NEMO are shown in green in Figure 3.2. There were 10000 non-dominated points in the final predicted Pareto optimal front. When these points were merged with the original Pareto optimal front, the final Pareto optimal front consisted of 9572 points.

3.4.3 DTLZ4a

For the DTLZ4a problem, the Pareto optimal front found using NSGA-II, consisting of 631 points, is shown in red in Figure 3.3. A NEMO was created using the second decision variable as input to predict the remaining 7 variables, using a σ of 3.78. The results of generating 10000 points using the NEMO are shown in green in Figure 3.3. There were 9999 non-dominated points in the final predicted Pareto optimal front. Notice that the predicted values for the third objective all lie in the range [0.02715, 0.02745]. When the predicted and original values were merged, the final Pareto optimal front consisted of 10626 points.

Although the NEMO was able to generate over 15 times the number of nondominated points that NSGA-II produced, the spread of those points was not sufficient. It was believed

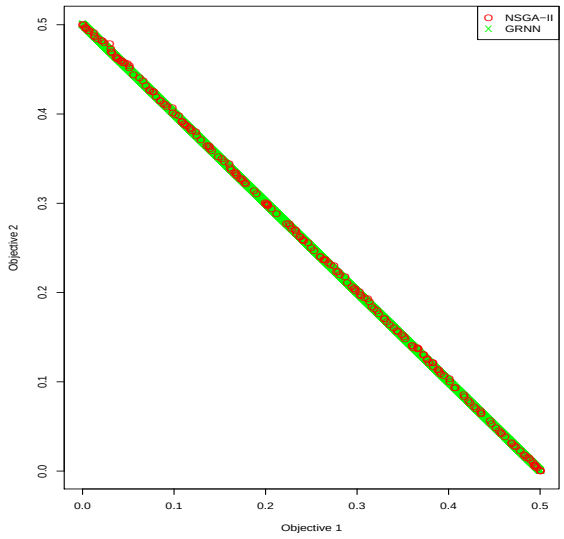


Figure 3.1: Overlay of Pareto optimal fronts for DTLZ1a from NSGA-II and GRNN

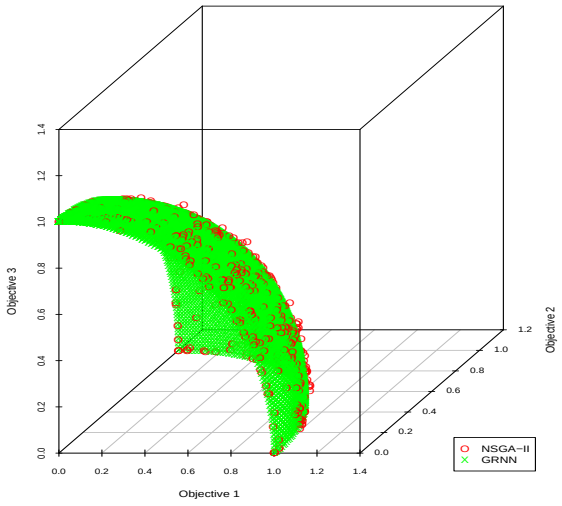


Figure 3.2: Overlay of Pareto optimal fronts for DTLZ2a from NSGA-II and GRNN

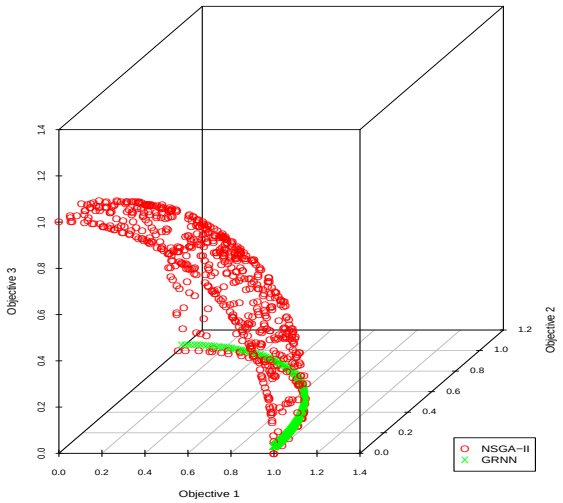


Figure 3.3: Overlay of Pareto optimal fronts for DTLZ4a from NSGA-II and GRNN

that the failure on this problem was that the smoothing parameter (σ) for the GRNN was too large. A further experiment was conducted in which the smoothing parameter was constrained to the interval $[0.001, 0.00000001]$. The result of this approach, using a σ of 0.000081, is shown in Figure 3.4. Here, it is clear that the coverage produced is much better than in Figure 3.3.

3.4.4 OKA2

For the OKA2 problem, the Pareto optimal front found using NSGA-II, consisting of 15 points, is shown in red in Figure 3.5. The neural enhancer was created using the third decision variable as input to predict the first 2 variables, using a σ of 0.01, which was the lower bound used by the GA. The results of generating 10000 points using the neural enhancer are shown in green in Figure 3.5. There were 39 non-dominated points in the final

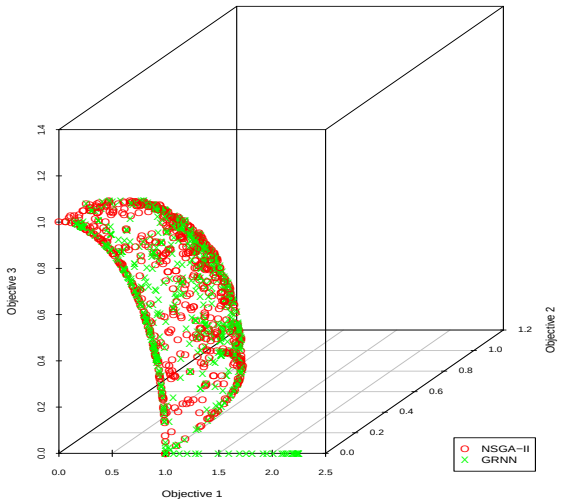


Figure 3.4: Overlay of Pareto optimal fronts for DTLZ4a from NSGA-II and GRNN using small σ value

predicted Pareto optimal front. After the predicted and original values were merged, the final Pareto optimal front also consisted of 39 points.

3.4.5 SDFLP

For the SDFLP problem, the Pareto optimal front found using NSGA-II, consisting of 1259 points, is shown in red in Figure 3.6. The neural enhancer was created using the second decision variable as input to predict the first, using a σ of 0.61. The results of generating 10000 points using the neural enhancer are shown in green in Figure 3.6. There were 3753 non-dominated points in the final predicted Pareto optimal front. After merging the predicted and original values, the Pareto optimal front consisted of 2015 points.

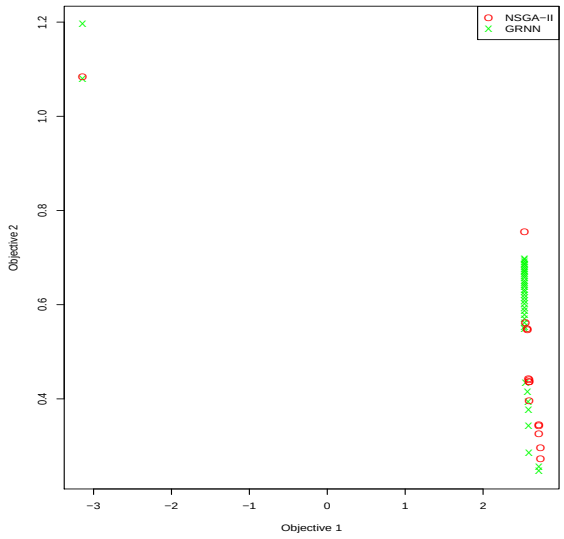


Figure 3.5: Overlay of Pareto optimal fronts for OKA2 from NSGA-II and GRNN

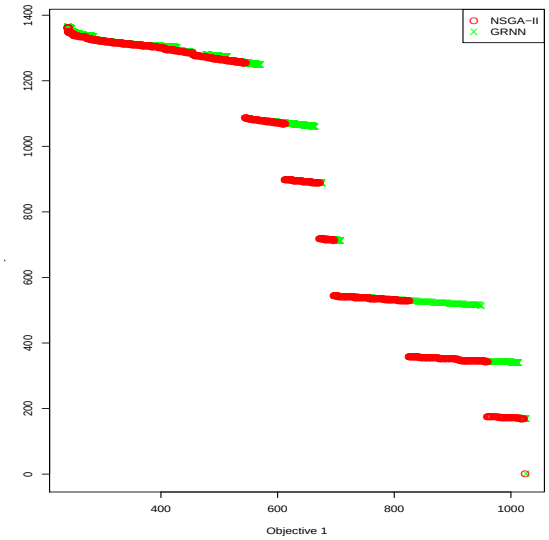


Figure 3.6: Overlay of Pareto optimal fronts for SDFLP from NSGA-II and GRNN

3.5 Conclusions

The neural enhancement approach presented here has been shown to be quite effective at expanding the Pareto optimal frontier for several benchmark multiobjective optimization problems. For each problem, the NEMO approach produced over 10 times more solutions than NSGA-II. However, the real-world problem tested proved more difficult. NEMO was only able to produce about 70% of the solutions that NSGA-II produced. Even so, about 20% of those solutions replaced the solutions produced by NSGA-II.

Given the very promising results presented here, further work must be done to determine their statistical significance. An extension of this study can also be made for problems having more than three objectives. Additionally, work should be done to find a more appropriate training approach for the GRNN, either through a better-tuned evolutionary computation system or through a different training method such as leave-one-out training. In this way, an acceptable smoothing parameter can be found more quickly, which will provide greater performance for the neural enhancer. In addition to a more efficient training algorithm, a computational time comparison should be made between NEMO and NSGA-II.

Chapter 4

Neural Enhancement Training Algorithm

The results achieved in the previous chapter are promising. However, one of the most important criticisms of the NEMO approach is the training time required to achieve success. (This was a criticism discovered in the peer-reviewed comments for [8].) In order to improve on this shortcoming, several experiments were conducted to determine whether a training approach exists that could generate good results in an acceptable time.

To more accurately evaluate NEMO's performance using the various training approaches, several modifications were made to the experimental setup described in the previous chapter. First, five additional multiobjective optimization problems were added to the test suite to provide a more thorough testbed for the NEMO approach. Second, the comparison between NEMO and NSGA-II was modified so as to be as fair as possible to NSGA-II. Previously, the results achieved by NEMO were compared against the original optimal solutions found by NSGA-II. This seemed unfair because NEMO was able essentially to benefit from the work already done by NSGA-II, giving it a "head start" toward producing optimal solutions. To remedy this, NSGA-II was executed for a given set of function evaluations, at which point the current optimal set found was stored and used to train NEMO. NSGA-II was then allowed to execute for an additional set of function evaluations (the same number given to NEMO). In this way, NEMO and NSGA-II could be compared from the same baseline set of optimal solutions. Finally, four additional indicators were used to compare the results of NEMO and NSGA-II to more clearly understand the strengths of their performances.

4.1 Test Suite

In order to test the neural enhancement system and the training approaches, all nine multiobjective benchmark problems were taken from [7]. These problems were KNO1, OKA1 [67], OKA2 [67], VLMOP2 [68], VLMOP3 [68], DTLZ1a, DTLZ2a, DTLZ4a, and DTLZ7a. Finally, the semi-desirable facility location problem (SDFLP) was once again included from [65] (where it was cited as test case 1.1).

4.1.1 KNO1

The KNO1 test function makes use of two decision variables to minimize two objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= 20 - r\cos(\phi) \\
f_2 &= 20 - r\sin(\phi) \\
r &= 9 - \left[3\sin\left(\frac{5}{2}(x_1 + x_2)^2\right) + 3\sin\left(4(x_1 + x_2)\right) + 5\sin\left(2(x_1 + x_2) + 2\right)\right] \\
\phi &= \frac{\pi}{12}(x_1 - x_2 + 3) \\
x_1, x_2 &\in [0, 3]
\end{aligned}
$$

The Pareto optimal set consists of all pairs of decision variable values that sum to 4.4116.

4.1.2 OKA1

The OKA1 test function makes use of two decision variables to minimize two objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= x_1' \\
f_2 &= \sqrt{2\pi} - \sqrt{|x_1'|} + 2\left|x_2' - 3\cos(x_1') - 3\right|^{\frac{1}{3}} \\
x_1' &= \cos\left(\frac{\pi}{12}\right) x_1 - \sin\left(\frac{\pi}{12}\right) x_2 \\
x_2' &= \sin\left(\frac{\pi}{12}\right) x_1 + \cos\left(\frac{\pi}{12}\right) x_2 \\
x_1 &\in \left[6\sin\left(\frac{\pi}{12}\right), 6\sin\left(\frac{\pi}{12}\right) + 2\pi\cos\left(\frac{\pi}{12}\right)\right] \\
x_2 &\in \left[-2\pi\sin\left(\frac{\pi}{12}\right), 6\cos\left(\frac{\pi}{12}\right)\right]
\end{aligned}
$$

The Pareto optima (i.e., the values in the efficient set) lie on the curve

$$x_2' = 3\cos(x_1') + 3, \qquad x_1' \in [0, 2\pi].$$

4.1.3 OKA2

The OKA2 test function uses three decision variables to minimize two objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= x_1 \\
f_2 &= 1 - \frac{1}{4\pi^2}(x_1 + \pi)^2 + |x_2 - 5\cos(x_1)|^{\frac{1}{3}} + |x_3 - 5\sin(x_1)|^{\frac{1}{3}} \\
x_1 &\in [-\pi, \pi] \\
x_2, x_3 &\in [-5, 5]
\end{aligned}
$$

The Pareto optimal set lies on a 3D spiral curve.

4.1.4 VLMOP2

The VLMOP2 test function uses two decision variables to minimize two objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= 1 - \exp\left(-\sum_{i=1}^{2}\left(x_i - \frac{1}{\sqrt{2}}\right)^2\right) \\
f_2 &= 1 - \exp\left(-\sum_{i=1}^{2}\left(x_i + \frac{1}{\sqrt{2}}\right)^2\right) \\
x_1, x_2 &\in [-2, 2]
\end{aligned}
$$

The Pareto optima lie on the diagonal passing from $(\frac{-1}{\sqrt{n}}, \frac{-1}{\sqrt{n}})$ to $(\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}})$.

4.1.5 VLMOP3

The VLMOP3 test function uses two decision variables to minimize three objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= 0.5(x_1^2 + x_2^2) + \sin(x_1^2 + x_2^2) \\
f_2 &= \frac{(3x_1 - 2x_2 + 4)^2}{8} + \frac{(x_1 - x_2 + 1)^2}{27} + 15 \\
f_3 &= \frac{1}{x_1^2 + x_2^2 + 1} - 1.1\exp(-x_1^2 - x_2^2) \\
x_1, x_2 &\in [-3, 3]
\end{aligned}
$$

This test function has a disconnected Pareto optimal set.

4.1.6 DTLZ1a

The DTLZ1a test function makes use of six decision variables to minimize two objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= \frac{1}{2}x_1(1 + g) \\
f_2 &= \frac{1}{2}(1 - x_1)(1 + g) \\
g &= 100\left[5 + \sum_{i=2}^{6}\left((x_i - 0.5)^2 - \cos(2\pi(x_i - 0.5))\right)\right] \\
x_i &\in [0, 1], \quad i \in \{1, \ldots, 6\}
\end{aligned}
$$

The Pareto front for this function is the line $f_1 = -f_2$. The Pareto optimal set consists of all decision variables set to 0.5 except the first value, which should come from $[0, 1]$.

4.1.7 DTLZ2a

The DTLZ2a test function uses eight decision variables to minimize three objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= (1+g)\cos(x_1\frac{\pi}{2})\cos(x_2\frac{\pi}{2}) \\
f_2 &= (1+g)\cos(x_1\frac{\pi}{2})\sin(x_2\frac{\pi}{2}) \\
f_3 &= \sin(x_1\frac{\pi}{2}) \\
g &= \sum_{i=3}^{8}(x_i - 0.5)^2 \\
x_i &\in [0,1], \quad i \in \{1,\ldots,8\}
\end{aligned}
$$

The Pareto optimal front for this function is one-eighth of a sphere centered at the origin with a radius of 1. The Pareto optimal set consists of all decision variables set to 0.5 except the first value, which should come from $[0, 1]$.

4.1.8 DTLZ4a

The DTLZ4a test function uses eight decision variables to minimize three objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= (1+g)\cos(x_1^{100}\frac{\pi}{2})\cos(x_2^{100}\frac{\pi}{2}) \\
f_2 &= (1+g)\cos(x_1^{100}\frac{\pi}{2})\sin(x_2^{100}\frac{\pi}{2}) \\
f_3 &= \sin(x_1^{100}\frac{\pi}{2}) \\
g &= \sum_{i=3}^{8}(x_i - 0.5)^2 \\
x_i &\in [0,1], \quad i \in \{1,\ldots,8\}
\end{aligned}
$$

The Pareto optimal front for this function is also one-eighth of a sphere centered at the origin with a radius of 1. The Pareto optimal set consists of all decision variables set to 0.5 except the first value, which should come from $[0, 1]$.

4.1.9 DTLZ7a

The DTLZ7a test function uses eight decision variables to minimize three objective functions and is defined as follows:

$$
\begin{aligned}
f_1 &= x_1 \\
f_2 &= x_2 \\
f_3 &= (1+g)h \\
g &= 1 + \frac{9}{6}\sum_{i=3}^{8} x_i \\
h &= 3 - \sum_{i=1}^{2}\left[\frac{f_i}{1+g}(1+\sin(3\pi f_i))\right] \\
x_i &\in [0,1], \quad i \in \{1,\ldots,8\}
\end{aligned}
$$

4.1.10 SDFLP

The semi-desirable facility location problem deals with positioning a facility (such as an airport or landfill) that is necessary for members of the community. However, the facility itself produces an undesirable by-product (such as traffic or pollution) that the community does not desire. The balance of desirable and undesirable effects leads to the multiobjective

problem.

$$f_1 = \sum_{i=1}^{7} (w_{1i} d(\vec{x}, \vec{a_i}))$$

$$f_2 = \sum_{i=1}^{7} v_i(\vec{x}, \vec{a_i})$$

$$v_i(\vec{x}, \vec{a_i}) = \begin{cases} 200 & \text{if } w_{2i} d(\vec{x}, \vec{a_i}) < 10 \\ 200 - w_{2i} d(\vec{x}, \vec{a_i}) & \text{if } 10 \leq w_{2i} d(\vec{x}, \vec{a_i}) < 30 \\ 0 & \text{if } 30 \leq w_{2i} d(\vec{x}, \vec{a_i}) \end{cases}$$

$$x_1, x_2 \in [-20, 40]$$

$$\vec{a} = ((5, 20), (18, 8), (22, 16), (14, 17), (7, 2), (5, 15), (12, 4))$$

$$\vec{w_1} = (5, 7, 2, 3, 6, 1, 5)$$

$$\vec{w_2} = (1, 1, 1, 1, 1, 1, 1)$$

Since the SDFLP is a real-world problem, its Pareto optimal set is unknown.

4.2 Performance Assessment

In the previous chapter, the only performance indicator used was the relative yield, or *yield ratio*. However, it is clear that the yield ratio is only an indicator of the relative sizes of the final Pareto optimal sets. This is not sufficient to adequately describe or compare the resultant Pareto optimal sets and fronts.

To overcome this limitation, five indicators were chosen as measures of success and points of comparison between NEMO and NSGA-II – training time, yield ratio, spacing [69], hypervolume, and binary $\epsilon+$ [70]. These indicators are described in more detail in the following sections. The work presented in [70–73] provide a more thorough discussion of assessment indicators for evolutionary multiobjective optimization.

4.2.1 Training Time

The training time measures the total time required to train NEMO with the given algorithm. This indicator does not take into account the time required to add solutions to the existing Pareto front approximation. This is because doing so would unnecessarily punish NEMO for generating many solutions. Therefore, only the actual training time is taken into account. Additionally, the training time for NSGA-II was not collected. However, all NSGA-II runs were completed in, at most, several seconds. Therefore, NEMO must also be trainable in seconds if it is to be a legitimate alternative. As will be seen, the training time for the GRNN using the best training approach found in these experiments has a complexity of $O(n)$, where n is the size of the training set.

4.2.2 Yield Ratio

As in the previous chapter, the yield ratio was calculated, though its definition was expanded to allow for comparison of algorithms that use different numbers of function evaluations. The *yield* of a multiobjective optimization (MO) algorithm is defined as the ratio of solutions in the Pareto optimal front approximation to the number of function evaluations used by the algorithm. The yield of a MO algorithm is expressed in units of solutions per function evaluation. The *yield ratio* is a binary indicator that is simply the

yield of one MO algorithm divided by the yield of another. For instance, if there are two MO algorithms, A and B, then the yield ratio would be

$$YR(A,B) = \frac{yield(A)}{yield(B)}.$$

In this way, the yield ratio is a unit-less measure of the number of solutions algorithm A produces relative to algorithm B in the same number of function evaluations. For example, a yield ratio of 2.0 means that algorithm A produces twice as many solutions as algorithm B.

4.2.3 Spacing

Veldhuizen and Lamont [69] make use of the spacing indicator to measure the "spread" or distribution of solutions throughout the Pareto optimal front. The indicator S is calculated as follows:

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\bar{d} - d_i)^2}$$

where $d_i = \min_j \left(\sum_{k=1}^{m} \left| f_k^i(\vec{x}) - f_k^j(\vec{x}) \right|\right)$, $i, j = 1, \ldots, n$, m is the number of objectives, $\bar{d}$ is the mean of all d_i, and n is the number of vectors in the Pareto optimal front approximation.

Clearly, each d_i represents the L_1 distance between solution i and the solution j that is closest to i. Likewise, $\bar{d}$ represents the mean of the closest distances for all solutions i. This means that the spacing indicator represents the average deviation from this mean across all solutions in the Pareto optimal set. With this indicator, a value of 0 would imply that the solutions found are all equally spaced. The larger the value, the less uniformly distributed the solutions tend to be.

4.2.4 Hypervolume Indicator

In [70] and [7], the hypervolume indicator is described. This indicator is used to measure the volume subsumed by a given Pareto optimal front. In essence, every solution on a Pareto optimal front approximation can be seen as a lower corner of a hypervolume into which no other Pareto optimal solution may fall. (The other solution would not be Pareto optimal given that the first dominates it.) While this hypervolume is, in fact, infinite in extent, it can be bounded by choosing any point dominated by it. In fact, it is possible [7] to find a single suitable bounding point that can be applied to two different Pareto optimal fronts in order to make their comparison possible.

To find a suitable bounding point, the minimum and maximum values must be found along all dimensions (i.e., objectives) and across all given Pareto optimal front approximations. Then, given the minimum and maximum points, the bounding point $\vec{b}$ is determined as follows:

$$b_i = max_i + \delta(max_i - min_i), \quad i = 1, \ldots, k$$

where k is the number of objectives. For these experiments, δ was set to 0.01. This is presented in algorithmic form in Listing 4.1.

```
ALGORITHM BoundingPoint(front, δ)
   // front is a set of Pareto optimal front approximations to be compared.
   // Each front[i] is a Pareto optimal front approximation.
   // numFronts is the number of fronts in the front set.
   // numSolutions[i] is the number of solutions in front[i].
   // δ is a user parameter that defaults to 0.01.
   min ← CreateArray(numObjectives, 0)
   max ← CreateArray(numObjectives, 0)
   FOR o ← 0 TO numObjectives - 1 LOOP
      FOR f ← 0 TO numFronts - 1 LOOP
         FOR s ← 0 TO numSolutions[f] - 1 LOOP
            soln ← front[f][s]
            IF f = 0 AND s = 0 THEN
               min[o] ← soln.output[o]
               max[o] ← soln.output[o]
            ELSE
               IF soln.output[o] < min[0] THEN
                  min[o] ← soln.output[o]
               ELSE IF soln.output[o] > max[0] THEN
                  max[o] ← soln.output[o]
               END IF
            END IF
         END LOOP
      END LOOP
   END LOOP
   boundingPoint ← CreatePoint(numObjectives, 0)
   FOR i ← 0 TO numObjectives - 1 LOOP
      boundingPoint[i] ← max[i] + δ * (max[i] - min[i])
   END LOOP
   return boundingPoint
```

Listing 4.1: Calculation of Bounding Point for Hypervolume Indicator

4.2.5 Binary $\epsilon+$ Indicator

The final indicator used in this work is the binary $\epsilon+$ indicator [7,70]. This indicator is needed to support the hypervolume indicator [7]. The hypervolume values do not immediately determine which Pareto set is better, and the bounding point chosen to calculate the hypervolumes is, in some sense, arbitrary. In order to define this indicator, it is necessary to make use of a comparison known as ϵ-dominance [7]. Given two solutions a and b, each with k objectives,

$$a \preceq_{\epsilon+} b \equiv a_i \leq \epsilon + b_i \; \forall i = 1, \ldots, k$$

Each binary $\epsilon+$ indicator is actually a pair of numbers (I_A, I_B) calculated as follows:

$$I_A = \inf_{\epsilon \in \Re} \{\forall b \in B \; \exists a \in A : a \preceq_{\epsilon+} b\}$$

$$I_B = \inf_{\epsilon \in \Re} \{\forall a \in A \; \exists b \in B : b \preceq_{\epsilon+} a\}$$

In essence, I_A is the smallest value for ϵ that can be added to every dimension of every solution in B such that some solution is A dominates it. It may be considered something like a "fudge factor" that can ensure that A dominates B. I_B can be interpreted in a similar fashion. (This definition assumes that all objectives are to be minimized. Maximization problems can be handled similarly.)

The results of the binary $\epsilon+$ indicator are conclusive in the event that either I_A or I_B is positive and the other is not. In such a case, the nonpositive element (for instance, I_A) clearly dominates (in other words, A dominates B). However, In the event that both elements I_A and I_B are positive, the indicator is inconclusive. However, the smaller of the

two values can imply a very weak form of dominance given that it takes a smaller "fudge factor" to create a Pareto-dominance situation.

4.3 Experimental Setup

Five different experiments were conducted to determine the effectiveness of various training methods for NEMO. NSGA-II was run five times for each test problem, yielding 50 different Pareto optimal sets/fronts. NSGA-II was initialized with a population size of 256 individuals for each run, and it was given 100 generations to execute (for a total of 25600 function evaluations). At that point, the current Pareto optimal set/front was saved (to be used for training NEMO), and NSGA-II was allowed to execute for an additional 100 generations (for a total of 51200 function evaluations). In this way, it was possible to accurately compare NEMO against NSGA-II because both would be judged on their performance after being given the results of the first 25600 function evaluations from NSGA-II.

As in the previous chapter, the implementation used for NSGA-II was taken from the EMOO repository [66] and was implemented in C by Kalyanmoy Deb et. al. It was used without modification except to add the problems from the test suite. It was successfully compiled under Borland version 5.5 and was executed on a Pentium 4 system running Windows XP. For each problem, NSGA-II was initialized with the default parameters of the software — a population size of 256, a crossover probability of 0.9, a mutation probability of $\frac{1}{n_d}$ where n_d is the number of decision variables, a distribution index for crossover of 10, and a distribution index for polynomial mutation of 50.

4.3.1 Evolve I/O and Evolve Single Sigma

A steady-state genetic algorithm (GA) was used to evolve the subset of decision variables to be used as inputs to the GRNN, as well as the value of the GRNN's sigma parameter. Each individual in the population was represented by an array of binary values, one for each decision variable representing its inclusion/exclusion as an input to the learning system, and a single real-number value, representing the σ smoothing parameter for the GRNN. The values available for the σ component of the individual was taken from the interval [0.01, 20.0] (which was determined experimentally to produce decent performance). The GA was created with a population size of 100, uniform crossover, bit-flip mutation on the binary segment of the chromosome, and Gaussian mutation on the real-coded segment. The crossover usage rate was set to 1.0, the mutation rates for both types of mutations were set to 0.1, and the mutation range for the Gaussian mutation was set to 1.0.

To evaluate a given candidate input/output assignment and σ value, the inputs and outputs were used along with the σ value to construct a GRNN. This GRNN was trained using 90% of the given Pareto optimal set and was then tested on the remaining 10%. (In effect, the first element out of every 10 in the Pareto optimal set was used for testing.) The mean squared error across the test set was used as the fitness value.

4.3.2 Heuristic I/O and Evolve Single Sigma

A very simple heuristic (described in Listing 4.2) was used to determine the particular inputs and outputs that would be used by the GRNN. For each decision variable (i.e., candidate input/output) for a given problem, the variance of that variable in the training

set was calculated. The decision variable with the highest variance was used as the input in order to predict the remaining decision variables.

A steady-state GA was used to evolve the value of the GRNN's σ parameter. Each individual in the population was simply a single real-number value, representing the σ smoothing parameter. As before, the allowed range of the σ value was given to be [0.01, 20.0]. The GA was created with a population size of 100, uniform crossover, and Gaussian mutation. The crossover usage rate was set to 1.0, the mutation rate was set to 0.1, and the mutation range was set to 1.0.

To evaluate a given candidate σ value, the pre-calculated inputs/outputs were used along with the candidate value to construct a GRNN. This GRNN was trained using 90% of the given Pareto optimal set and was then tested on the remaining 10%. The mean squared error across the test set was used as the fitness value.

```
ALGORITHM HeuristicIO(front, N, ioMask)
   variance ← CreateArray(numDecisionVars, 0.0)
   FOR d ← 0 TO numDecisionVars - 1 LOOP
      sum ← 0
      FOR i ← 0 TO N − 1 LOOP
         sum ← sum + front[i].input[d]
      END LOOP
      mean ← sum/N
      sum ← 0
      sumSq ← 0
      FOR i ← 0 TO N − 1 LOOP
         sumSq ← sumSq + (front[i].input[d] - mean)^2
         sum ← sum + (front[i].input[d] - mean)
      END LOOP
      variance[d] ← (sumSq − sum^2)/(N(N−1))
   END LOOP
   largest ← FindLargest(variance)
   Initialize(ioMask, numDecisionVars, -1)
   ioMask[largest] ← 1
```

Listing 4.2: Heuristic Calculation of I/O Values

4.3.3 Heuristic I/O and Evolve Multiple Sigmas

As in the previous section, this experiment used the same heuristic approach to find the inputs and outputs of the GRNN. However, the GRNN here was created with a separate σ smoothing parameter for each element of the training set. Therefore, the evolutionary approach was slightly modified to allow a chromosome of length equal to that of the Pareto optimal set produced by NSGA-II after 25600 function evaluations. As before each σ value

was allowed to range between 0.01 and 20.0, inclusive. And, as before, evaluating a given set of candidate σ values was accomplished by using a 9:1 training/testing ratio and calculating the mean squared error on the test set.

4.3.4 Heuristic I/O and Heuristic Single Sigma

As before, the heuristic approach was used to calculate the inputs and outputs of the GRNN. In this experiment, however, a heuristic approach was also applied to calculating the σ values for the GRNN. This approach is described in Listing 4.3. Essentially, for each point p in the Pareto optimal set, the distance from p to all the other points is calculated. Then, the average distance between p and its K closest neighbors is used as the σ value for element p. This is repeated for all points in the Pareto optimal set. Those calculated σ's are then averaged together to arrive at a single σ for the GRNN.

```
ALGORITHM HeuristicSigma(front, ioMask, N, K, sigma)
    count ← 0
    FOR i ← 0 TO numDecisionVars - 1 LOOP
        IF ioMask[i] THEN
            inputIndex[count] ← i
            count ← count + 1
        END IF
    END LOOP
    FOR i ← 0 TO N - 1 LOOP
        distance ← CreateArray(N - 1, 0.0)
        index ← 0
        FOR j ← 0 TO N - 1 LOOP
            IF i ≠ j THEN
                FOR d ← 0 TO count - 1 LOOP
                    distance[index] ← distance[index] +
                        (front[i].input[inputIndex[d]] - front[j].input[inputIndex[d]])^2
                END FOR
                distance[index] ← √distance[index]
                index ← index + 1
            END IF
        END FOR
        SortAscending(distance)
        FOR j ← 0 TO K - 1 LOOP
            sigma[i] ← sigma[i] + distance[j]
        END FOR
        sigma[i] ← sigma[i] / K
    END FOR
```

Listing 4.3: Heuristic Calculation of σ Values

4.3.5 Heuristic I/O and Heuristic Multiple Sigmas

In this final experiment, the heuristic was used to determine the I/O mask, and the heuristic in the previous section was used to find a σ value for each point in the Pareto optimal set. However, the σ values found were used without being averaged together.

4.4 Results

Figures 4.1-4.6 provide graphical depictions of the results of each training algorithm on each test problem, grouped according to the indicator in question. In these figures, the test suite problems lie along the horizontal axis while the indicator values lie along the vertical axis. Each problem is associated with five vertical bars, representing the performance of each of the training approaches on that problem. In each figure, the results of the SDFLP problem were omitted because the NSGA-II algorithm failed to produce any further solutions when allowed to execute for another 100 generations. As Table 4.1 shows, however, the NEMO approach (with all training approaches) found a great number of solutions in each case for SDFLP compared to the original NSGA-II production (denoted "NSGA-II (half)" in the table).

For each training approach and test problem, the solutions found by NSGA-II and NEMO are plotted for the first of the five runs. In each of these graphs, the NSGA-II solutions are depicted in red while the NEMO solutions are in blue. These solutions are those found only after the initial 100 generations of the NSGA-II algorithm. (This eliminates the "clutter" of solutions that were the common foundation of both approaches and serves to highlight their individual performances.) It is worth noting that these graphs were created

by first plotting the NEMO results and then the NSGA-II results. This means that any visible NEMO points certainly do not obscure any NSGA-II points.

Likewise, the efficient sets produced by NSGA-II and NEMO for the first run on problems KNO1, OKA1, VLMOP2, VLMOP3, and SDFLP are represented graphically for each training approach. Only those five test problems are displayed, since they have only two decision variables. The efficient set predicted by NEMO is represented in black, the NSGA-II solutions in the efficient set are represented in red, and the NEMO solutions are represented in blue.

File	NSGA-II (half)	EIO-ESS	HIO-ESS	HIO-EMS	HIO-HSS	HIO-HMS
1	1502	5427	5297	12544	5439	6349
2	1482	6747	6136	12472	5231	6356
3	1510	6105	6130	12630	6046	6564
4	1498	6694	6694	12689	5440	6251
5	1519	10423	7320	12466	5933	7195
Average	1502.2	7079.2	6315.4	12560.2	5617.8	6543
St. Dev.	13.86	1944.13	751.04	97.95	352.1	381.91

Table 4.1: NEMO Solutions for SDFLP Compared with NSGA-II (half)

4.4.1 Evolve I/O and Evolve Single Sigma

The results of this approach on the first run are depicted in Figure 4.7. Figure 4.8 shows the efficient sets produced for this training approach. The results from all five runs are provided in Appendix A as Table A.1.

The most relevant and important indicator for this approach is training time (Figure 4.1). Evolving I/O and sigma values takes much more time than the other approaches, especially for DTLZ2a and DTLZ4a, requiring an average of more than 20 minutes to train.

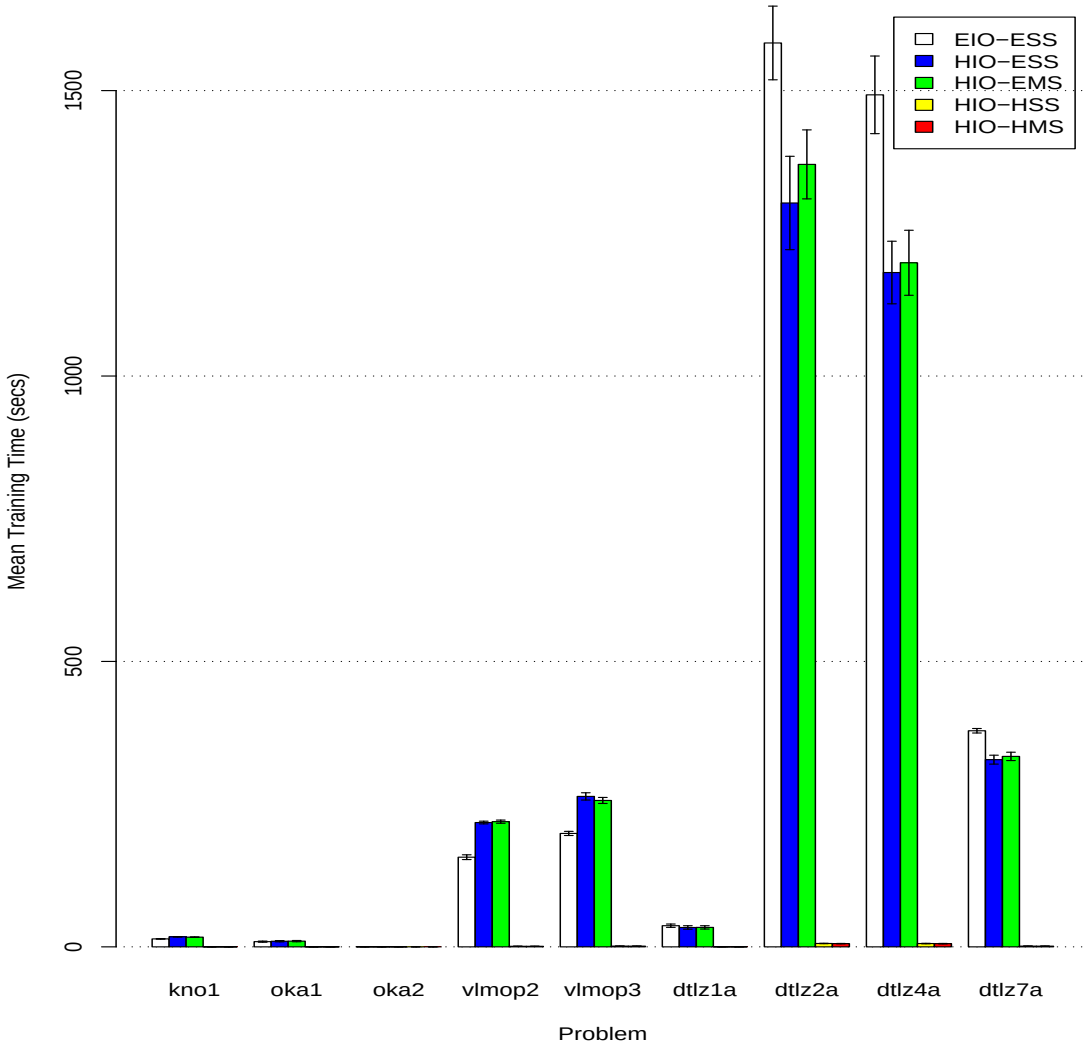


Figure 4.1: Comparison of Training Method Impact on Training Time with Standard Error Bars.

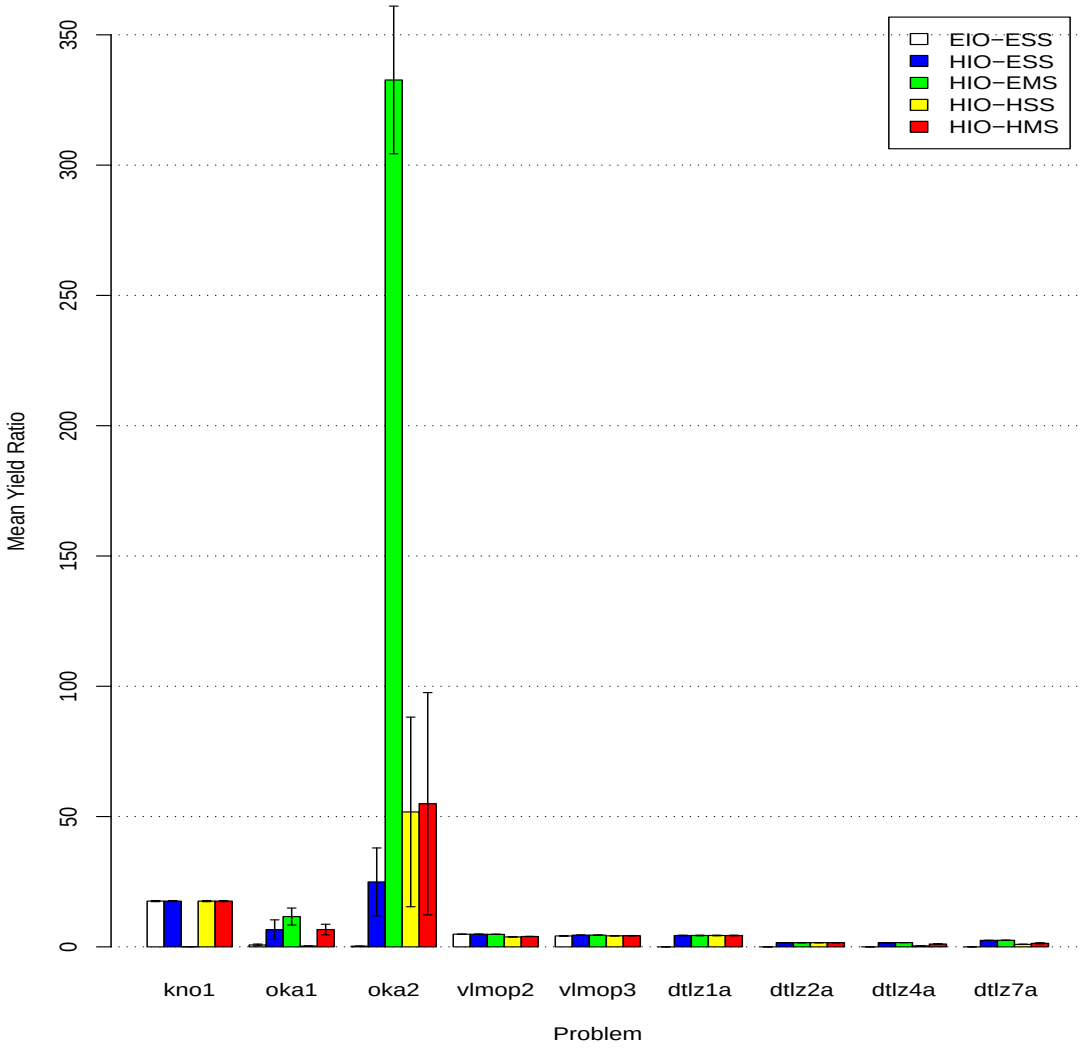


Figure 4.2: Comparison of Training Method Impact on Yield Ratio with Standard Error Bars.

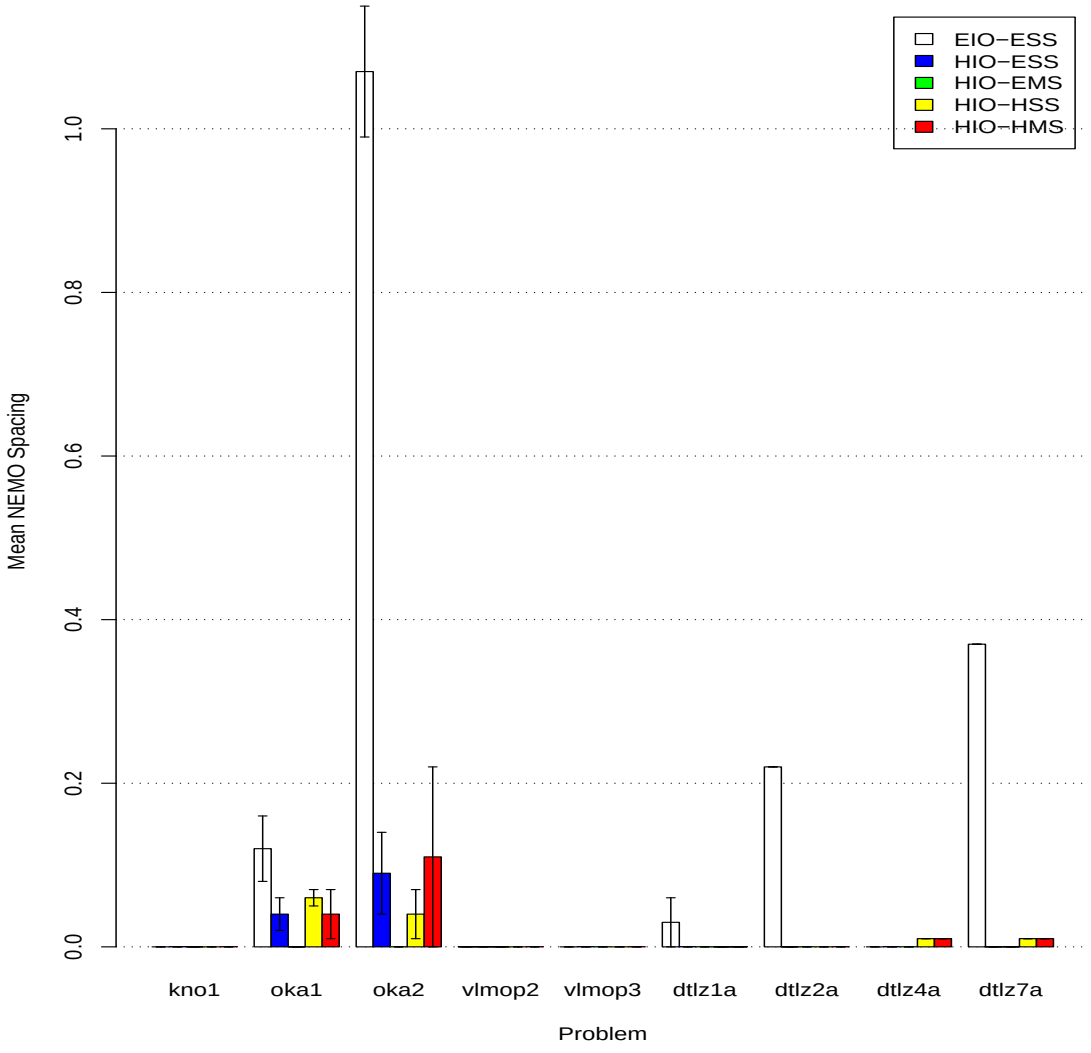


Figure 4.3: Comparison of Training Method Impact on NEMO Spacing with Standard Error Bars.

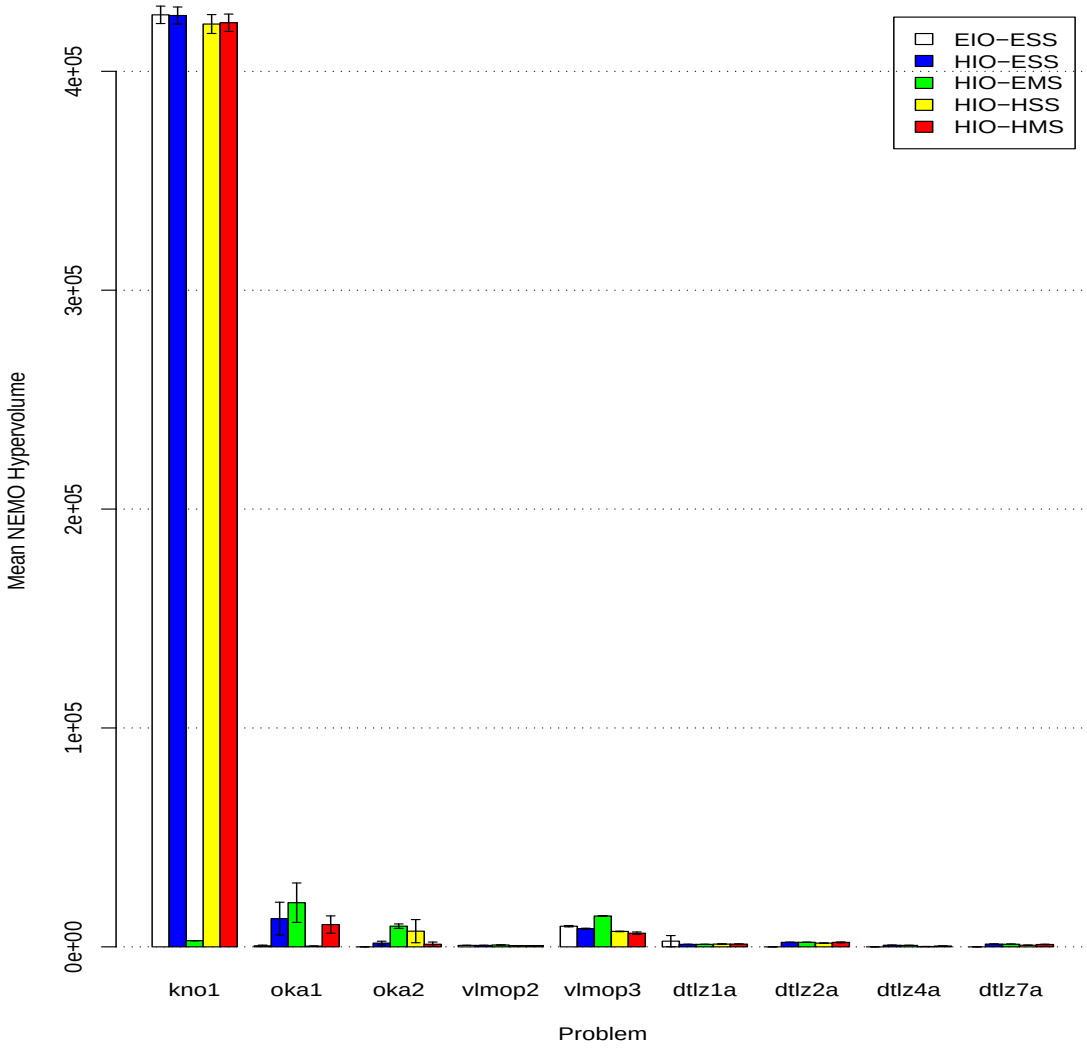


Figure 4.4: Comparison of Training Method Impact on NEMO Hypervolume with Standard Error Bars.

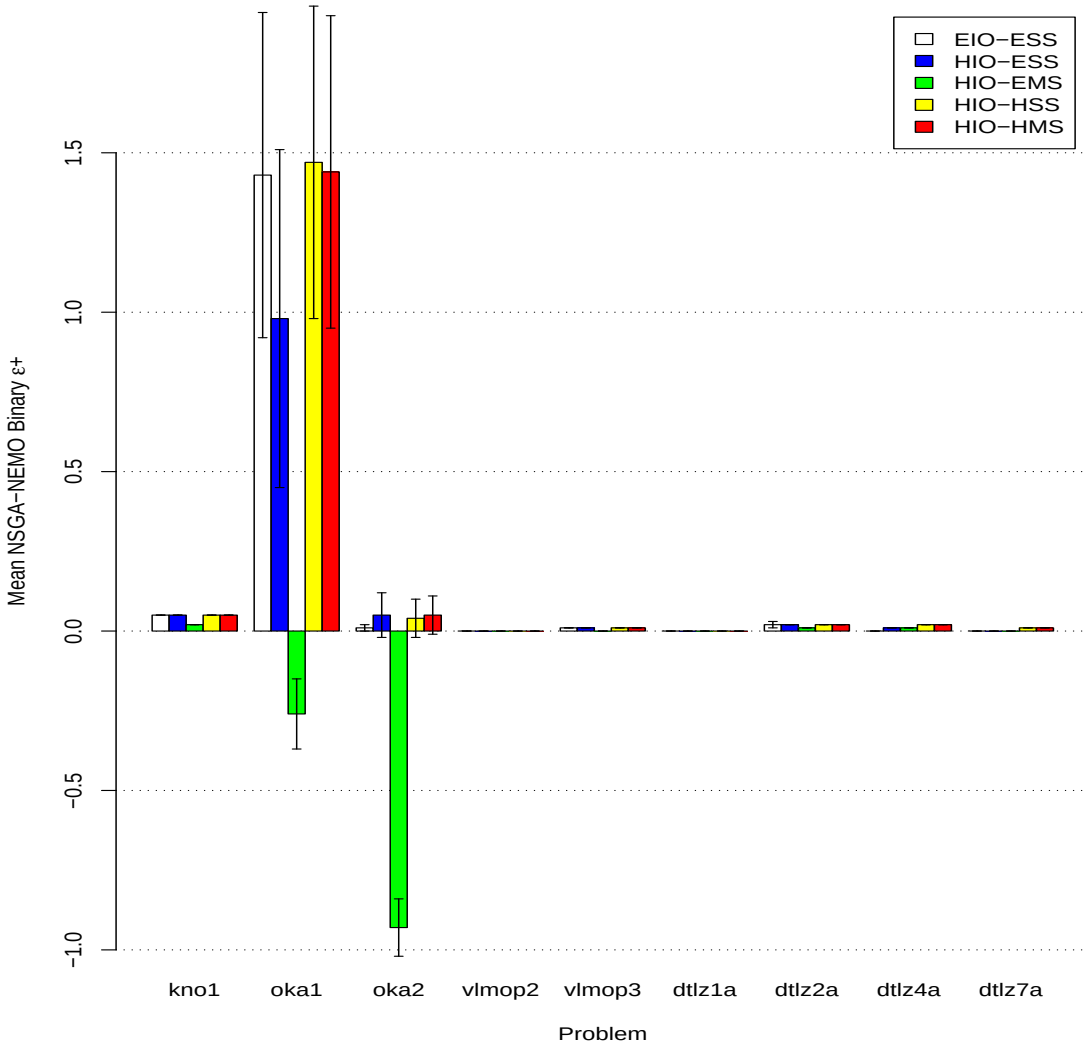


Figure 4.5: Comparison of Training Method Impact on NSGA-NEMO Binary $\epsilon+$ Indicator with Standard Error Bars.

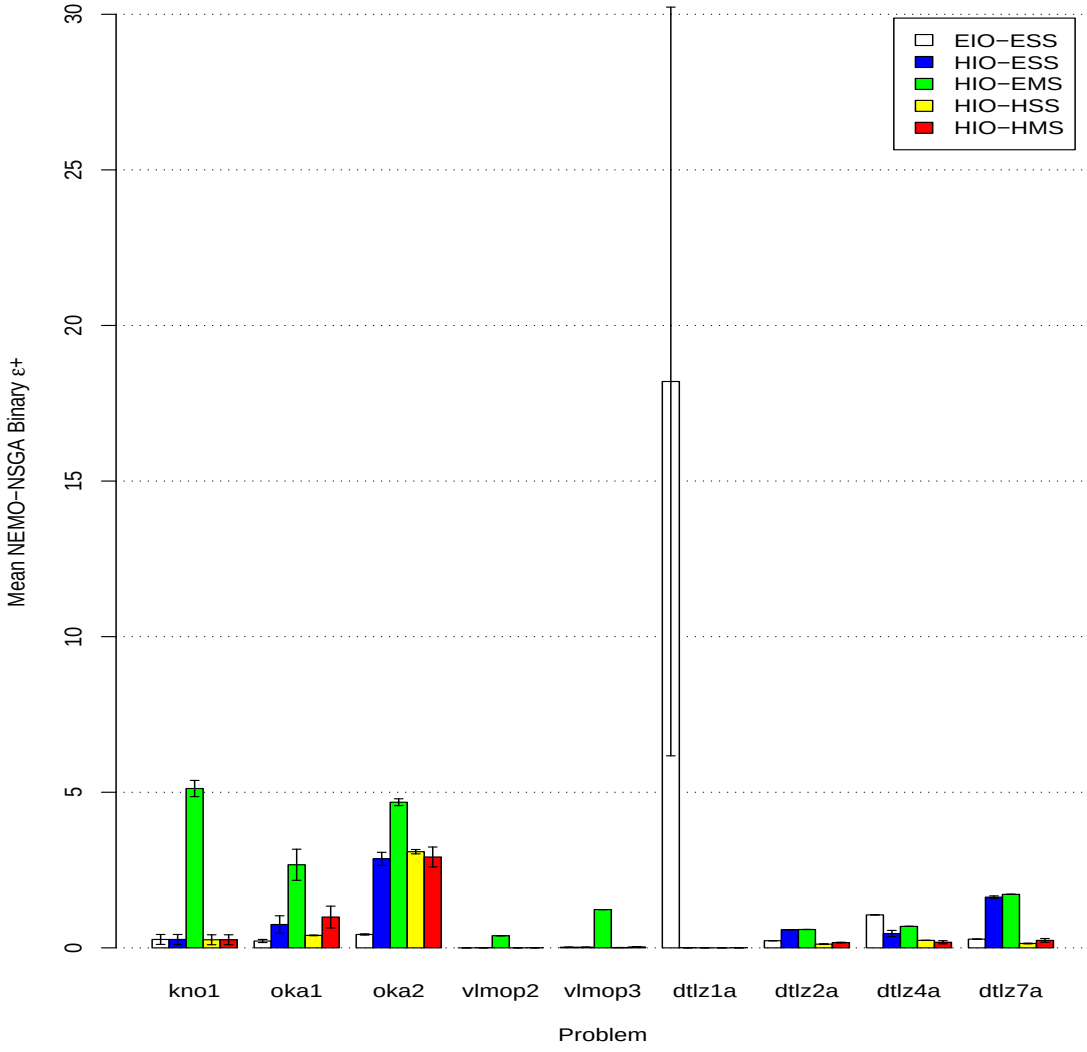


Figure 4.6: Comparison of Training Method Impact on NEMO-NSGA Binary $\epsilon+$ Indicator with Standard Error Bars.

In terms of yield ratio (Figure 4.2), this approach generally does no better than the other approaches, and in many cases (OKA1, OKA2, DTLZ1a, DTLZ2a, DTLZ4a, and DTLZ7a) it performs significantly worse. With spacing (Figure 4.3), this approach performs particularly poorly on OKA1, OKA2, DTLZ1a, DTLZ2a, and DTLZ7a. Finally, in terms of hypervolume (Figure 4.4), evolving I/O and sigma values is unable to outperform other approaches.

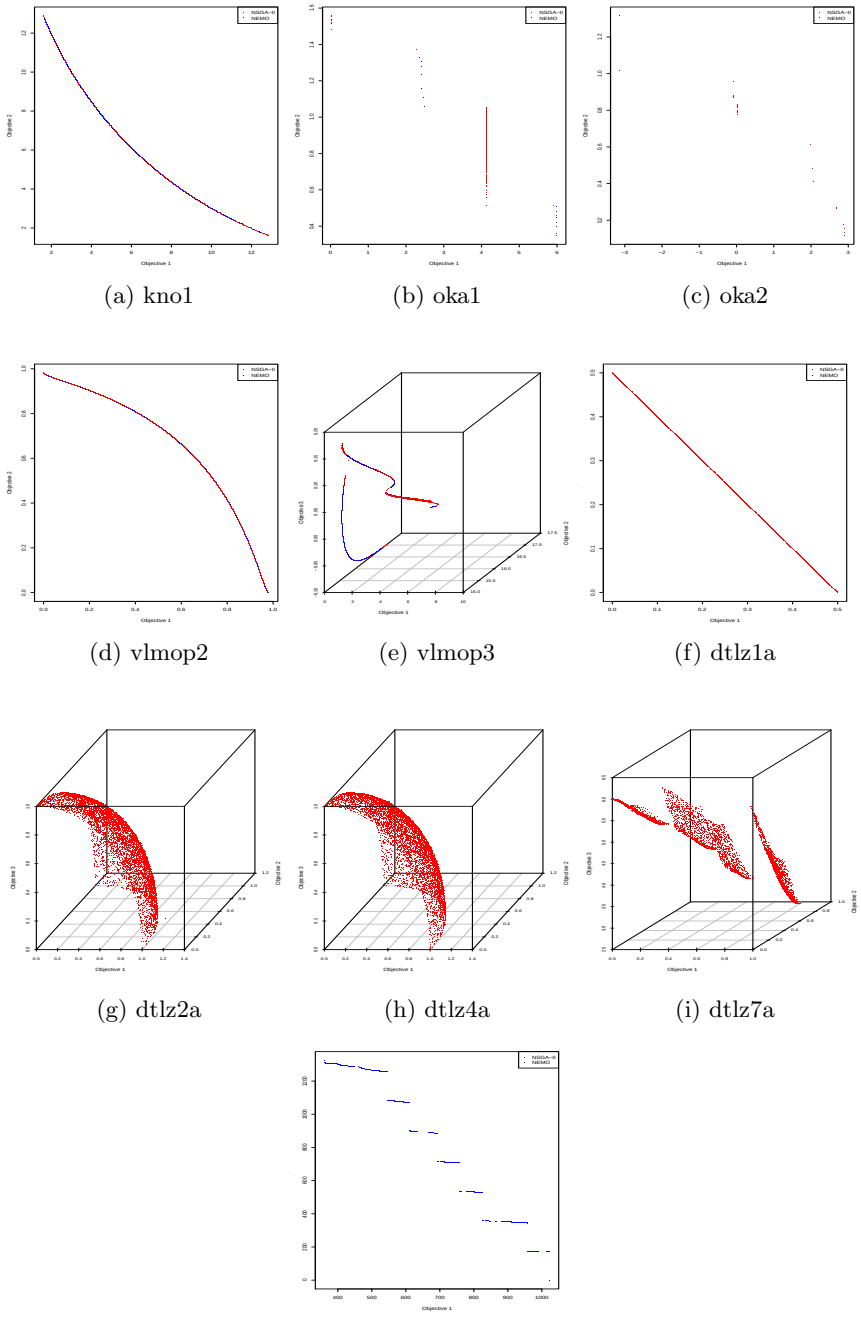
4.4.2 Heuristic I/O and Evolve Single Sigma

The results of this approach on the first run are depicted in Figure 4.9. Figure 4.10 shows the efficient sets produced for this training approach. The results from all five runs are provided in Appendix A as Table A.2.

As with all of the evolutionary training approaches, the training time (Figure 4.1) is very high when compared to the non-evolutionary approaches, especially for DTLZ2a and DTLZ4a. In terms of yield ratio (Figure 4.2), this approach performs relatively well, on par with the yield ratio of the non-evolutionary approaches. The spacing indicator (Figure 4.3) reveals that this approach performs better than the previous one, but is outperformed by the non-evolutionary approaches. Finally, in terms of hypervolume (Figure 4.4), this approach has performance that is on par with the non-evolutionary approaches.

4.4.3 Heuristic I/O and Evolve Multiple Sigmas

The results of this approach on the first run are depicted in Figure 4.11. Figure 4.12 shows the efficient sets produced for this training approach. The results from all five runs are provided in Appendix A as Table A.3.



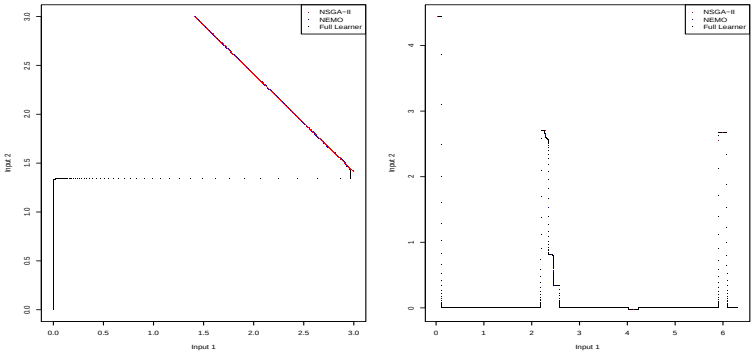
(a) kno1 (b) oka1 (c) oka2

(d) vlmop2 (e) vlmop3 (f) dtlz1a

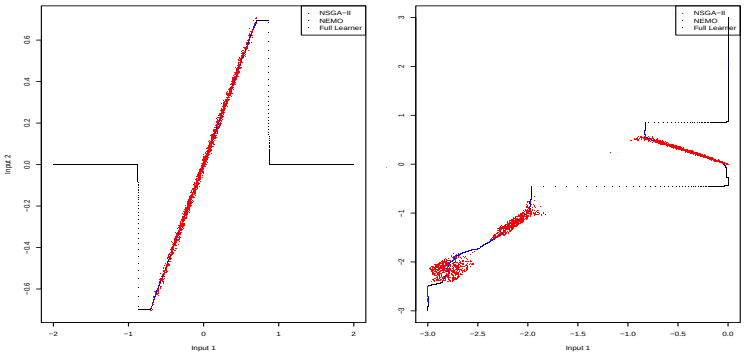
(g) dtlz2a (h) dtlz4a (i) dtlz7a

(j) sdflp

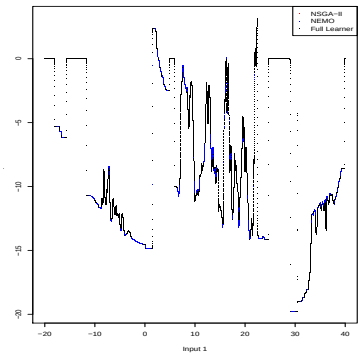
Figure 4.7: Results of Evolving I/O and Single Sigma on Test Suite



(a) kno1 (b) oka1

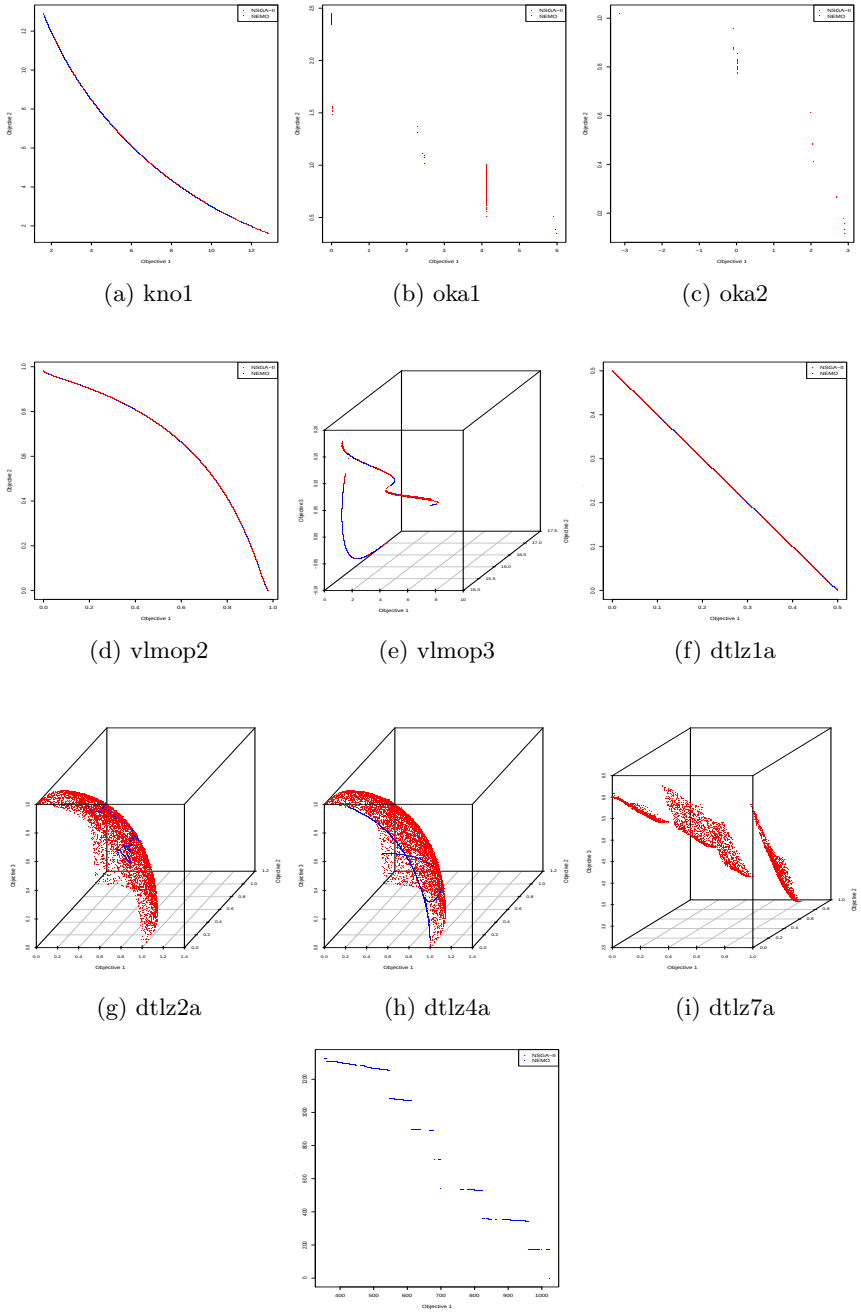


(c) vlmop2 (d) vlmop3



(e) sdflp

Figure 4.8: Efficient Set Created by Evolving I/O and Single Sigma on Test Suite



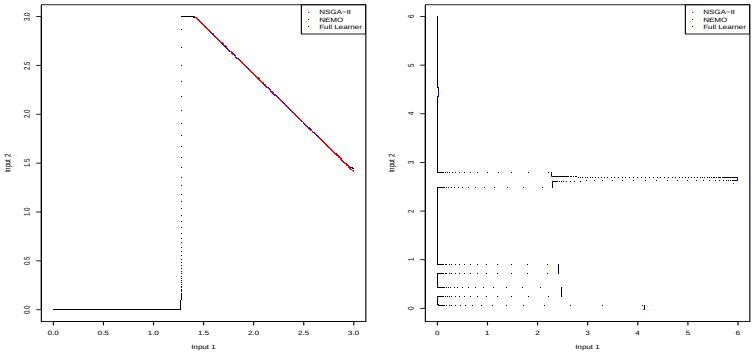
(a) kno1 (b) oka1 (c) oka2

(d) vlmop2 (e) vlmop3 (f) dtlz1a

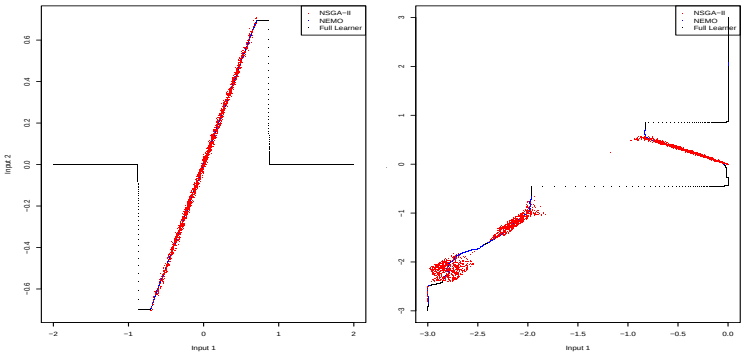
(g) dtlz2a (h) dtlz4a (i) dtlz7a

(j) sdflp

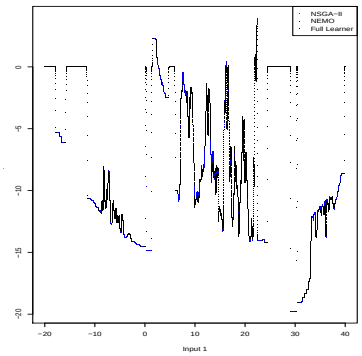
Figure 4.9: Results of Heuristic I/O and Evolving Single Sigma on Test Suite



(a) kno1 (b) oka1



(c) vlmop2 (d) vlmop3



(e) sdflp

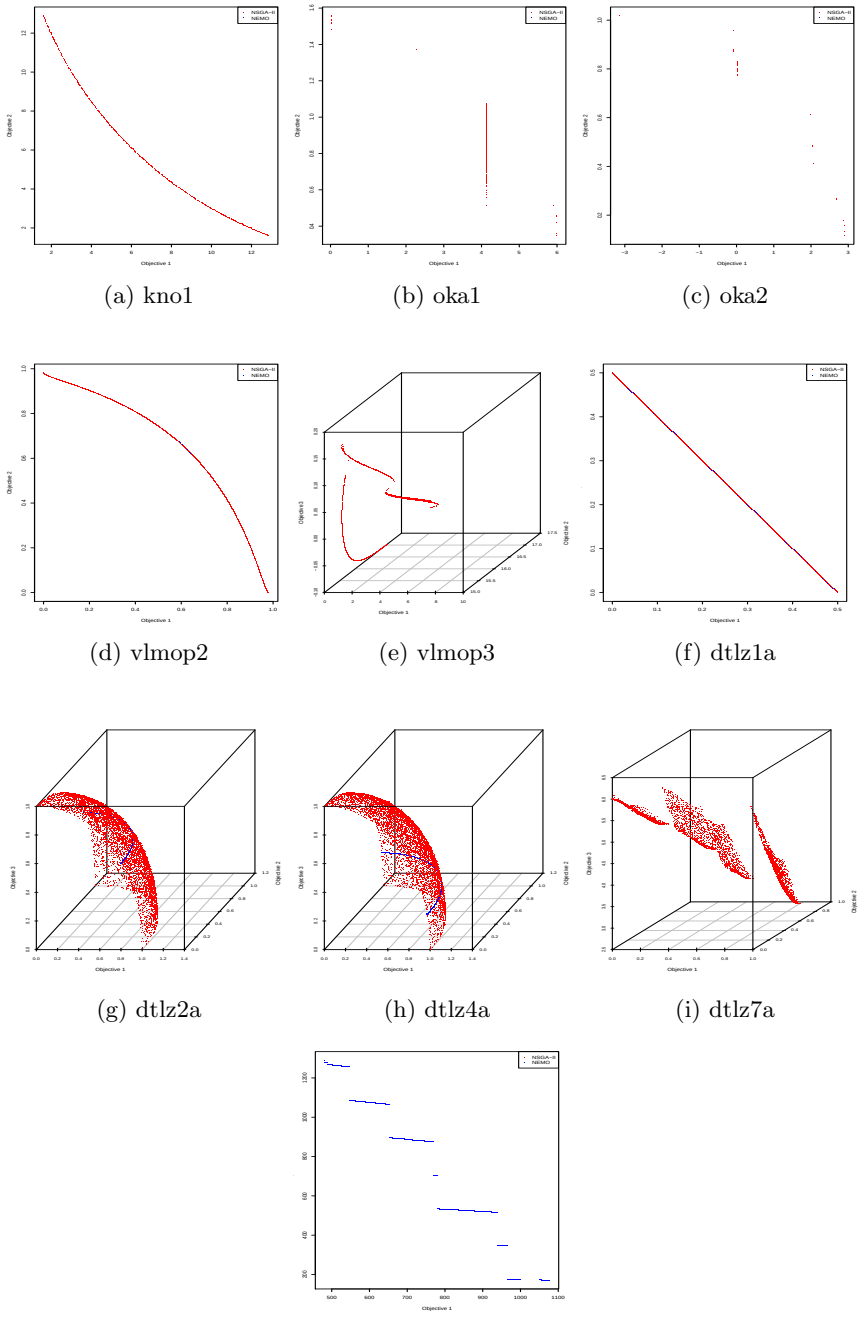
Figure 4.10: Efficient Set Created by Heuristic I/O and Evolving Single Sigma on Test Suite

Just like the other evolutionary training approaches, the training time (Figure 4.1) is very high when compared to the non-evolutionary approaches, especially for DTLZ2a and DTLZ4a. In terms of yield ratio (Figure 4.2), this approach performs relatively well, on par with the yield ratio of the non-evolutionary approaches, with the exception of KNO1 (where it performs very poorly) and OKA2 (where it performs remarkably well, achieving a yield ratio of over 300). With the spacing indicator (Figure 4.3), this system outperforms all other training approaches. Finally, in terms of hypervolume (Figure 4.4), this approach performs well except on the KNO1 test problem.

4.4.4 Heuristic I/O and Heuristic Single Sigma

The results of this approach on the first run are depicted in Figure 4.13. Figure 4.14 shows the efficient sets produced for this training approach. The results from all five runs are provided in Appendix A as Table A.4.

Clearly, the training time (Figure 4.1) for this and the next non-evolutionary approach is quite low, averaging less than 2 seconds on all problems except DTLZ2a and DTLZ4a, where it needed around 6 seconds to train. The yield ratio (Figure 4.2) is greater than 1 (which means that NEMO produced more solutions than NSGA-II) on all but three of the test problems (OKA1, DTLZ4a, and DTLZ7a). For the spacing indicator (Figure 4.3), this approach performs very well. It actually performs statistically better than NSGA-II on four of the problems (KNO1, OKA2, VLMOP3, and DTLZ2a) and is only outperformed on DTLZ1a. The hypervolume indicator (Figure 4.4) reveals that this approach outperforms NSGA-II on six of the problems (KNO1, OKA2, VLMOP2, VLMOP3, DTLZ1a, and DTLZ2a) while being outperformed on only two problems (OKA1 and DTLZ4a). For the



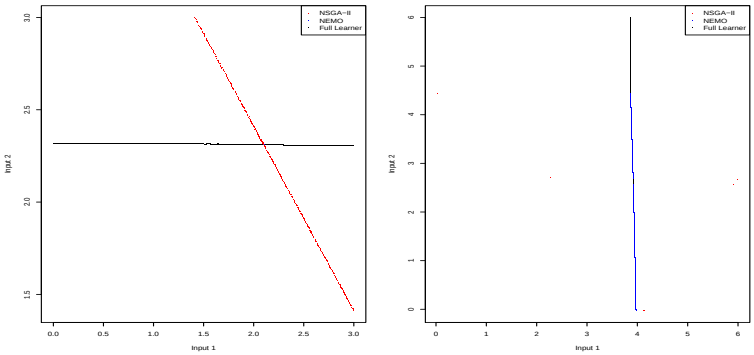
(a) kno1 (b) oka1 (c) oka2

(d) vlmop2 (e) vlmop3 (f) dtlz1a

(g) dtlz2a (h) dtlz4a (i) dtlz7a

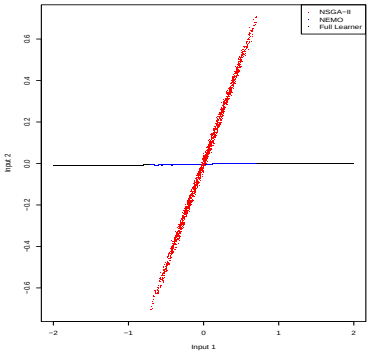
(j) sdflp

Figure 4.11: Results of Heuristic I/O and Evolving Multiple Sigmas on Test Suite

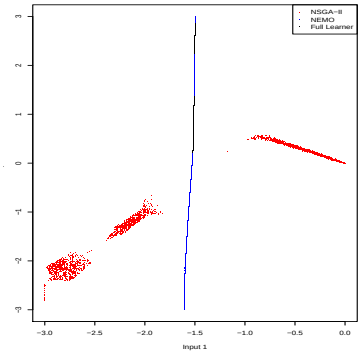


(a) kno1

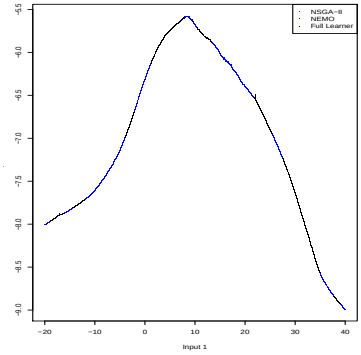
(b) oka1



(c) vlmop2



(d) vlmop3



(e) sdflp

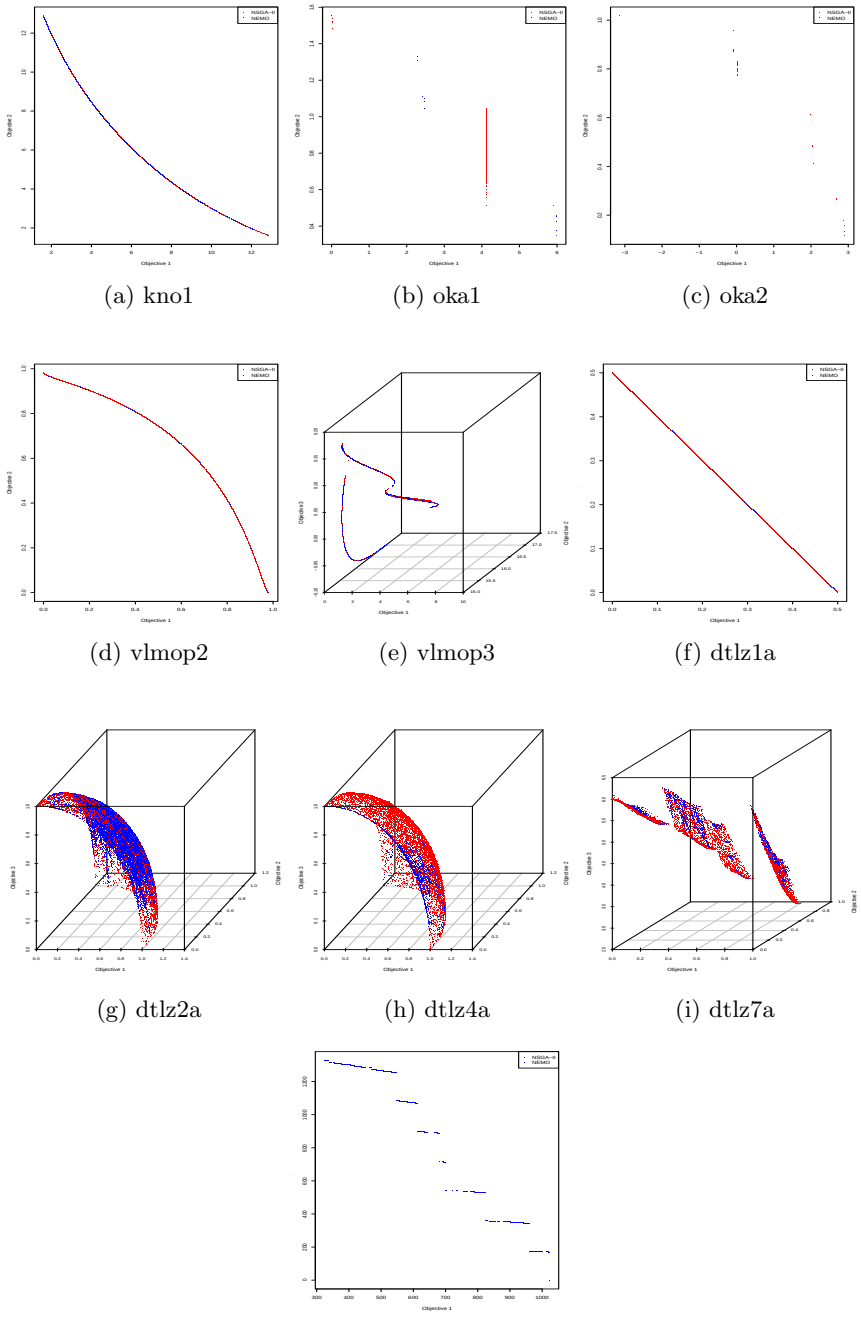
Figure 4.12: Efficient Set Created by Heuristic I/O and Evolving Multiple Sigmas on Test Suite

$\epsilon+$ indicator, neither NSGA-II nor NEMO clearly outperforms the other on average, but NSGA-II has a statistically significant lower value for this indicator on OKA2, DTLZ2a, DTLZ4a, and DTLZ7a.

4.4.5 Heuristic I/O and Heuristic Multiple Sigmas

The results of this approach on the first run are depicted in Figure 4.15. Figure 4.16 shows the efficient sets produced for this training approach. The results from all five runs are provided in Appendix A as Table A.5.

Once again, the training time (Figure 4.1) for this non-evolutionary approach is quite low, averaging less than 2 seconds on all problems except DTLZ2a and DTLZ4a, where it needed around 6 seconds to train. The yield ratio (Figure 4.2) is greater than 1 (which means that NEMO produced more solutions than NSGA-II) on all test problems, reaching over 4 (which means 4 times the number of NSGA-II solutions) on 6 of the 10 problems. As with the previous non-evolutionary approach, the spacing indicator (Figure 4.3) reveals that NEMO trained with this approach is statistically better than NSGA-II on KNO1, OKA2, VLMOP3, and DTLZ2a, while being outperformed on DTLZ1a. For the hypervolume indicator, NEMO with this training approach is statistically superior to NSGA-II on KNO1, VLMOP2, VLMOP3, DTLZ1a, and DTLZ2a, while being outperformed on none of the problems. As before, for the $\epsilon+$ indicator, neither NSGA-II nor NEMO clearly outperforms the other on average, but NSGA-II has a statistically significant lower value for this indicator on OKA2, DTLZ2a, DTLZ4a, and DTLZ7a.



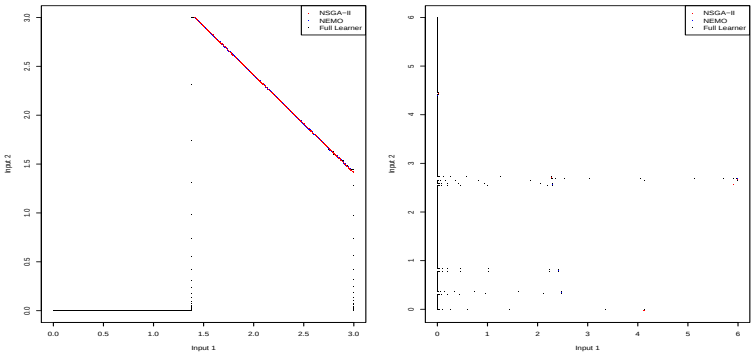
(a) kno1 (b) oka1 (c) oka2

(d) vlmop2 (e) vlmop3 (f) dtlz1a

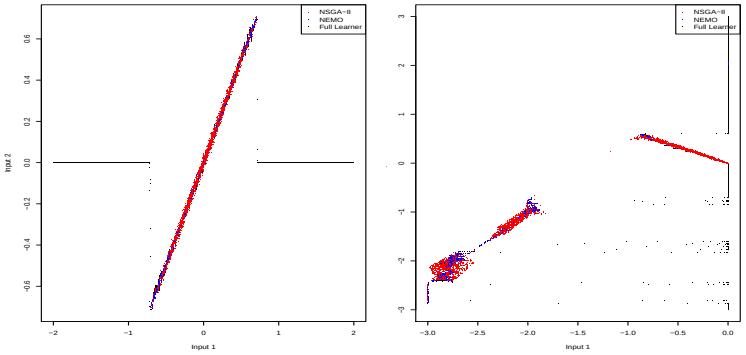
(g) dtlz2a (h) dtlz4a (i) dtlz7a

(j) sdflp

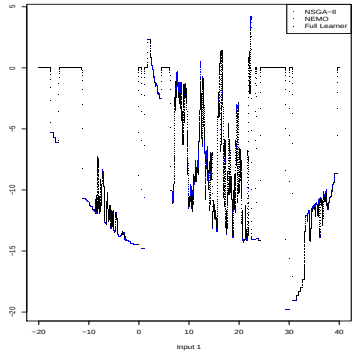
Figure 4.13: Results of Heuristic I/O and Heuristic Single Sigma on Test Suite



(a) kno1 (b) oka1

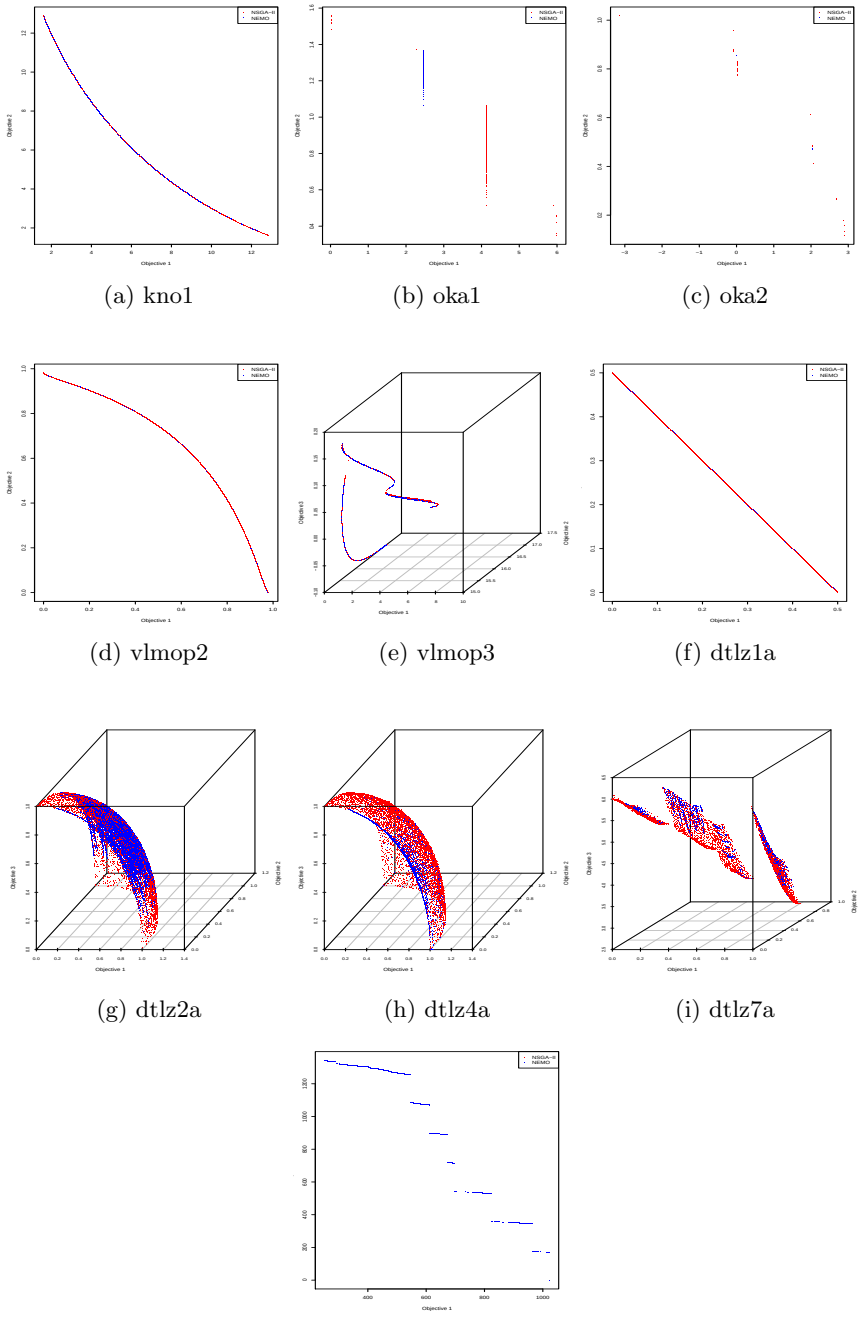


(c) vlmop2 (d) vlmop3



(e) sdflp

Figure 4.14: Efficient Set Created by Heuristic I/O and Heuristic Single Sigma on Test Suite



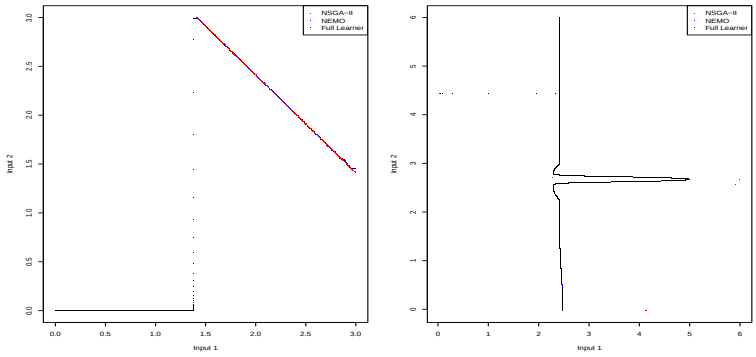
(a) kno1 (b) oka1 (c) oka2

(d) vlmop2 (e) vlmop3 (f) dtlz1a

(g) dtlz2a (h) dtlz4a (i) dtlz7a

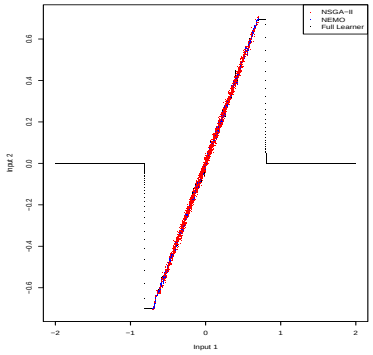
(j) sdflp

Figure 4.15: Results of Heuristic I/O and Heuristic Multiple Sigmas on Test Suite

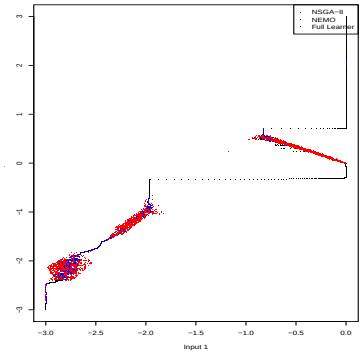


(a) kno1

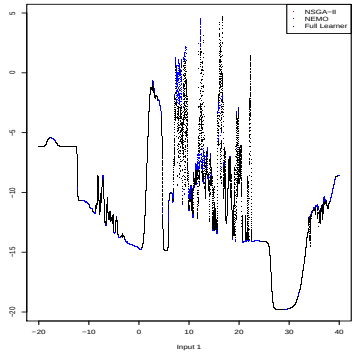
(b) oka1



(c) vlmop2



(d) vlmop3



(e) sdflp

Figure 4.16: Efficient Set Created by Heuristic I/O and Heuristic Multiple Sigmas on Test Suite

4.5 Conclusions

These experiments reveal several aspects of the NEMO approach. First, the non-evolutionary training approaches are superior to the evolutionary approaches in terms of training time, which is clearly the most important indicator for this comparison. Second, the heuristic I/O and heuristic multiple sigmas (HIO-HMS) training approach is superior to the other non-evolutionary approach in the remaining indicators. Third, NEMO trained with HIO-HMS more often outperforms NSGA-II on all indicators except for $\epsilon+$. Finally, the $\epsilon+$ indicator revealed that neither NEMO nor NSGA-II clearly outperformed the other.

Using the HIO-HMS training algorithm to produce a NEMO is the most effective approach, taking only a few seconds to train. Such a NEMO can be quite competitive with NSGA-II in all indicators. Most importantly, it can even produce many times more solutions than NSGA-II for most of the test problems considered here.

Chapter 5

Neural Enhancement Using Other Foundational Optimization Algorithms

In the previous chapter, the NEMO approach was shown to be practical and effective in expanding the set of optimal solutions found by NSGA-II. It seems reasonable to expect that similar results would arise by using a different optimization algorithm to produce the training set for NEMO. However, it is important to determine whether NEMO can perform as well when attached to other optimization approaches. There may be aspects of certain optimization techniques that enable NEMO to perform even more successful enhancement of the Pareto optimal set. Or it may be that NEMO requires a certain level of success from the underlying optimization algorithm in order to have any measure of success of its own.

In this chapter, two additional multiobjective optimization approaches are considered – Pareto Archived Evolution Strategies (PAES) [42] and Multiobjective Particle Swarm Optimization (MOPSO) [4, 53]. These two approaches differ both from NSGA-II and from one another in interesting ways. First, PAES is an evolutionary approach, like NSGA-II, but it is not a population-based evolutionary system, being based on the simple {1+1} evolution strategy [20]. Likewise, the MOPSO approach is population-based, like NSGA-II, but it is not truly an evolutionary system since it does not explicitly use evolutionary operators such as recombination or mutation. The results achieved by using each of these algorithms to train NEMO are compared to one another and to the previous results from NSGA-II.

5.1 Test Suite

The set of multiobjective optimization problems from the previous chapter was also used for this experiment. These are the nine multiobjective benchmark problems taken from [7] – KNO1, OKA1, OKA2 [67], VLMOP2, VLMOP3 [68], DTLZ1a, DTLZ2a, DTLZ4a, and DTLZ7a – and the semi-desirable facility location problem (SDFLP) [65].

5.2 Performance Assessment

As with the test suite, most of the performance indicators from the previous chapter were used in this experiment as well. These indicators were spacing [69], hypervolume [70], and binary $\epsilon+$ [70]. These indicators were described in detail in the previous chapter and, thus, will not be discussed again here. However, the yield ratio that was used in the previous chapter was not used for this work. Instead, only the sizes of the resultant Pareto optimal frontiers were used.

For each indicator, the two Pareto optimal frontiers under comparison were merged to determine the number of solutions from each that would remain nondominated. This approach reduces the size of the Pareto optimal frontiers under comparison, but a side effect is that sometimes the resultant Pareto frontier for a particular approach is empty. In such a case, it is possible for the value of the indicators to be undefined (such as the yield ratio). In the following sections, those situations will be acknowledged when necessary.

5.3 Experimental Setup

Three evolutionary multiobjective optimization approaches – NSGA-II, MOPSO, and PAES – were used to produce the training sets for NEMO. As in the previous chapter,

each EMO approach was allowed 25600 function evaluations after which the current Pareto optimal set was saved to be used for training. Then, the EMO was allowed to continue for another 25600 function evaluations (51200 total). The subsections below describe the implementations of the EMO approaches used.

Three different NEMOs were then created using those training sets – NEMO$_{NSGA-II}$, NEMO$_{MOPSO}$, and NEMO$_{PAES}$. Each NEMO was allowed 25600 function evaluations to expand the training set for comparison with the EMO-only approach. As before, each NEMO consisted of a general regression neural network [32] trained using heuristic approaches for finding the input/output assignment and multiple σ values.

In order to determine statistical significance, each EMO was used to produce 30 different Pareto optimal fronts (using a different pseudorandom seed each time) on each of the 10 problems in the test suite. Therefore, 30 different runs were generated, each yielding three different EMOs and three different NEMOs.

5.3.1 NSGA-II

The implementation used for NSGA-II was the same as was used in the previous chapters. For each problem, NSGA-II was initialized using the software's default parameters as specified in Table 5.1.

5.3.2 PAES

The implementation used for PAES was based on C source code from Joshua Knowles' professional website [74]. This code was re-implemented in C++ by the author for these experiments using object-oriented programming techniques to make it more modular. The

Problem	Population Size	Crossover Probability	Mutation Probability	Crossover Distribution Index	Mutation Distribution Index
kno1	256	0.9	0.5	10	50
oka1	256	0.9	0.5	10	50
oka2	256	0.9	0.3333	10	50
vlmop2	256	0.9	0.5	10	50
vlmop3	256	0.9	0.5	10	50
dtlz1a	256	0.9	0.1667	10	50
dtlz2a	256	0.9	0.125	10	50
dtlz4a	256	0.9	0.125	10	50
dtlz7a	256	0.9	0.125	10	50
sdflp	256	0.9	0.5	10	50

Table 5.1: Parameters for NSGA-II

new implementation was tested against Knowles' existing code to ensure a correct implementation. It was successfully compiled under Borland version 5.5 and was executed on a Pentium 4 system running Windows XP. For each problem, PAES was initialized with parameters as illustrated in Table 5.2.

5.3.3 MOPSO

The implementation used for MOPSO was based on C source code from the EMOO repository [75]. As with the PAES source code, this code was re-implemented in C++ by the author for these experiments using object-oriented programming techniques. The new implementation was tested against the original code to ensure a correct implementation. It was successfully compiled under Borland version 5.5 and was executed on a Pentium 4 system running Windows XP. For all problems, MOPSO was initialized with a population

Problem	Population Size	Archive Size	Grid Divisions	Mutation Usage Rate	Mutation Range
kno1	100	100	5	0.1	0.2
oka1	100	100	5	0.1	0.001
oka2	100	100	5	0.1	0.001
vlmop2	100	100	5	0.1	0.1
vlmop3	100	100	5	0.1	0.1
dtlz1a	100	100	5	0.1	0.001
dtlz2a	100	100	5	0.1	0.1
dtlz4a	100	100	5	0.1	0.01
dtlz7a	100	100	5	0.1	0.01
sdflp	100	100	5	0.1	0.1

Table 5.2: Parameters for PAES

size of 100, an archive size of 100, cognitive and social rates of 1.0, an inertia weight of 0.4, and 30 grid divisions. Mutation was also used on the population as described in [53] with a mutation rate of 0.1.

5.4 Results

The results from this experiment consist of six Pareto optimal sets – one from each of the EMOs and one from each of the NEMOs. In order to analyze these sets and calculate the performance indicators, it was necessary to consider them pairwise as one EMO versus one NEMO. The Pareto optimal sets from each pair were "merged" to determine the solutions that "survived", and only those solutions were kept in each optimal set for comparison. Additionally, the hypervolume indicator requires a bounding point for comparison, and such a bounding point must be calculated using a pair of optimal sets.

Once the indicators were calculated for each of the 30 runs on a given pairing, the mean and standard deviation were calculated for the sizes of the Pareto optimal sets, the spacing indicator, and the hypervolume indicator. The median and interquartile range (IQR) were calculated for the binary $\epsilon+$ indicator instead of the mean and standard deviation. This is because a median of less than 0 for the $\epsilon+$ indicator means that the indicator was less than 0 for at least 50% of the runs. Finally, tests for statistical significance were performed. Because there is no evidence to suggest that the distribution of the indicators is approximately normal, a Mann-Whitney rank-sum test was used to test for statistical significance. In the following nine sections, each pairwise comparison is analyzed.

For each pairwise comparison described below, tables are provided that display the average indicator values for the Pareto optimal front size, spacing indicator, and hypervolume indicator for each problem in the test suite, along with the p-value from the rank-sum test. To make the analysis more accessible, if $p < 0.05$, the statistically dominant average in each table is boldfaced and underlined. Additionally, tables are given that show the median and interquartile ranges for the binary $\epsilon+$ indicator, along with the p-value from the rank-sum test (with similar boldfacing and underlining denoting statistical significance). Likewise, figures are provided that display the average and standard errors of the optimal front sizes, spacing, hypervolume indicators, and the median and standard errors of the binary $\epsilon+$ indicators, for each problem except SDFLP. It was removed from the figures because the indicator values produced by NEMO were, in many cases, so large that the graph was rendered essentially useless for the remaining problems in the test suite.

5.4.1 NSGA-II versus NEMO$_{NSGA-II}$

This comparison is identical to the final experiment in the previous chapter. However, in this case, 30 runs were used to ensure statistical significance. Tables 5.3-5.5 show the average Pareto optimal front size, spacing indicator, and hypervolume indicators, while Table 5.6 shows the median and interquartile ranges for the binary $\epsilon+$ indicator. Additionally, Figures 5.1-5.4 display the average and standard errors of the optimal front sizes, spacing, hypervolume indicators, and the median and standard errors of the binary $\epsilon+$ indicators.

As was seen previously, NEMO$_{NSGA-II}$ greatly outperforms NSGA-II in terms of Pareto optimal set size. It also outperforms NSGA-II on the spacing and hypervolume indicators for most problems in the test suite. However, on the binary $\epsilon+$ indicator, NSGA-II outperforms NEMO$_{NSGA-II}$ on most problems, but it specifically dominates on OKA2, where it achieves a negative median value.

Problem	NSGA-II		NEMO$_{NSGA-II}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	611.57	19.04	**9856.07**	201.07	3.00E-011
oka1	**795.53**	234.77	239.1	102.12	7.38E-011
oka2	**29.17**	7.48	2.97	9.73	3.17E-010
vlmop2	2526.03	41.17	**8434.1**	149.72	3.00E-011
vlmop3	2016.47	1182.48	**10211.1**	562.54	3.02E-011
dtlz1a	2210.1	229.07	**7867.37**	1180.86	3.02E-011
dtlz2a	162.43	44.68	**10874.63**	38.69	3.00E-011
dtlz4a	3467.77	2012.24	**7327.6**	1348.46	3.50E-009
dtlz7a	4274.87	1294.04	3639.5	725.25	0.19
sdflp	1236.1	36.35	**5318.03**	440.75	3.01E-011

Table 5.3: Pareto Size Indicators for NSGA-II and NEMO$_{NSGA-II}$

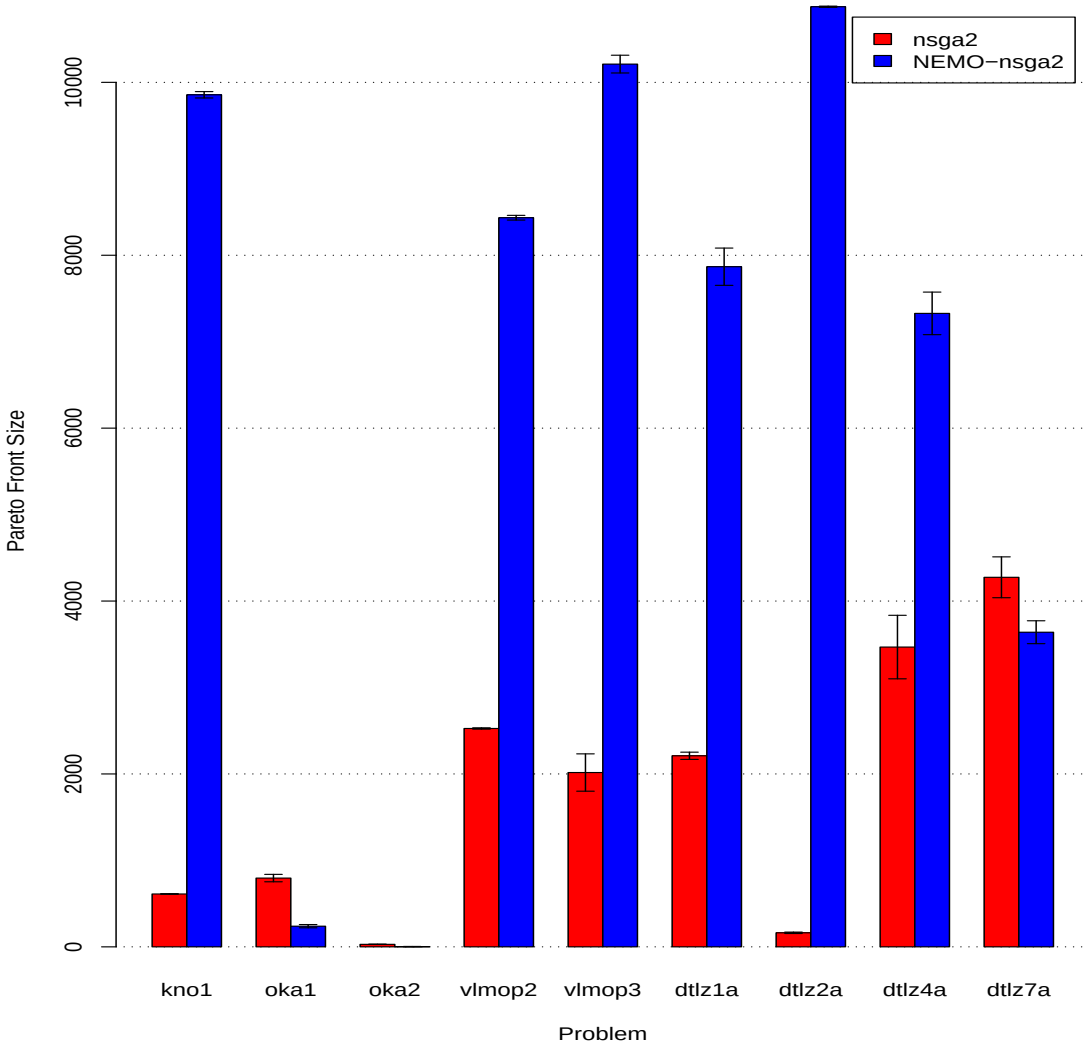


Figure 5.1: Pareto Size Indicators for NSGA-II and NEMO$_{NSGA-II}$ with Standard Error Bars.

Problem	NSGA-II		NEMO$_{NSGA-II}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	0.02	0	**0**	0	4.15E-014
oka1	**0.06**	0.04	0.12	0.15	0.02
oka2	0.49	0.33	**0.06**	0.26	2.95E-010
vlmop2	0	0	0	0	NaN
vlmop3	0.01	0	**0**	0	5.36E-009
dtlz1a	0	0	0	0	NaN
dtlz2a	0.06	0.02	**0.01**	0	4.50E-012
dtlz4a	0.01	0.01	0.01	0	0.16
dtlz7a	0.03	0.03	0.03	0.02	0.76
sdflp	2.36	2.23	2.27	0.4	0.53

Table 5.4: Spacing Indicators for NSGA-II and NEMO$_{NSGA-II}$

Problem	NSGA-II		NEMO$_{NSGA-II}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	18242.5	782.66	**350345.27**	6445.41	3.02E-011
oka1	**943.09**	353.19	276.74	127.2	5.97E-009
oka2	**11.6**	4.45	1.59	4.41	2.14E-009
vlmop2	237.79	3.85	**655.41**	9.28	3.02E-011
vlmop3	2155.04	1314.66	**6592.21**	728.88	3.02E-011
dtlz1a	99.34	13.51	**331.46**	69.83	3.02E-011
dtlz2a	10.41	3.6	**769.54**	201.72	3.02E-011
dtlz4a	285.93	213.42	**478.05**	197.55	0
dtlz7a	3932.94	2409.43	3118.31	2098.3	0.13
sdflp	135300000	4549725.27	**6.81E+008**	70958462.84	2.94E-011

Table 5.5: Hypervolume Indicators for NSGA-II and NEMO$_{NSGA-II}$

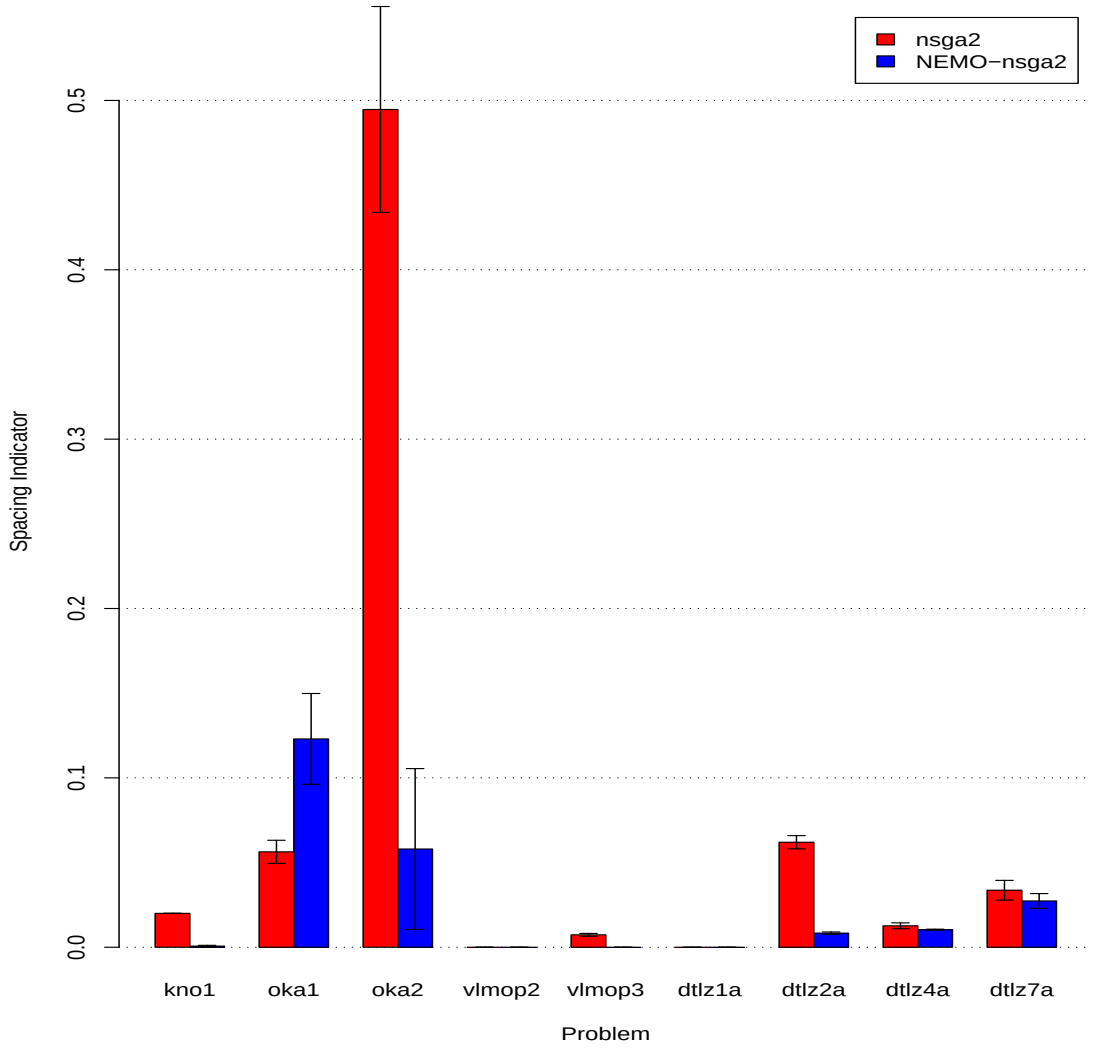


Figure 5.2: Spacing Indicators for NSGA-II and NEMO$_{NSGA-II}$ with Standard Error Bars.

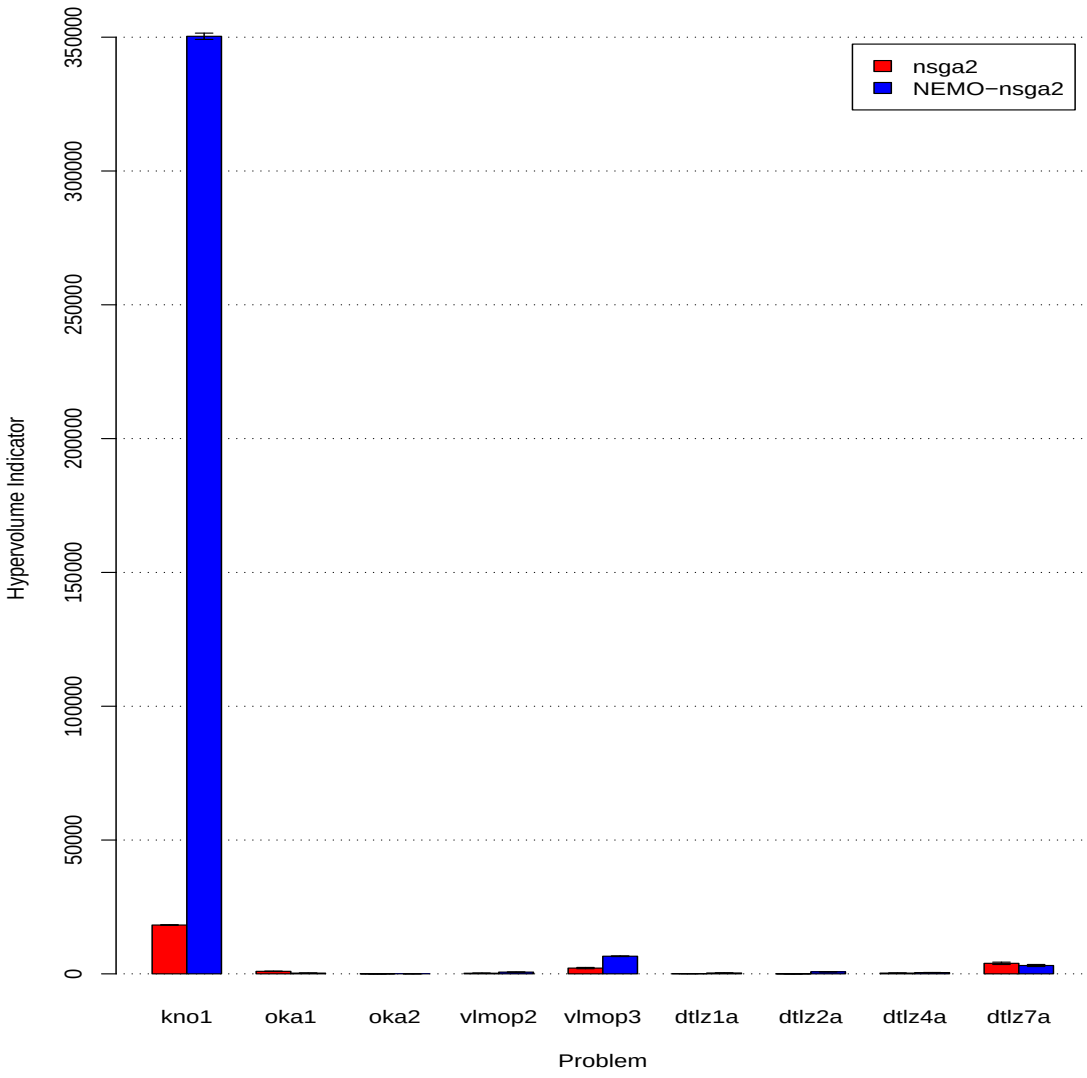


Figure 5.3: Hypervolume Indicators for NSGA-II and NEMO$_{NSGA-II}$ with Standard Error Bars.

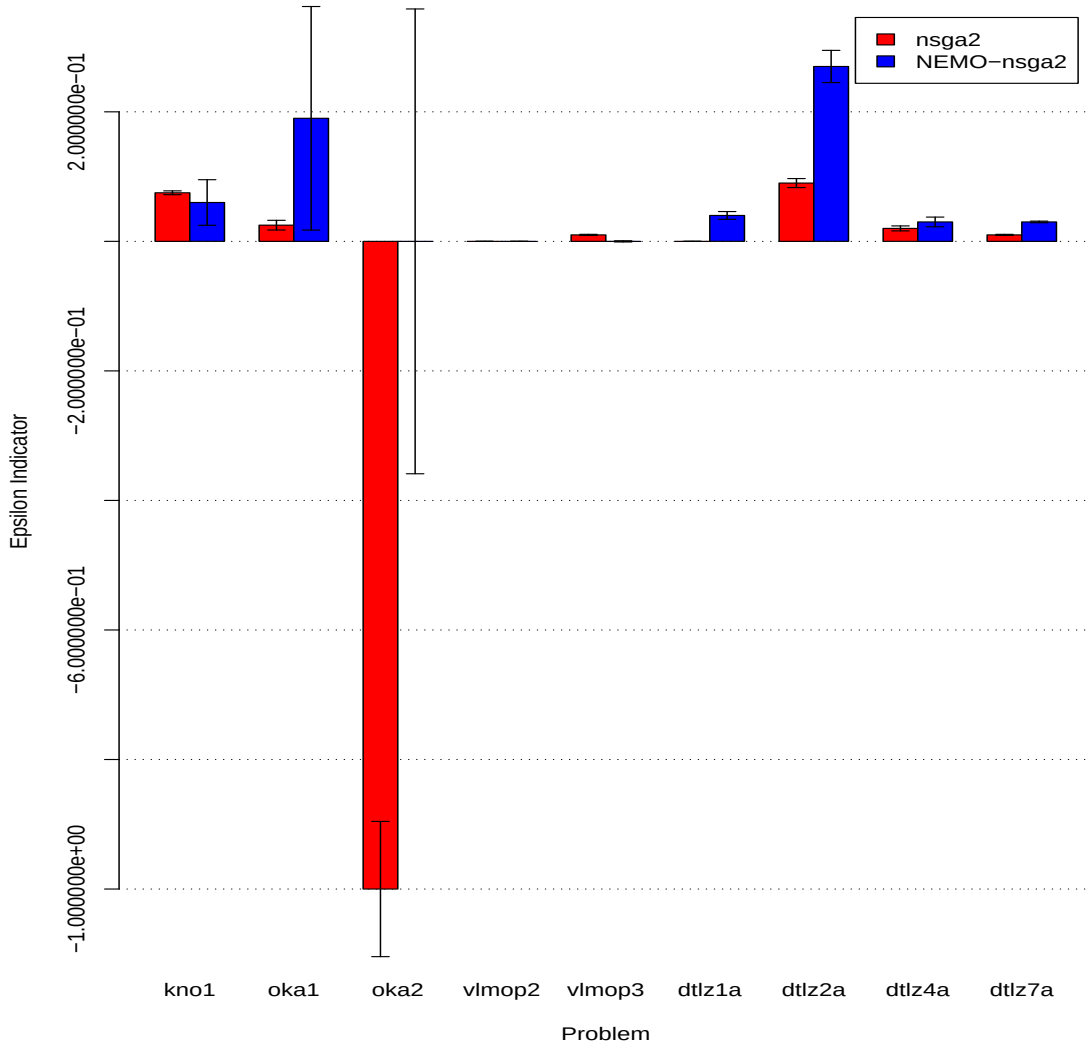


Figure 5.4: $\epsilon+$ Indicators for NSGA-II and NEMO$_{NSGA-II}$ with Standard Error Bars.

Problem	NSGA-II		NEMO$_{NSGA-II}$		p-value
	Median	IQR	Median	IQR	
kno1	0.08	0.02	**0.06**	0.04	0.01
oka1	**0.03**	0.06	0.19	1.64	4.14E-010
oka2	**-1**	1.04	0	3.14	6.63E-005
vlmop2	0	0	0	0	NaN
vlmop3	0.01	0	**0**	0.01	0
dtlz1a	**0**	0	0.04	0.02	1.09E-012
dtlz2a	**0.09**	0.03	0.27	0.25	1.73E-009
dtlz4a	**0.02**	0.01	0.03	0.03	0.01
dtlz7a	**0.01**	0.01	0.03	0.01	3.24E-012
sdflp	170	168.14	**0**	0	1.20E-012

Table 5.6: $\epsilon+$ Indicators for NSGA-II and NEMO$_{NSGA-II}$

5.4.2 NSGA-II versus NEMO$_{MOPSO}$

In this comparison, MOPSO was used as the underlying EMO algorithm to train NEMO. Those results were compared against NSGA-II. Tables 5.7-5.9 show the average Pareto optimal front size, spacing indicator, and hypervolume indicator, and Table 5.10 shows the median and interquartile ranges for the binary $\epsilon+$ indicator. Figures 5.5-5.8 display the average and standard errors of the optimal front sizes, spacing, and hypervolume indicators, and the median and standard errors of the binary $\epsilon+$ indicators.

In terms of the size of the Pareto optimal set, NEMO$_{MOPSO}$ outperforms NSGA-II in 60% of the problems in the test suite. It outperforms NSGA-II in terms of the spacing indicator on 50% of the problems, while being outperformed on only 30% of the problems. For the hypervolume indicator, NEMO$_{MOPSO}$ outperforms NSGA-II in 60% of the problems in the test suite. However as before, on the binary $\epsilon+$ indicator, NSGA-II outperforms NEMO$_{MOPSO}$ on 60% of the problems. While it never dominates NEMO$_{MOPSO}$ conclusively (i.e., attaining a negative value for $\epsilon+$), NSGA-II does very well compared to the NEMO approach in terms of this indicator.

5.4.3 NSGA-II versus NEMO$_{PAES}$

In this comparison, PAES was used as the underlying EMO algorithm to train NEMO and compared against NSGA-II. This comparison is interesting in that NSGA-II has been shown to consistently outperform PAES in a head-to-head comparison [6]. If NEMO$_{PAES}$ is capable of even competing with NSGA-II, then the power of the NEMO approach will be realized.

Problem	NSGA-II		NEMO$_{MOPSO}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	206.53	317.78	**9916.73**	3011.68	9.72E-010
oka1	**793.83**	160.03	14.4	8.05	2.94E-011
oka2	**28.47**	7.55	3.3	4.09	1.94E-011
vlmop2	101.43	29.16	**10898.03**	29.14	3.01E-011
vlmop3	881.03	823.62	**10118.83**	823.56	3.02E-011
dtlz1a	1726.8	933.24	**7252.73**	3817.22	8.88E-006
dtlz2a	3891.13	2143.89	**7105**	2143.56	3.32E-006
dtlz4a	**3605.77**	3694.08	920.63	1395.93	0.01
dtlz7a	**5333.2**	1423.7	1261.3	2849.29	1.73E-007
sdflp	1196.33	33.13	**2399.1**	382.7	3.01E-011

Table 5.7: Pareto Size Indicators for NSGA-II and NEMO$_{MOPSO}$

Problem	NSGA-II		NEMO$_{MOPSO}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	0.21	0.16	**0.01**	0.01	5.58E-010
oka1	**0.06**	0.04	0.3	0.36	0.02
oka2	**0.2**	0.26	0.65	1.07	0.01
vlmop2	0.01	0	**0**	0	6.11E-014
vlmop3	0.02	0.01	**0**	0	6.27E-010
dtlz1a	0	0	9.20E-008	5.04E-007	1
dtlz2a	0.01	0	**0**	0.01	1.29E-007
dtlz4a	0.02	0.03	0.01	0.01	0.18
dtlz7a	**0.02**	0.02	0.16	0.18	1.16E-005
sdflp	4.35	1.55	**0.82**	0.91	3.86E-007

Table 5.8: Spacing Indicators for NSGA-II and NEMO$_{MOPSO}$

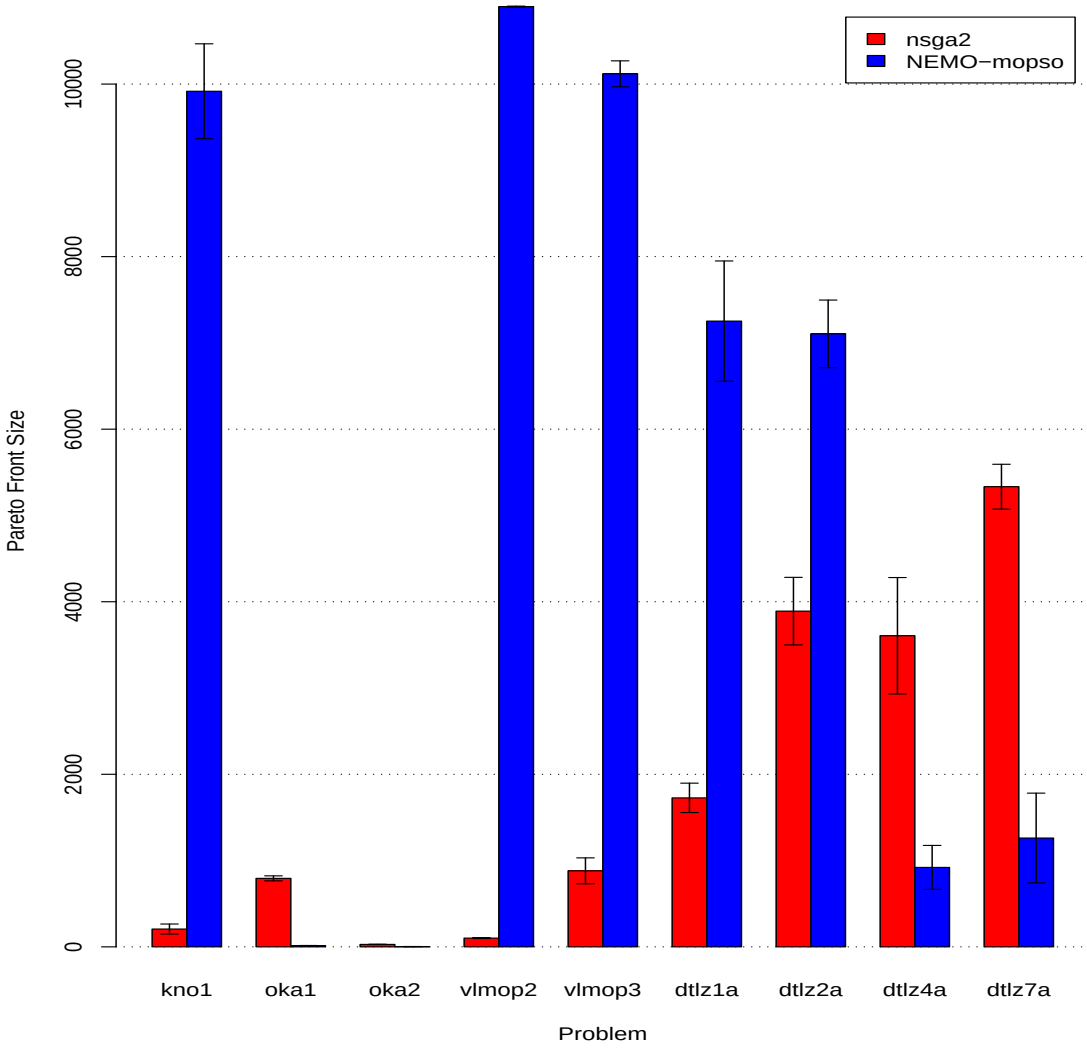


Figure 5.5: Pareto Size Indicators for NSGA-II and NEMO$_{NSGA-II}$ with Standard Error Bars.

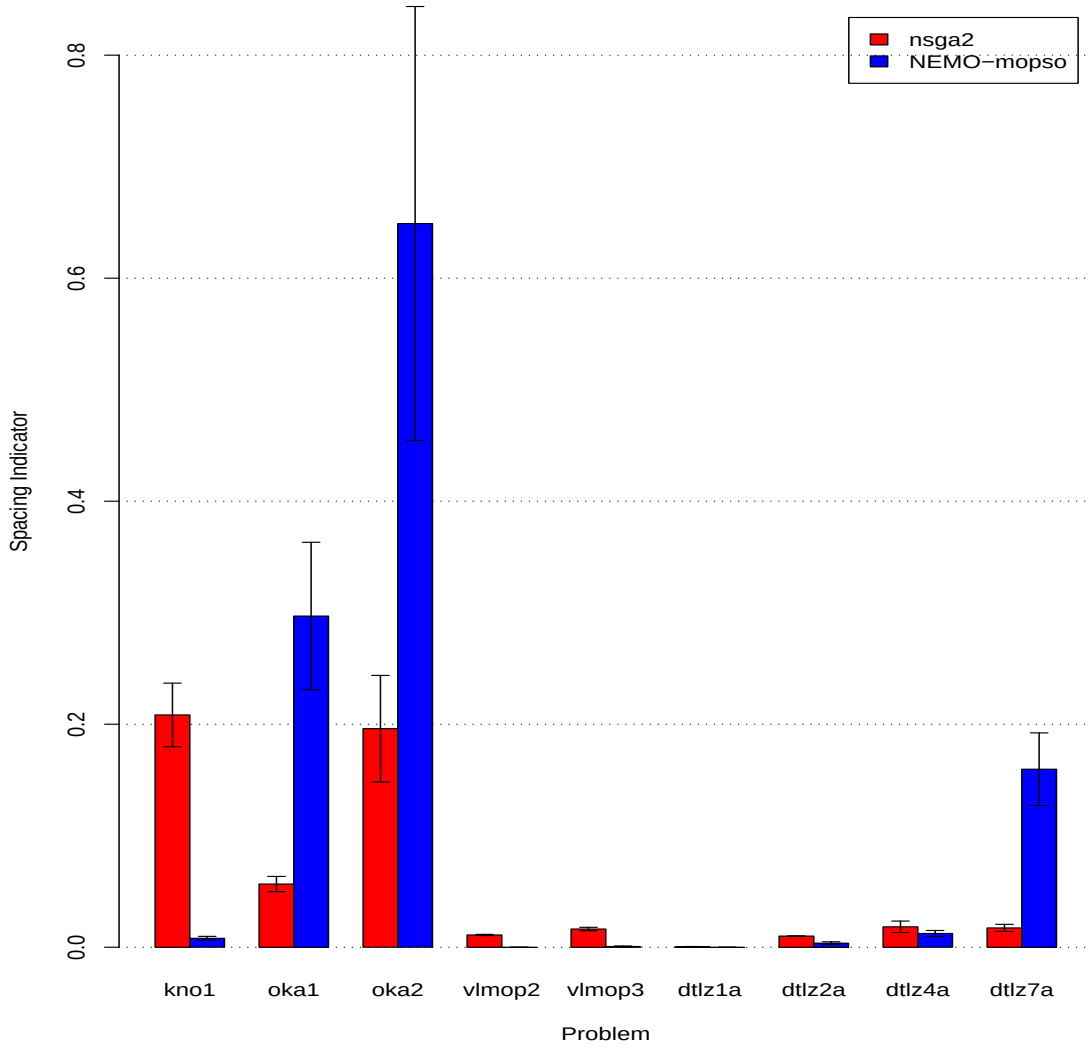


Figure 5.6: Spacing Indicators for NSGA-II and NEMO$_{NSGA-II}$ with Standard Error Bars.

Problem	NSGA-II		NEMO$_{MOPSO}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	6169.27	10544.5	**347121.6**	104564.27	1.41E-009
oka1	**947.34**	317.35	15.75	9.68	8.15E-011
oka2	**13.37**	5.56	0.82	1.52	4.13E-011
vlmop2	8.46	2.06	**872.23**	10.12	3.01E-011
vlmop3	1060.54	830.44	**5508.68**	1283.84	3.02E-011
dtlz1a	74.38	42.44	**310.36**	157.96	6.28E-006
dtlz2a	303.52	199.25	**434.55**	126.85	0
dtlz4a	**235.6**	243.77	52.96	105.58	0
dtlz7a	**4308.02**	3040.91	236.97	513.52	2.67E-009
sdflp	132700000	3524789.06	**287100000**	54732169.2	2.94E-011

Table 5.9: Hypervolume Indicators for NSGA-II and NEMO$_{MOPSO}$

Problem	NSGA-II		NEMO$_{MOPSO}$		p-value
	Median	IQR	Median	IQR	
kno1	0.51	0.59	**0.14**	0.33	0.01
oka1	**0.05**	0.04	2.26	0.04	3.72E-011
oka2	2.84	3.76	0.88	0.35	0.37
vlmop2	0.03	0.01	**0**	0	7.50E-013
vlmop3	0.02	0.02	0.01	0.02	0.08
dtlz1a	**0**	0	0.07	0.15	1.31E-010
dtlz2a	**0.02**	0.02	0.59	0.03	2.03E-011
dtlz4a	**0.02**	0.05	0.26	0.34	0
dtlz7a	**8.03E-006**	3.33E-005	2.25	1.6	5.54E-010
sdflp	**1.9**	2.46	170.11	0.19	2.72E-010

Table 5.10: $\epsilon+$ Indicators for NSGA-II and NEMO$_{MOPSO}$

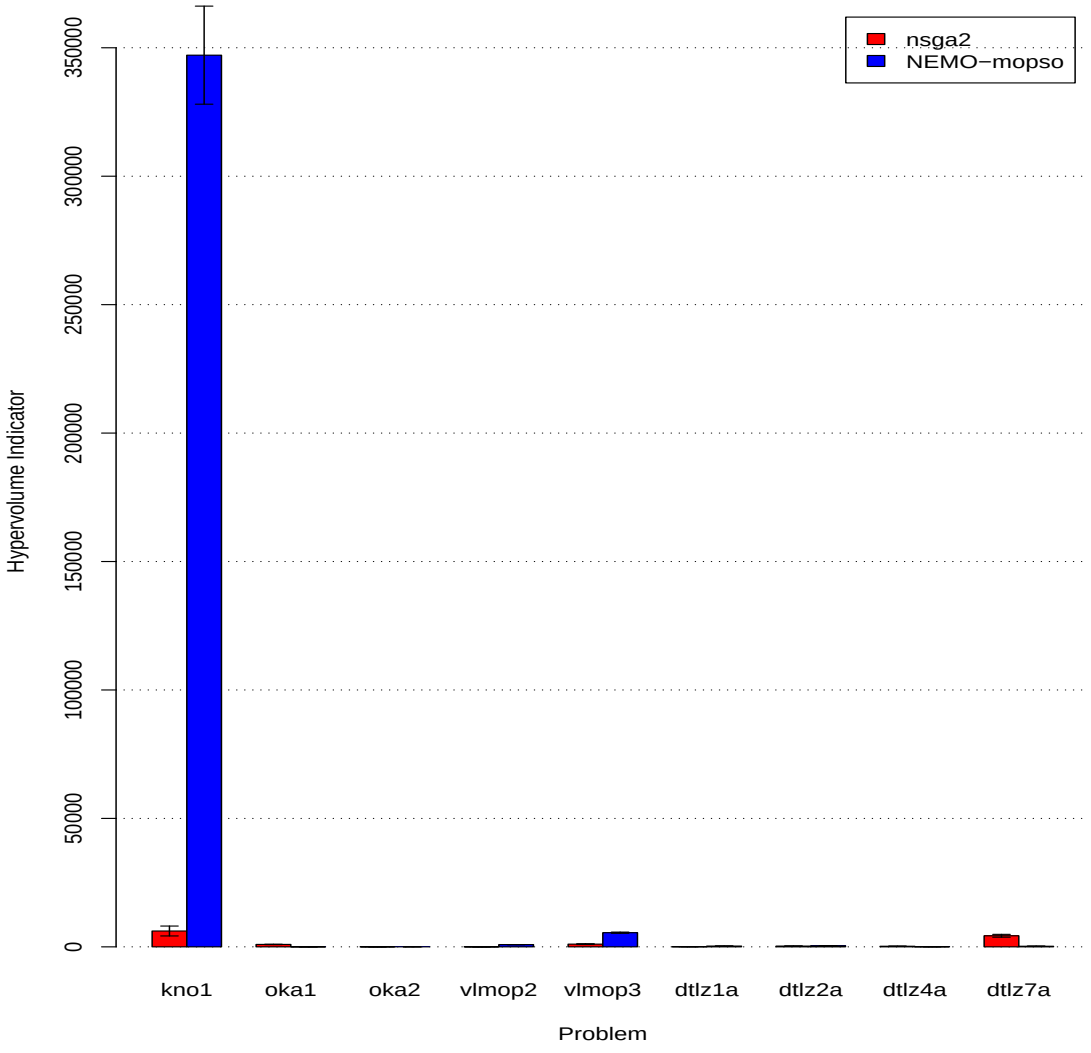


Figure 5.7: Hypervolume Indicators for NSGA-II and NEMO$_{NSGA-II}$ with Standard Error Bars.

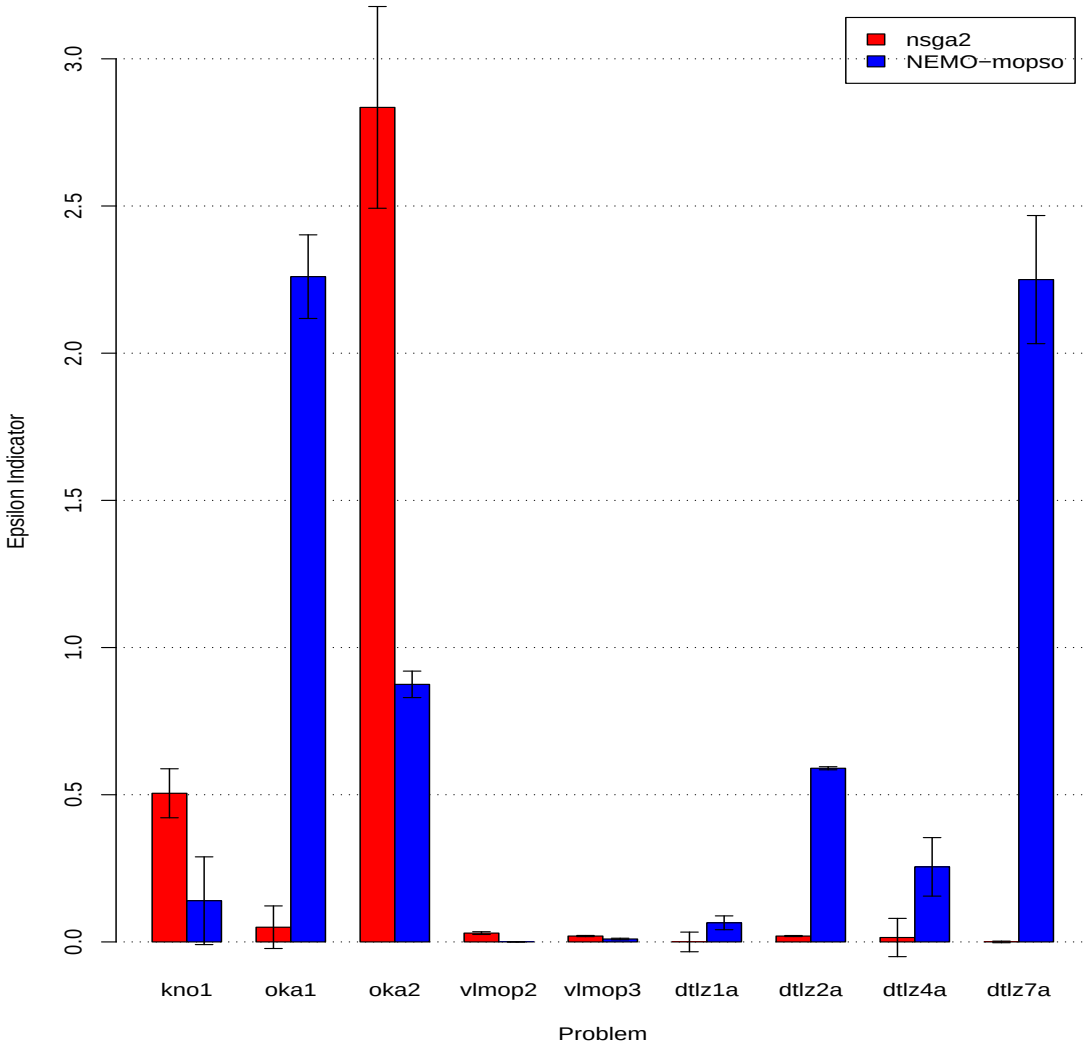


Figure 5.8: $\epsilon+$ Indicators for NSGA-II and NEMO$_{NSGA-II}$ with Standard Error Bars.

Tables 5.11-5.13 show the average Pareto optimal front size, spacing indicator, and hypervolume indicator for each problem in the test suite, while Table 5.14 shows the median and interquartile ranges for the binary $\epsilon+$ indicator. Likewise, Figures 5.9-5.12 display the average and standard errors of the optimal front sizes, spacing, and hypervolume indicators, and the median and standard errors of the binary $\epsilon+$ indicators.

In terms of the size of the Pareto optimal set, NEMO$_{PAES}$ outperforms NSGA-II in 50% of the problems in the test suite. It outperforms NSGA-II in terms of the spacing indicator on 40% of the problems, while being outperformed on only 30% of the problems. For the hypervolume indicator, NEMO$_{PAES}$ outperforms NSGA-II in 50% of the problems in the test suite. However, on the binary $\epsilon+$ indicator, NSGA-II outperforms NEMO$_{PAES}$ on 80% of the problems. While it never dominates NEMO$_{PAES}$ conclusively (i.e., attaining a negative value for $\epsilon+$), NSGA-II does very well compared to the NEMO approach in terms of this indicator.

Problem	NSGA-II		NEMO$_{PAES}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	863.67	126.84	**5788.2**	2295.89	1.07E-007
oka1	**721.83**	105.15	11.1	10.98	2.98E-011
oka2	**29.03**	7.82	1.13	0.35	4.03E-012
vlmop2	2412.37	42.36	**8097.97**	121.75	3.01E-011
vlmop3	465.83	172.62	**10533.8**	172.51	3.01E-011
dtlz1a	**2756.23**	244.18	806.47	288.01	3.02E-011
dtlz2a	2795.47	357.41	**8203.43**	357.41	3.02E-011
dtlz4a	**7863.73**	4430.74	4.3	4.92	0
dtlz7a	**5609.97**	387.23	2764.8	1453.37	1.20E-008
sdflp	1027.97	30.6	**3592.17**	326.18	3.02E-011

Table 5.11: Pareto Size Indicators for NSGA-II and NEMO$_{PAES}$

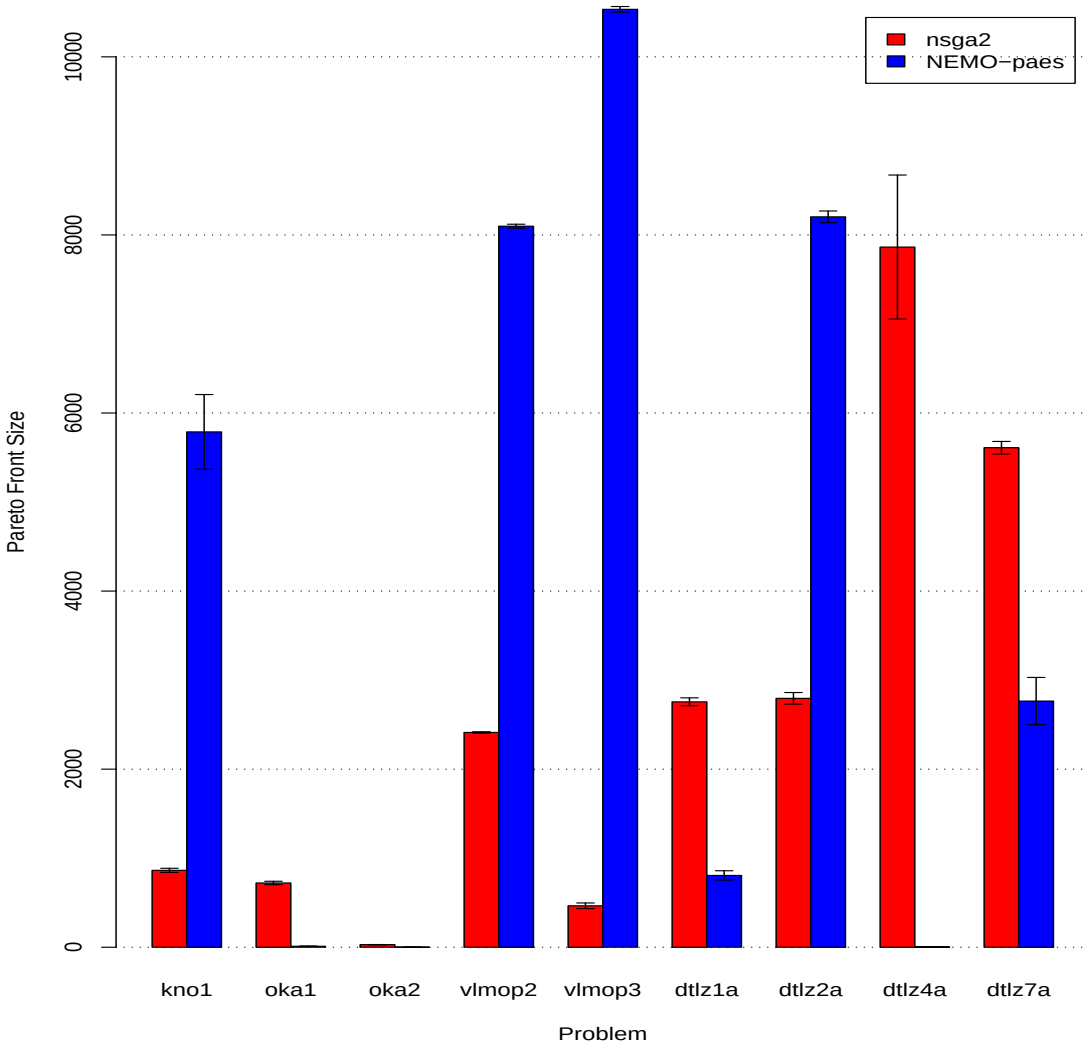


Figure 5.9: Pareto Size Indicators for NSGA-II and NEMO$_{PAES}$ with Standard Error Bars.

Problem	NSGA-II		NEMO$_{PAES}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	**0.01**	0.01	0.02	0.04	0
oka1	**0.05**	0.04	0.5	0.58	0.02
oka2	0.37	0.33	**0**	0	1.19E-012
vlmop2	0	0	0	0	NaN
vlmop3	0.02	0.01	**0**	0	5.09E-013
dtlz1a	**0**	0	13.64	11.3	2.07E-011
dtlz2a	0.01	0	**0**	0	3.78E-010
dtlz4a	0.01	0	0.28	0.45	0.34
dtlz7a	0.01	0.01	0.01	0	0.01
sdflp	3.59	2.37	**1.17**	1.24	5.25E-007

Table 5.12: Spacing Indicators for NSGA-II and NEMO$_{PAES}$

Problem	NSGA-II		NEMO$_{PAES}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	27593.36	4615.17	**210057.82**	82407.67	1.07E-007
oka1	**857.13**	231.13	11.31	12.66	3.02E-011
oka2	**46.9**	45.1	0.26	0.81	3.17E-011
vlmop2	228.48	3.74	**613.15**	9.89	3.02E-011
vlmop3	678.68	204.08	**5379.99**	313.68	3.02E-011
dtlz1a	**253036.69**	197674.82	4718.41	5041.25	0
dtlz2a	185.34	35.88	**372.58**	27.55	3.02E-011
dtlz4a	**709.54**	510.14	0.04	0.07	8.30E-005
dtlz7a	**1194.34**	1587.76	1044.37	1483.07	0
sdflp	115466666.67	4023494.22	**404233333.33**	42154627.78	2.95E-011

Table 5.13: Hypervolume Indicators for NSGA-II and NEMO$_{PAES}$

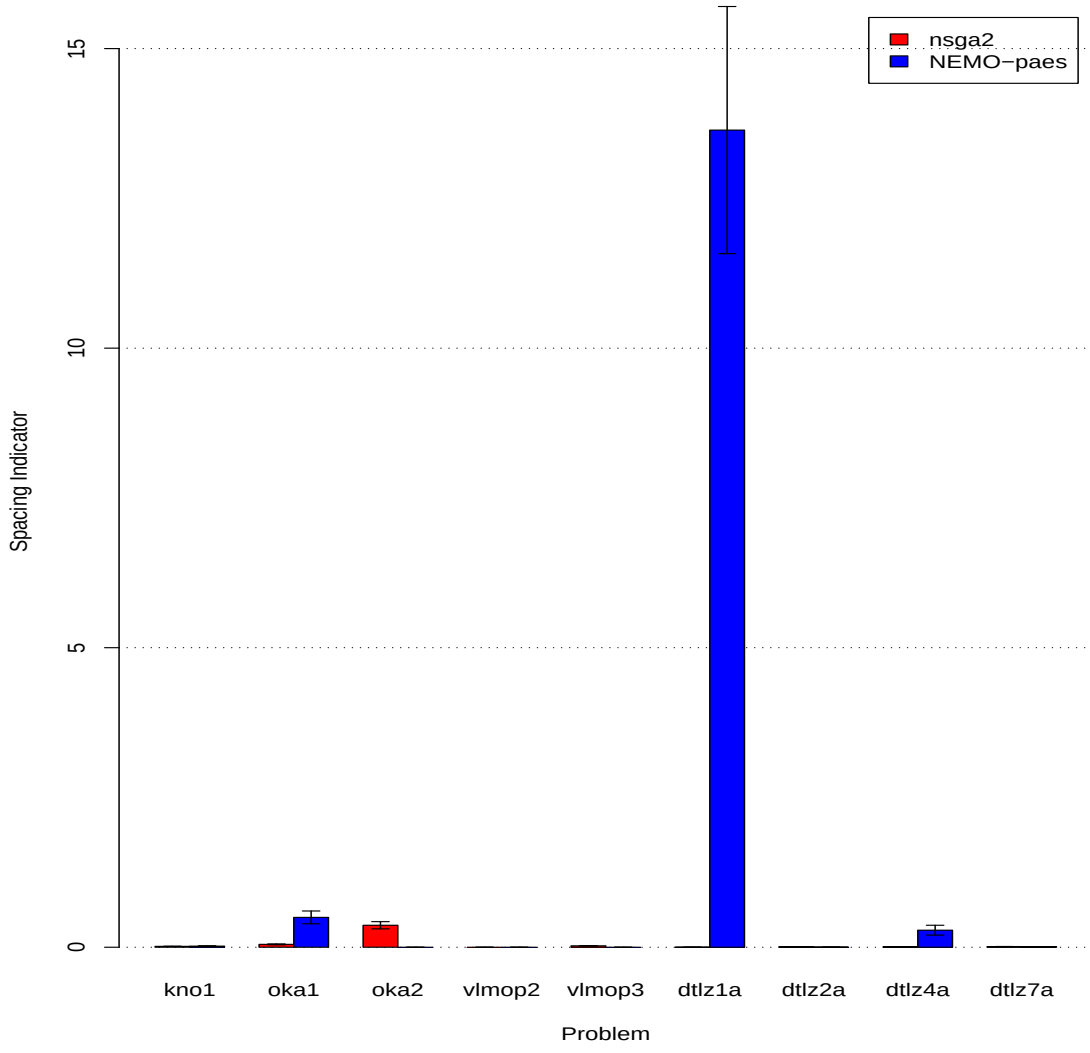


Figure 5.10: Spacing Indicators for NSGA-II and NEMO$_{PAES}$ with Standard Error Bars.

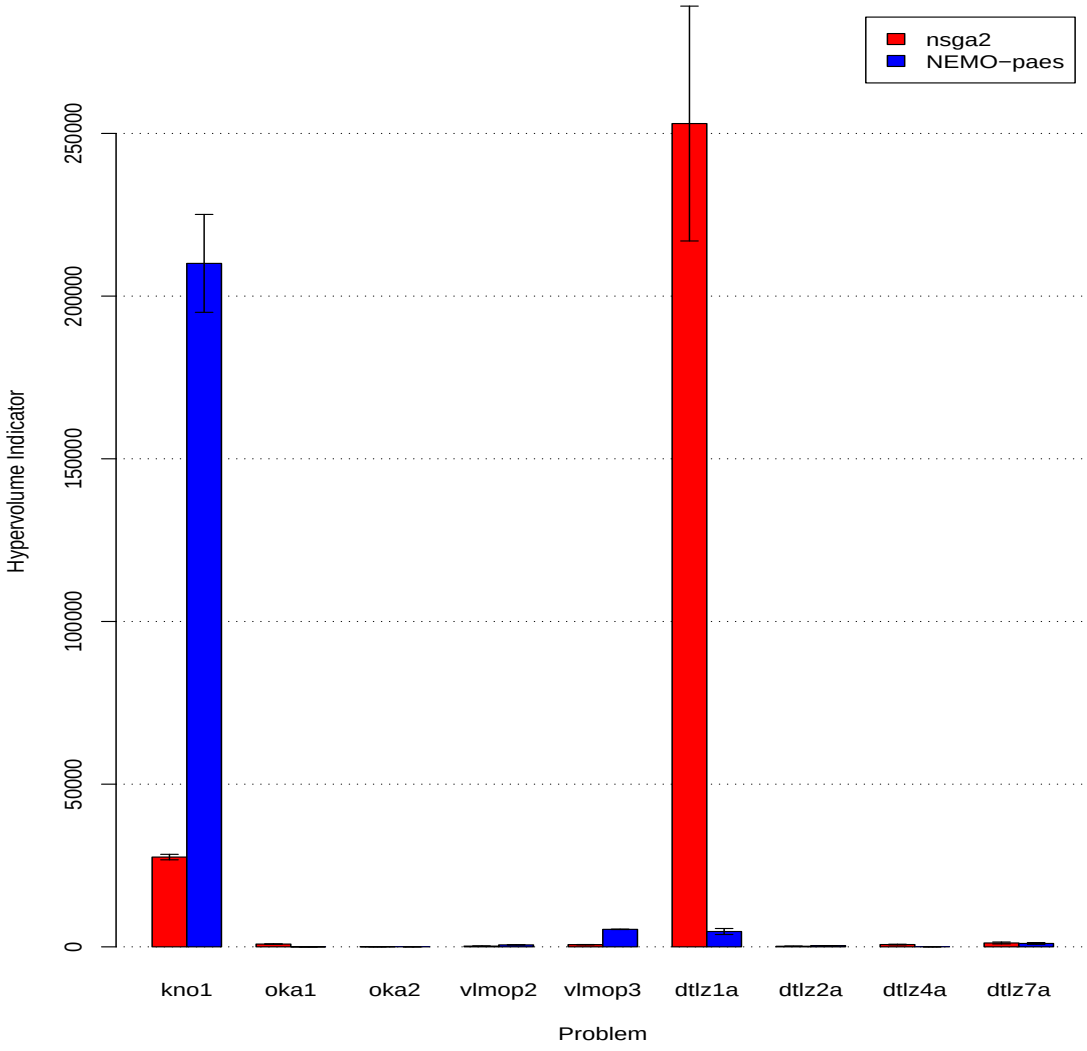


Figure 5.11: Hypervolume Indicators for NSGA-II and NEMO$_{PAES}$ with Standard Error Bars.

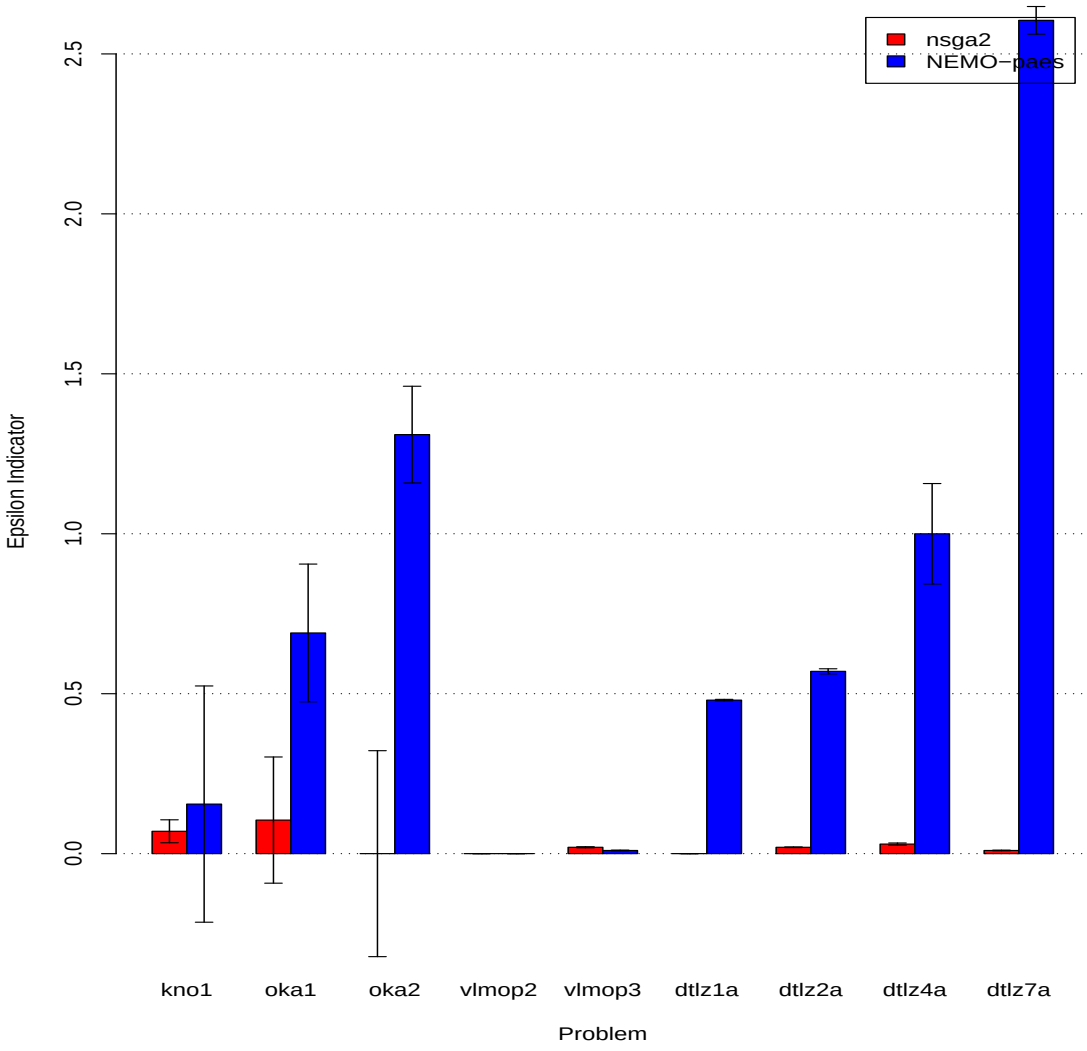


Figure 5.12: $\epsilon+$ Indicators for NSGA-II and NEMO$_{PAES}$ with Standard Error Bars.

Problem	NSGA-II		NEMO$_{PAES}$		p-value
	Median	IQR	Median	IQR	
kno1	**0.07**	0.02	0.16	0.07	1.77E-009
oka1	**0.11**	2.18	0.69	1.79	0
oka2	**2.65E-006**	3.03	1.31	1.55	0.05
vlmop2	0	0	0	0	NaN
vlmop3	0.02	0.01	**0.01**	0	8.43E-007
dtlz1a	**0**	0	0.48	0.01	8.73E-013
dtlz2a	**0.02**	0	0.57	0.06	8.99E-012
dtlz4a	**0.03**	0.02	1	0.02	0
dtlz7a	**0.01**	0.01	2.61	0.08	1.22E-011
sdflp	**4.07**	167.66	170.04	165.96	0.02

Table 5.14: $\epsilon+$ Indicators for NSGA-II and NEMO$_{PAES}$

5.4.4 MOPSO versus NEMO$_{NSGA-II}$

In this comparison, NSGA-II was used as the underlying EMO algorithm to train NEMO and compared against MOPSO. Tables 5.15-5.17 show the average Pareto optimal front size, spacing indicator, and hypervolume indicator, and Table 5.18 shows the median and interquartile ranges for the binary $\epsilon+$ indicator. Figures 5.13-5.16 display the average and standard errors of the optimal front sizes, spacing, and hypervolume indicators, and the median and standard errors of the binary $\epsilon+$ indicators.

Not surprisingly, NEMO$_{NSGA-II}$ outperforms MOPSO in 80% of the problems in terms of the size of the Pareto optimal set. It outperforms MOPSO in terms of the spacing indicator on 40% of the problems, while being outperformed on only 20% of the problems. For the hypervolume indicator, NEMO$_{NSGA-II}$ outperforms MOPSO in 80% of the problems

in the test suite. Finally, on the binary $\epsilon+$ indicator, NEMO$_{NSGA=II}$ outperforms MOPSO on 60% of the problems (though without any conclusive results).

Problem	MOPSO		NEMO$_{NSGA-II}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	**5693.4**	1244.95	5252.57	1162.38	0
oka1	31.07	14.89	**484.17**	122.62	3.45E-010
oka2	5.5	6.78	**35.07**	28.16	1.80E-009
vlmop2	4047.77	186.34	**6952**	186.3	3.02E-011
vlmop3	1491.77	378.88	**9507.6**	378.6	3.02E-011
dtlz1a	5152.43	3118.65	5801.17	3139.25	0.67
dtlz2a	48.1	50.4	**10950.9**	50.4	2.93E-011
dtlz4a	357.2	361.53	**7817.9**	1521.3	3.02E-011
dtlz7a	488.53	1323.81	**5565.4**	712.02	5.76E-011
sdflp	954.3	49.87	**4446.3**	428.86	3.01E-011

Table 5.15: Pareto Size Indicators for MOPSO and NEMO$_{NSGA-II}$

5.4.5 MOPSO versus NEMO$_{MOPSO}$

In this comparison, MOPSO was used as the underlying EMO algorithm to train NEMO and compared against standard MOPSO. This is similar to the experiment carried out in the previous chapter in which NSGA-II was used as the underlying EMO. Like that experiment, this one is a "pure" evaluation of the NEMO approach since it uses the same EMO for the training set and for the comparative set. Tables 5.19-5.21 show the average Pareto optimal front size, spacing indicator, and hypervolume indicator, and Table 5.22 shows the median and interquartile ranges for the binary $\epsilon+$ indicator. Additionally, Figures 5.17-5.20 display the average and standard errors of the optimal front sizes, spacing, and hypervolume indicators, and the median and standard errors of the binary $\epsilon+$ indicators.

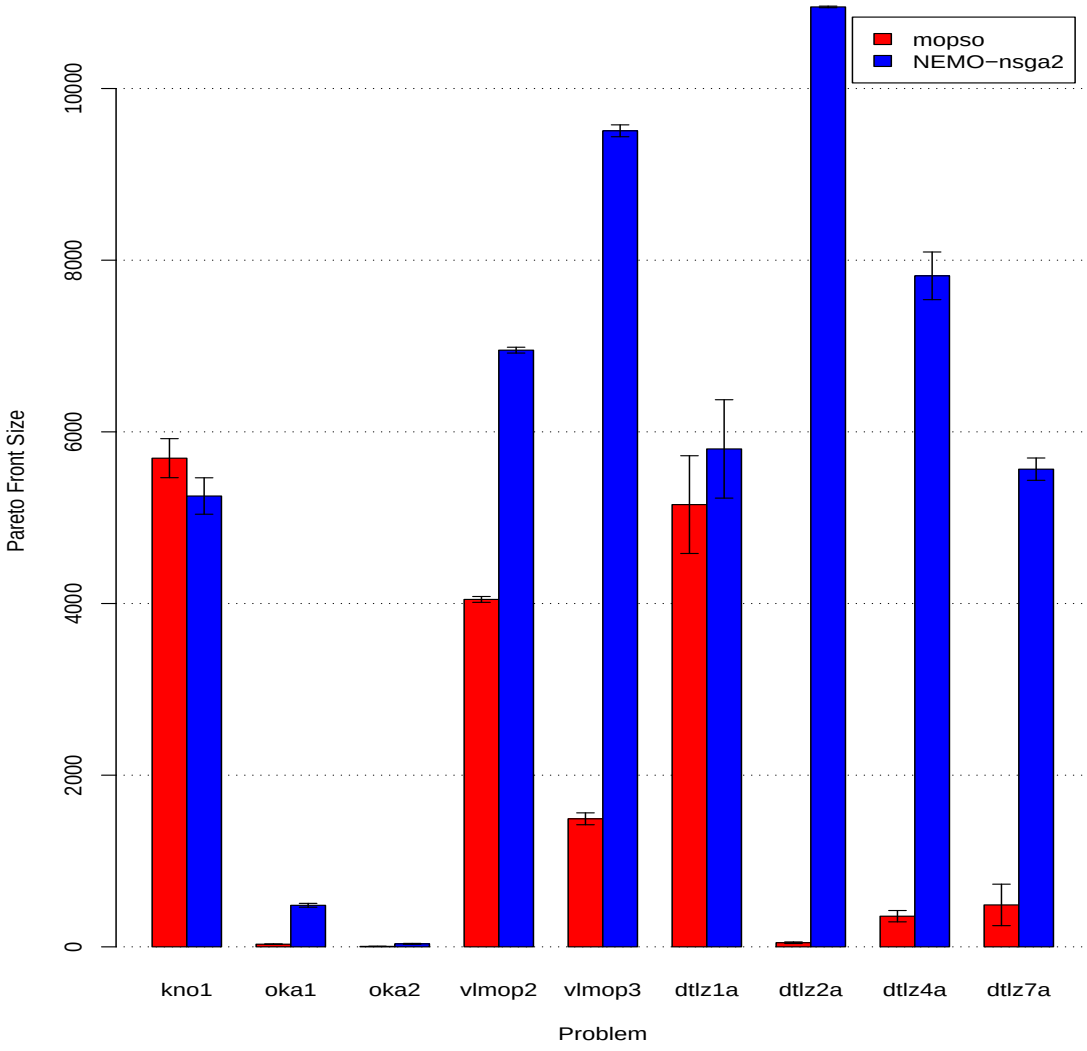


Figure 5.13: Pareto Size Indicators for MOPSO and NEMO$_{NSGA-II}$ with Standard Error Bars.

Problem	MOPSO		NEMO$_{NSGA-II}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	**0**	0	0.01	0.01	3.34E-011
oka1	0.15	0.19	0.09	0.1	0.74
oka2	0.85	0.97	0.35	0.51	0.5
vlmop2	0	0	0	0	NaN
vlmop3	0.01	0	**0**	0	1.55E-013
dtlz1a	0	0.01	0	0	0.63
dtlz2a	0.04	0.04	**0.01**	0	1.10E-005
dtlz4a	0.04	0.04	**0.01**	0	0
dtlz7a	0.15	0.18	**0.02**	0.02	0
sdflp	**1.41**	1.43	2.26	0.82	0

Table 5.16: Spacing Indicators for MOPSO and NEMO$_{NSGA-II}$

Problem	MOPSO		NEMO$_{NSGA-II}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	**212313.37**	46357.67	179715.17	39159.2	2.88E-006
oka1	34.49	19.32	**594.16**	231.79	2.39E-008
oka2	1.78	2.72	**12.52**	9.71	1.23E-009
vlmop2	390.29	18.48	**526.68**	12.57	3.02E-011
vlmop3	837.76	264.35	**6549.11**	597.26	3.02E-011
dtlz1a	274.52	165.17	205.67	137.71	0.09
dtlz2a	1.42	1.69	**619.68**	165.65	2.95E-011
dtlz4a	19.27	22.32	**452.21**	126.93	3.02E-011
dtlz7a	119.61	353.65	**4547.41**	3695.88	1.46E-010
sdflp	96530000	3838565.64	**560366666.67**	66567302.39	3.01E-011

Table 5.17: Hypervolume Indicators for MOPSO and NEMO$_{NSGA-II}$

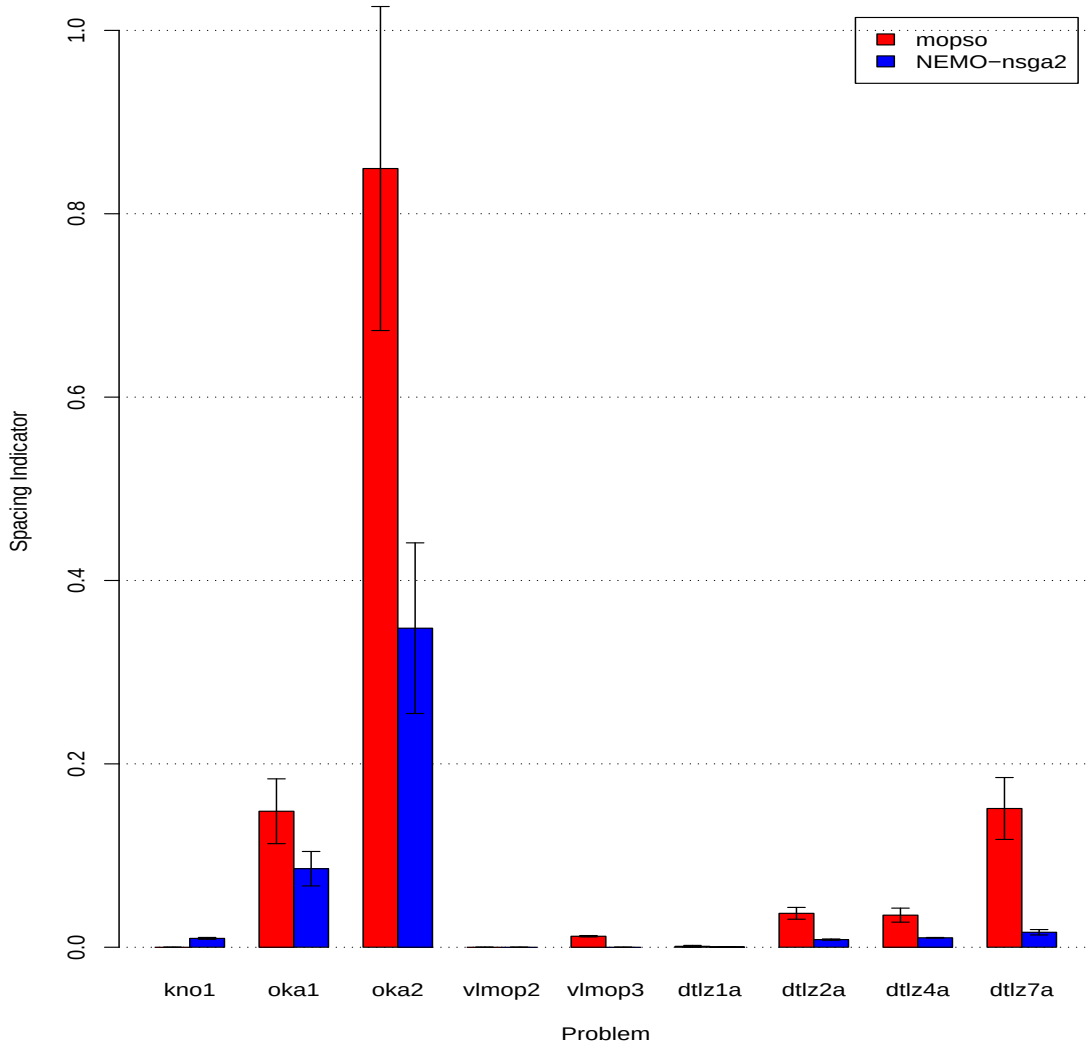


Figure 5.14: Spacing Indicators for MOPSO and NEMO$_{NSGA-II}$ with Standard Error Bars.

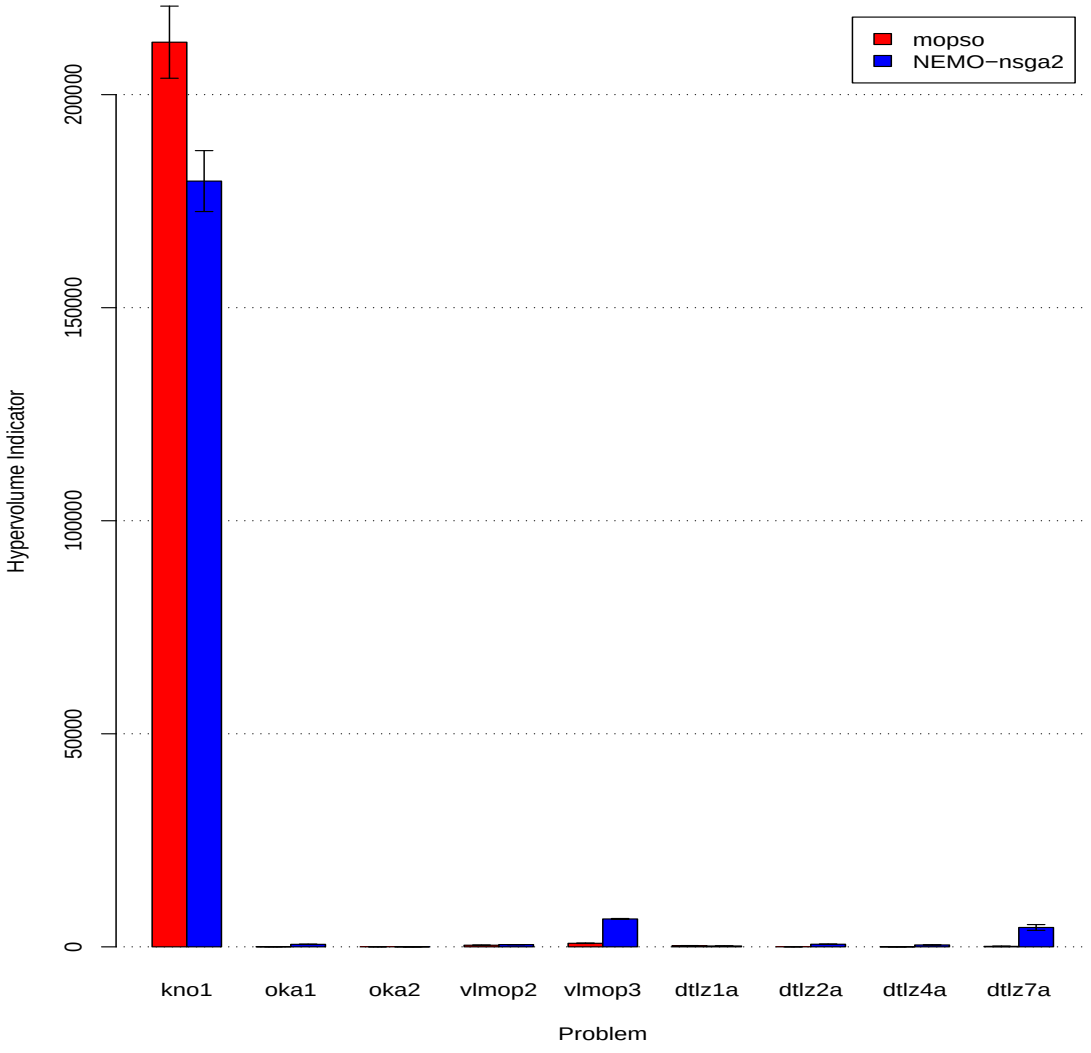


Figure 5.15: Hypervolume Indicators for MOPSO and NEMO$_{NSGA-II}$ with Standard Error Bars.

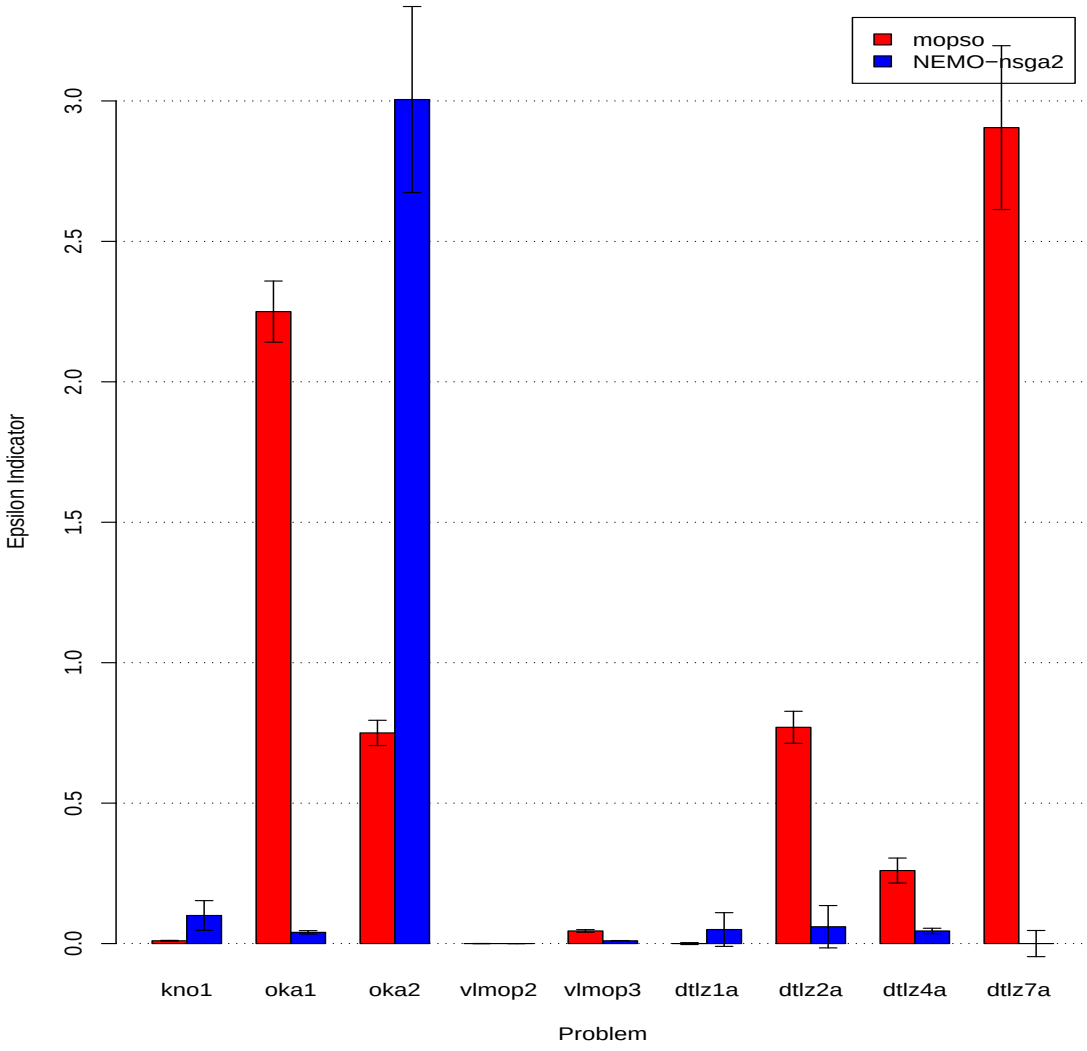


Figure 5.16: $\epsilon+$ Indicators for MOPSO and NEMO$_{NSGA-II}$ with Standard Error Bars.

Problem	MOPSO		NEMO$_{NSGA-II}$		p-value
	Median	IQR	Median	IQR	
kno1	**0.01**	0.01	0.1	0.08	3.39E-010
oka1	2.25	0.06	**0.04**	0.05	2.75E-011
oka2	0.75	0.39	3.01	3.65	0.32
vlmop2	0	0	0	0	0.33
vlmop3	0.05	0.04	**0.01**	0	1.24E-010
dtlz1a	**0**	0	0.05	0.05	4.56E-007
dtlz2a	0.77	0.19	**0.06**	0.1	1.30E-006
dtlz4a	0.26	0.38	**0.05**	0.06	1.30E-010
dtlz7a	2.91	1.5	**2.17E-005**	0.01	3.67E-009
sdflp	170.11	0.2	**1.58**	0.9	2.03E-010

Table 5.18: $\epsilon+$ Indicators for MOPSO and NEMO$_{NSGA-II}$

Not surprisingly, NEMO$_{MOPSO}$ outperforms MOPSO in 70% of the problems in terms of the size of the Pareto optimal set. It outperforms MOPSO in terms of the spacing indicator on 60% of the problems, while being outperformed on none of the problems. For the hypervolume indicator, NEMO$_{MOPSO}$ outperforms MOPSO in 60% of the problems in the test suite. Finally, on the binary $\epsilon+$ indicator, MOPSO outperforms NEMO$_{MOPSO}$ on 60% of the problems (though without any conclusive results).

5.4.6 MOPSO versus NEMO$_{PAES}$

In this comparison, PAES was used as the underlying EMO algorithm to train NEMO and compared against MOPSO. Tables 5.23-5.25 show the average Pareto optimal front size, spacing indicator, and hypervolume indicator, while Table 5.26 shows the median and interquartile ranges for the binary $\epsilon+$ indicator. Figures 5.21-5.24 display the average and

Problem	MOPSO		NEMO$_{MOPSO}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	1476.47	1037.93	**9433.67**	2353.51	9.75E-010
oka1	**63.5**	28.31	38.2	21.7	0
oka2	14.23	19.13	13.93	32.04	0.44
vlmop2	2597.33	172.99	**10539.07**	74.36	3.02E-011
vlmop3	1339.73	1547.22	**10508.97**	778.98	3.02E-011
dtlz1a	3799.57	2007.53	**6382.9**	3376.67	0
dtlz2a	2114.17	1047.76	**9097.13**	1031.99	3.02E-011
dtlz4a	480.83	797.17	1131.8	1665.3	0.16
dtlz7a	3374.77	2372.77	**9332.1**	1951.78	9.52E-010
sdflp	1288.53	52.12	**3567.2**	430.29	3.01E-011

Table 5.19: Pareto Size Indicators for MOPSO and NEMO$_{MOPSO}$

Problem	MOPSO		NEMO$_{MOPSO}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	0.02	0.01	**0.01**	0.01	2.41E-005
oka1	0.23	0.14	0.19	0.16	0.3
oka2	0.46	0.55	0.23	0.6	0.23
vlmop2	0	0	0	0	NaN
vlmop3	0.01	0.01	**0**	0	3.55E-011
dtlz1a	0	0	0	0.01	0.08
dtlz2a	0.02	0.01	**0.01**	0.01	1.15E-007
dtlz4a	0.02	0.02	**0.01**	0.01	0.05
dtlz7a	0.02	0.01	**0.01**	0.01	0
sdflp	4.8	0.1	**1.94**	1.29	2.98E-011

Table 5.20: Spacing Indicators for MOPSO and NEMO$_{MOPSO}$

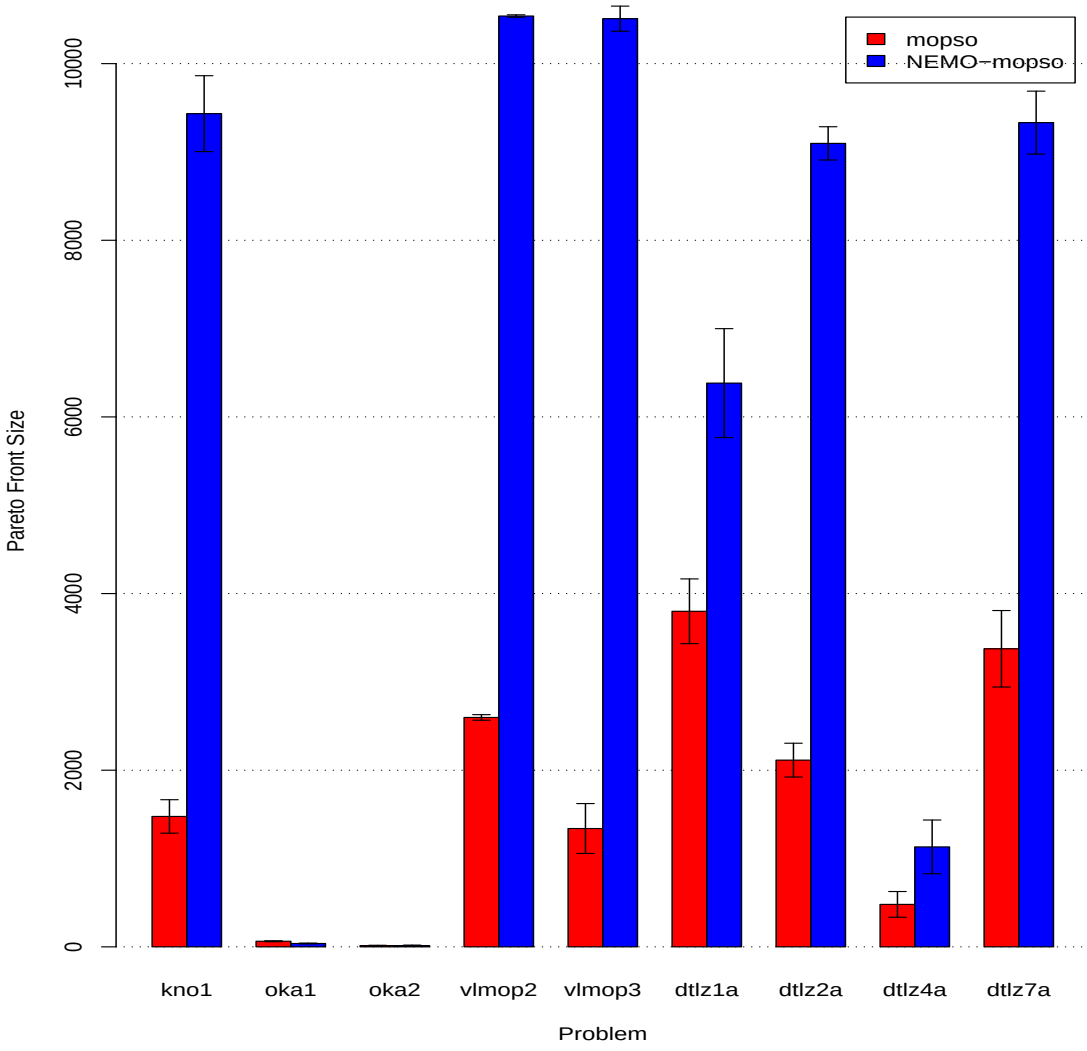


Figure 5.17: Pareto Size Indicators for MOPSO and NEMO$_{MOPSO}$ with Standard Error Bars.

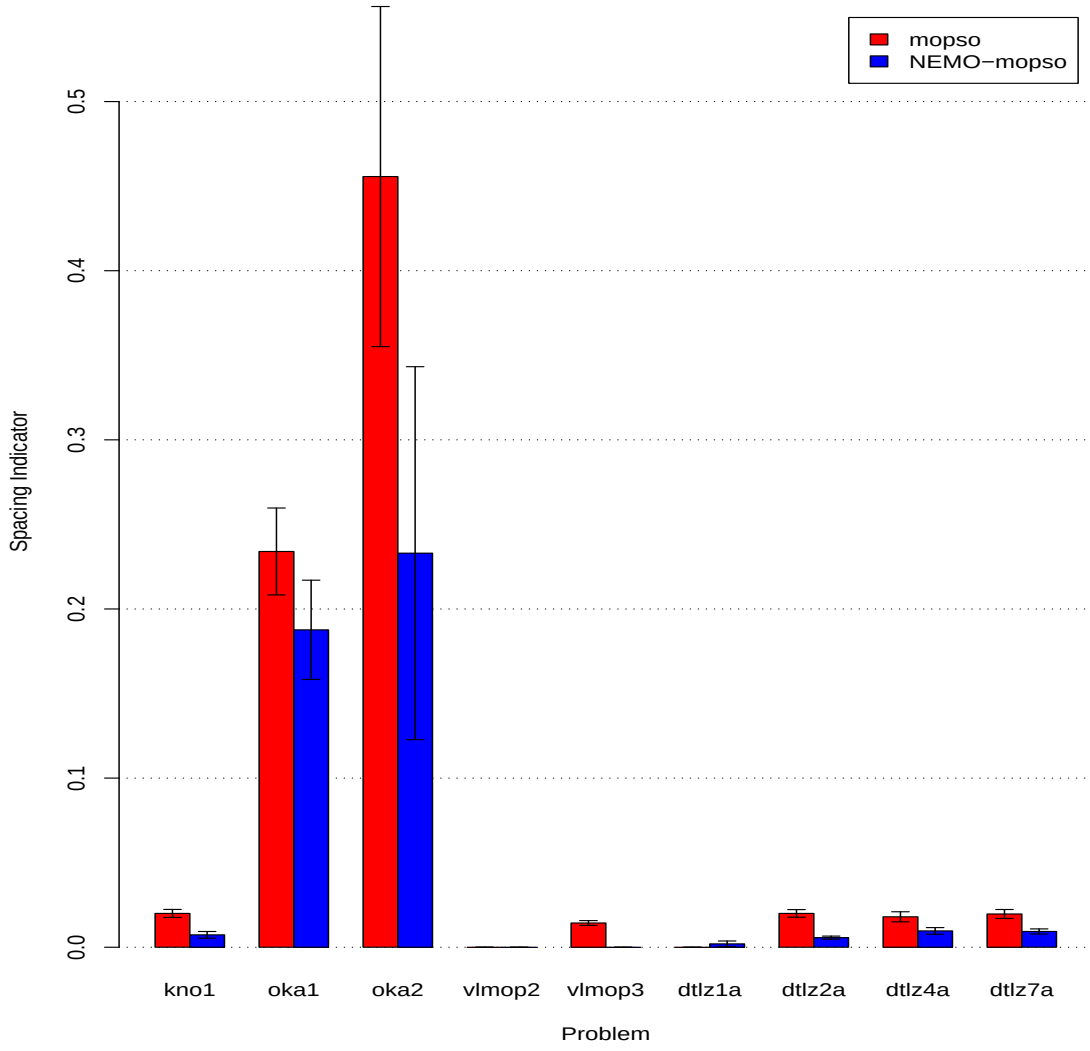


Figure 5.18: Spacing Indicators for MOPSO and NEMO$_{MOPSO}$ with Standard Error Bars.

Problem	MOPSO		NEMO$_{MOPSO}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	51648.34	39912.43	**320608.43**	82603.25	8.10E-010
oka1	**24.07**	18.05	14.37	9.86	0.03
oka2	2.67	4.24	2.16	7.29	0.53
vlmop2	252.39	17.14	**836.49**	7.78	3.02E-011
vlmop3	898.78	962.6	**5989.47**	1244.66	3.02E-011
dtlz1a	191.58	111.35	244.04	134.87	0.08
dtlz2a	36.7	25.84	**283.96**	253.61	6.70E-011
dtlz4a	21.38	36.56	74.24	127.22	0.22
dtlz7a	91.89	139.85	**958.32**	1449.83	2.38E-007
sdflp	137833333.33	4202489.68	**423766666.67**	60634395.23	2.96E-011

Table 5.21: Hypervolume Indicators for MOPSO and NEMO$_{MOPSO}$

Problem	MOPSO		NEMO$_{MOPSO}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	**0.05**	0.02	0.3	0.33	1.21E-008
oka1	**0.04**	0.1	0.13	0.16	0
oka2	**0**	0.05	0.05	0.6	0.03
vlmop2	0	0	0	0	NaN
vlmop3	0.03	0.02	**0**	0.01	5.39E-011
dtlz1a	**0**	0	0.05	0.08	1.15E-012
dtlz2a	0.17	0.12	**0.09**	0.05	0
dtlz4a	**0.01**	0.02	0.24	0.49	0
dtlz7a	0.1	0.2	**0**	0.01	1.53E-008
sdflp	**2.82**	3.36	5.11	166.89	0

Table 5.22: $\epsilon+$ Indicators for MOPSO and NEMO$_{MOPSO}$

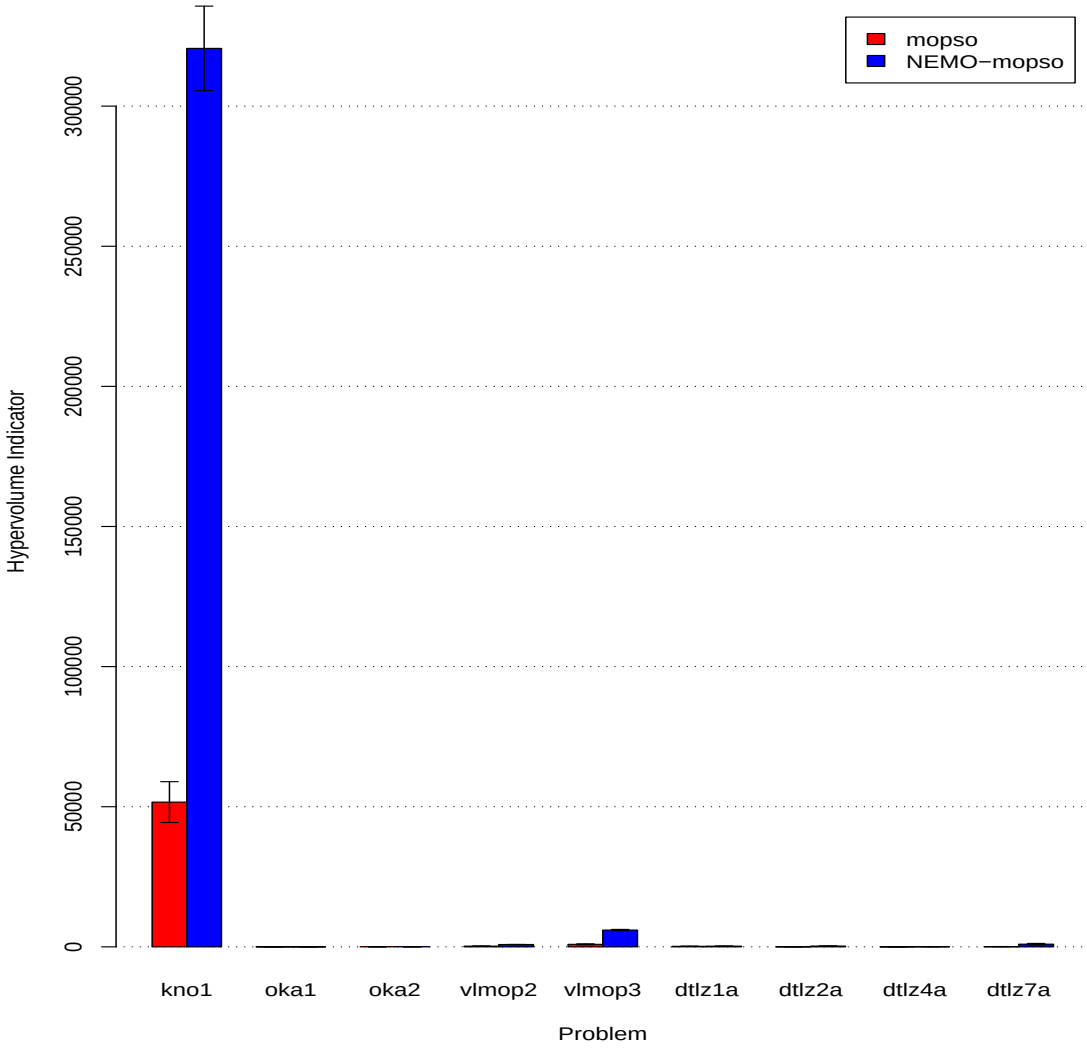


Figure 5.19: Hypervolume Indicators for MOPSO and NEMO$_{MOPSO}$ with Standard Error Bars.

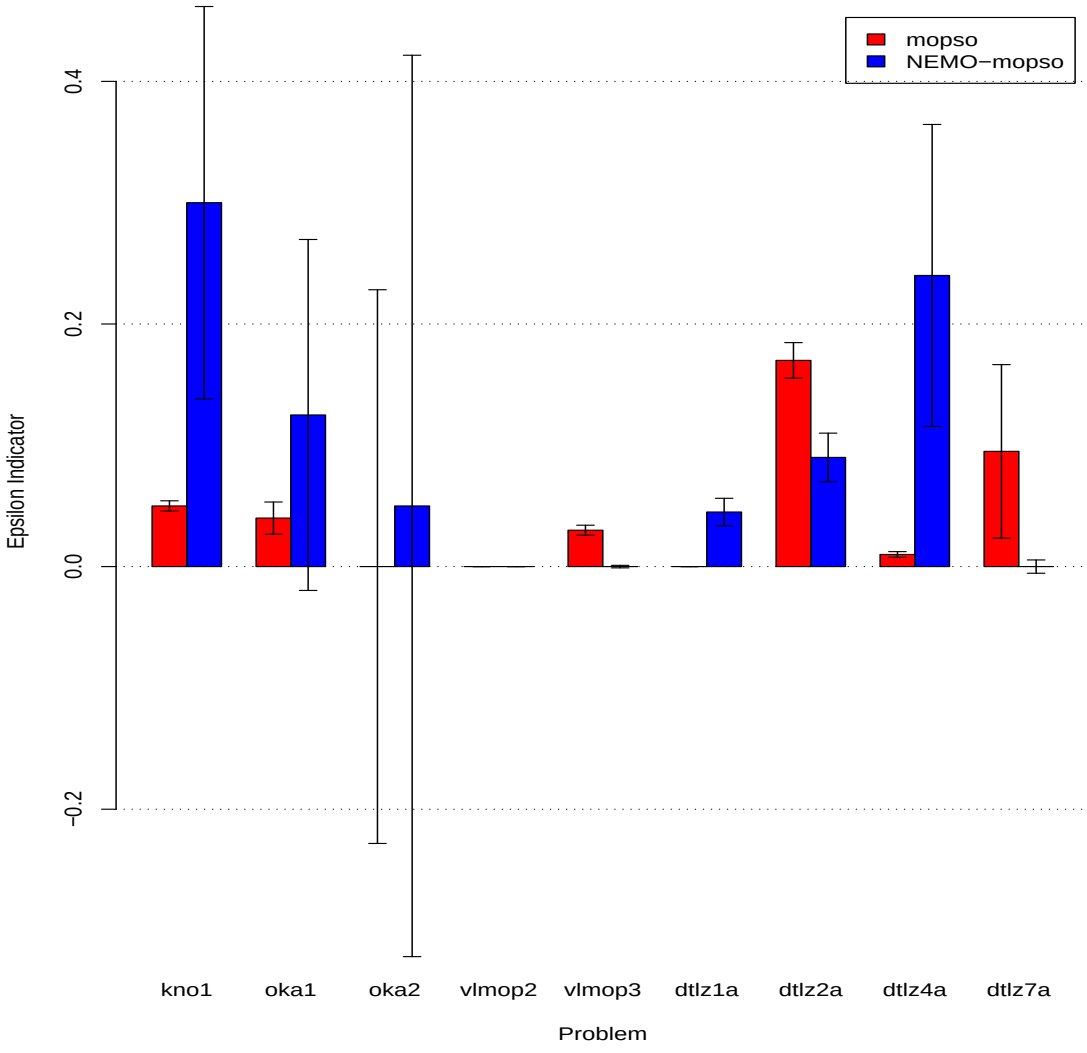


Figure 5.20: $\epsilon+$ Indicators for MOPSO and NEMO$_{MOPSO}$ with Standard Error Bars.

standard errors of the optimal front sizes, spacing, and hypervolume indicators, and the median and standard errors of the binary $\epsilon+$ indicators.

NEMO$_{PAES}$ outperforms MOPSO in 50% of the problems in terms of the size of the Pareto optimal set. It outperforms MOPSO in terms of the spacing indicator on 40% of the problems, while being outperformed on only 20% of the problems. For the hypervolume indicator, NEMO$_{PAES}$ outperforms MOPSO in 50% of the problems in the test suite. Finally, on the binary $\epsilon+$ indicator, MOPSO outperforms NEMO$_{PAES}$ on 40% of the problems. On OKA2, the performance was conclusive, yielding a negative median value. However, NEMO$_{PAES}$ was able to outperform MOPSO in 30% of the problems, though none were conclusive.

Problem	MOPSO		NEMO$_{PAES}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	**8009.97**	2006.63	2411.07	1717.02	6.23E-009
oka1	**41.87**	25.26	20.1	19.47	6.01E-006
oka2	**13.53**	18.6	0.33	0.66	4.62E-009
vlmop2	4114.97	157.18	**6884.6**	157.21	3.02E-011
vlmop3	302.5	148.67	**10697.1**	148.54	3.01E-011
dtlz1a	**8982.03**	2029.81	1049.67	424.34	3.02E-011
dtlz2a	10.87	20.37	**10988.23**	20.43	2.30E-011
dtlz4a	**1008.53**	1410.45	16.93	24.28	0
dtlz7a	3561.6	2668.1	**5525.1**	1536.58	0.05
sdflp	951.23	43.88	**4109.73**	389.29	3.01E-011

Table 5.23: Pareto Size Indicators for MOPSO and NEMO$_{PAES}$

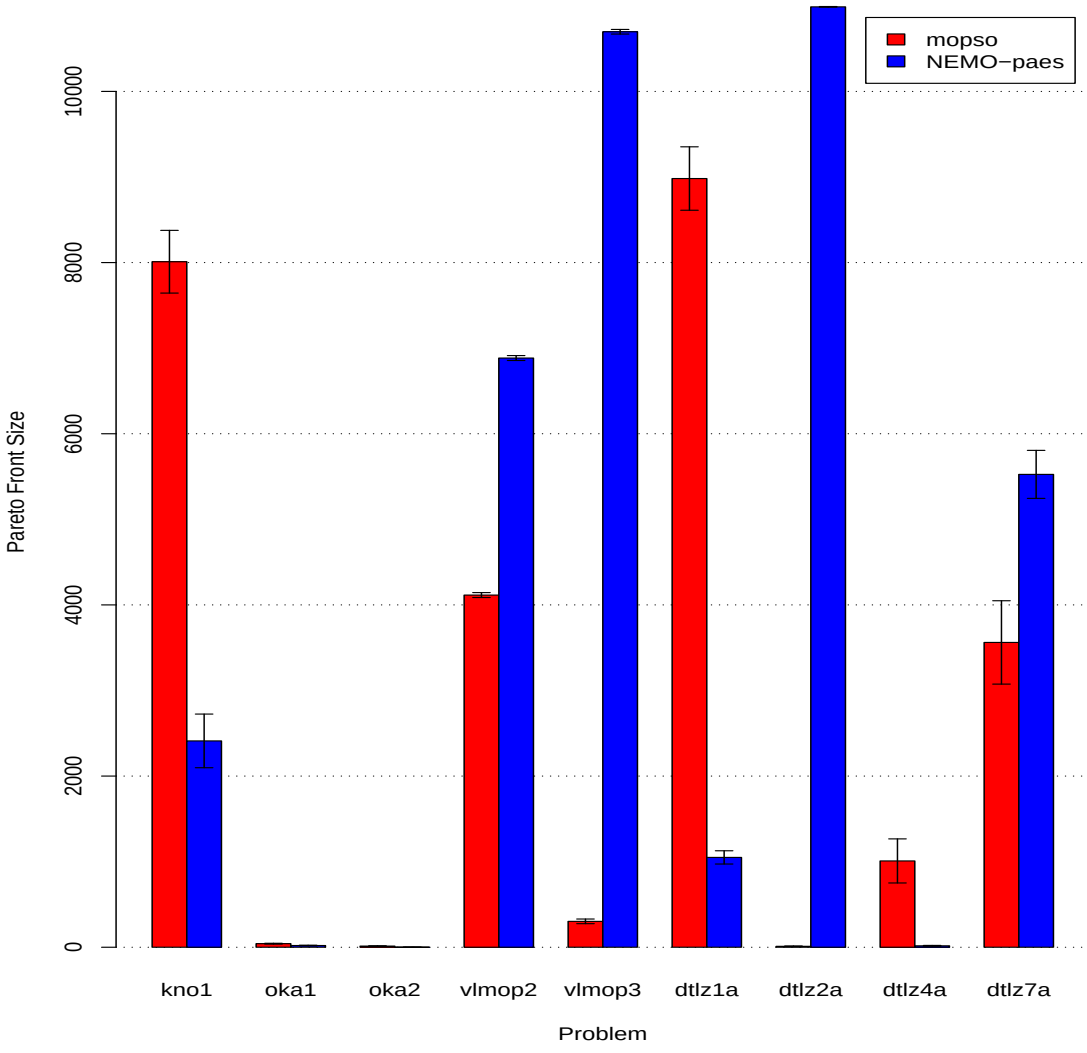


Figure 5.21: Pareto Size Indicators for MOPSO and NEMO$_{PAES}$ with Standard Error Bars.

Problem	MOPSO		NEMO$_{PAES}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	**0**	0	0.04	0.07	9.44E-012
oka1	**0.15**	0.16	0.52	0.43	0
oka2	0.35	0.52	**0**	0	5.36E-006
vlmop2	0	0	0	0	NaN
vlmop3	0.04	0.01	**0**	0	8.53E-013
dtlz1a	3.86E-005	4.51E-005	6.88E-006	7.70E-006	0.37
dtlz2a	0.07	0.11	**0**	0	0
dtlz4a	0.01	0.01	0.17	0.26	0.32
dtlz7a	0.02	0.05	**0**	0	0.03
sdflp	2.79	2.35	1.72	1.21	0.06

Table 5.24: Spacing Indicators for MOPSO and NEMO$_{PAES}$

Problem	MOPSO		NEMO$_{PAES}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	**298118.9**	74605.38	83326.47	61096.21	5.71E-009
oka1	**22.79**	10.72	13.76	13.02	0
oka2	**1.27**	2.01	0.03	0.14	2.40E-008
vlmop2	398.99	15.27	**500.73**	12.42	3.02E-011
vlmop3	291.23	152.77	**5725.36**	308.86	3.02E-011
dtlz1a	**481.19**	122.39	9.32	5.93	3.02E-011
dtlz2a	0.4	0.93	**287.66**	102.47	2.40E-011
dtlz4a	**56.86**	93.03	0.24	0.49	0
dtlz7a	48.84	94.38	**635.55**	633.14	1.00E-008
sdflp	101010000	3865796.08	**4.68E+008**	54288501.35	2.93E-011

Table 5.25: Hypervolume Indicators for MOPSO and NEMO$_{PAES}$

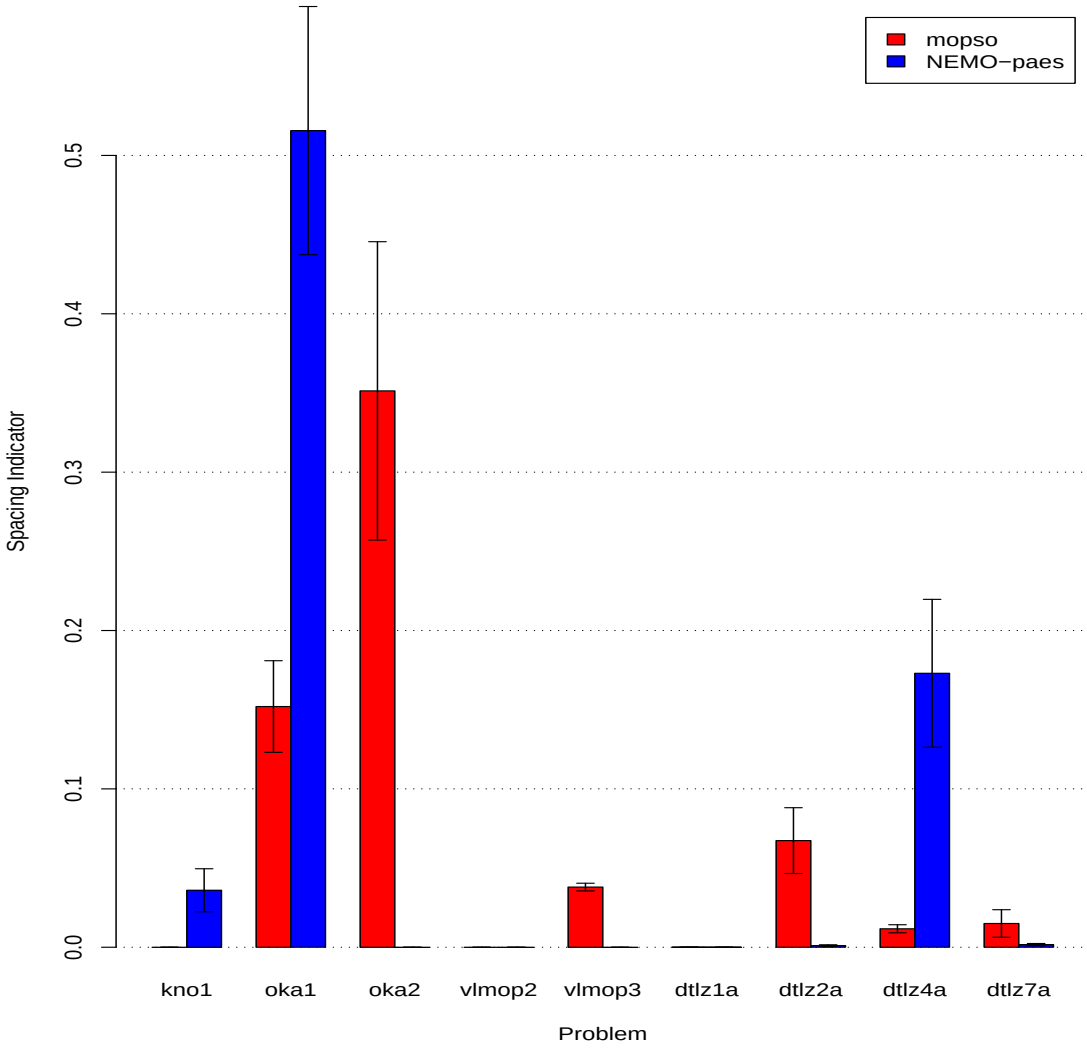


Figure 5.22: Spacing Indicators for MOPSO and NEMO$_{PAES}$ with Standard Error Bars.

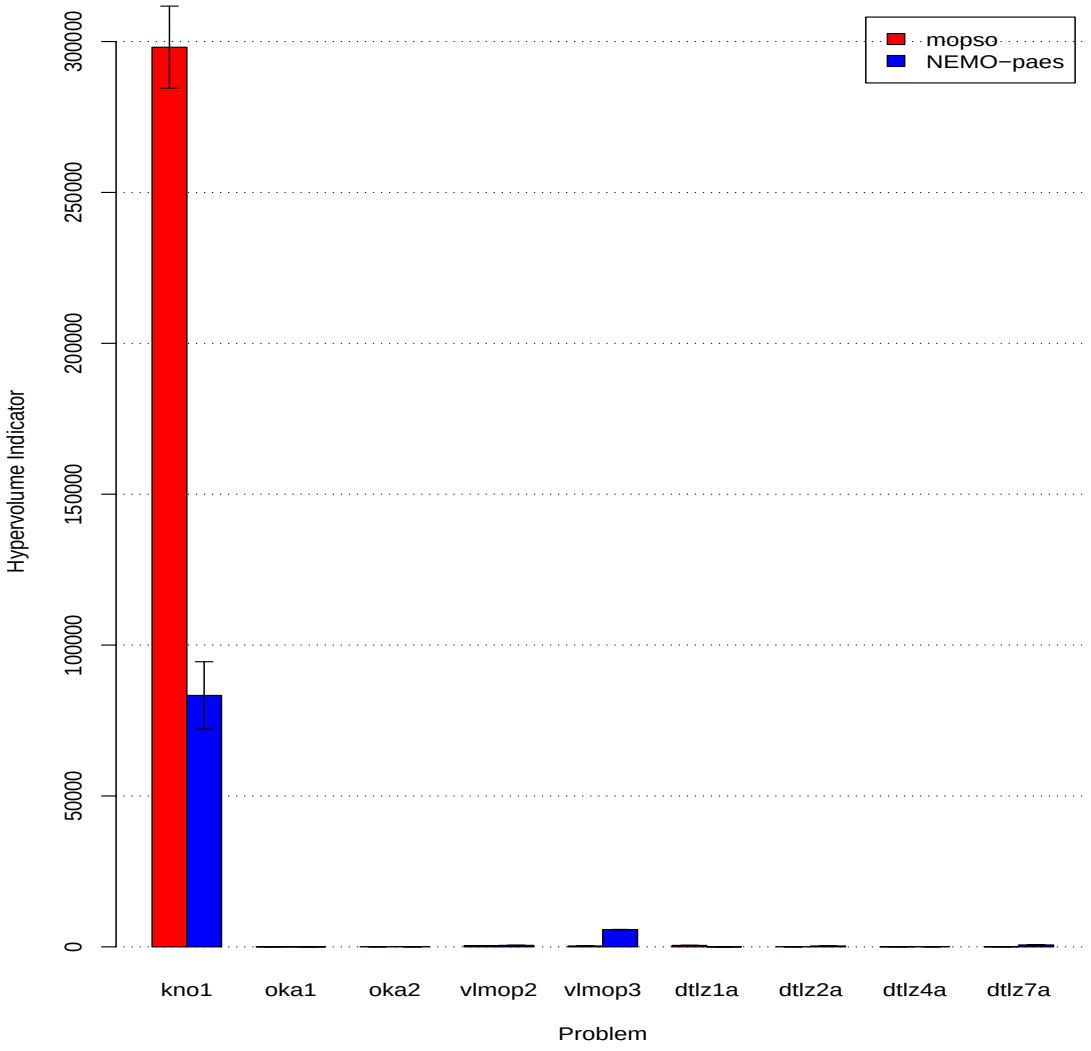


Figure 5.23: Hypervolume Indicators for MOPSO and NEMO$_{PAES}$ with Standard Error Bars.

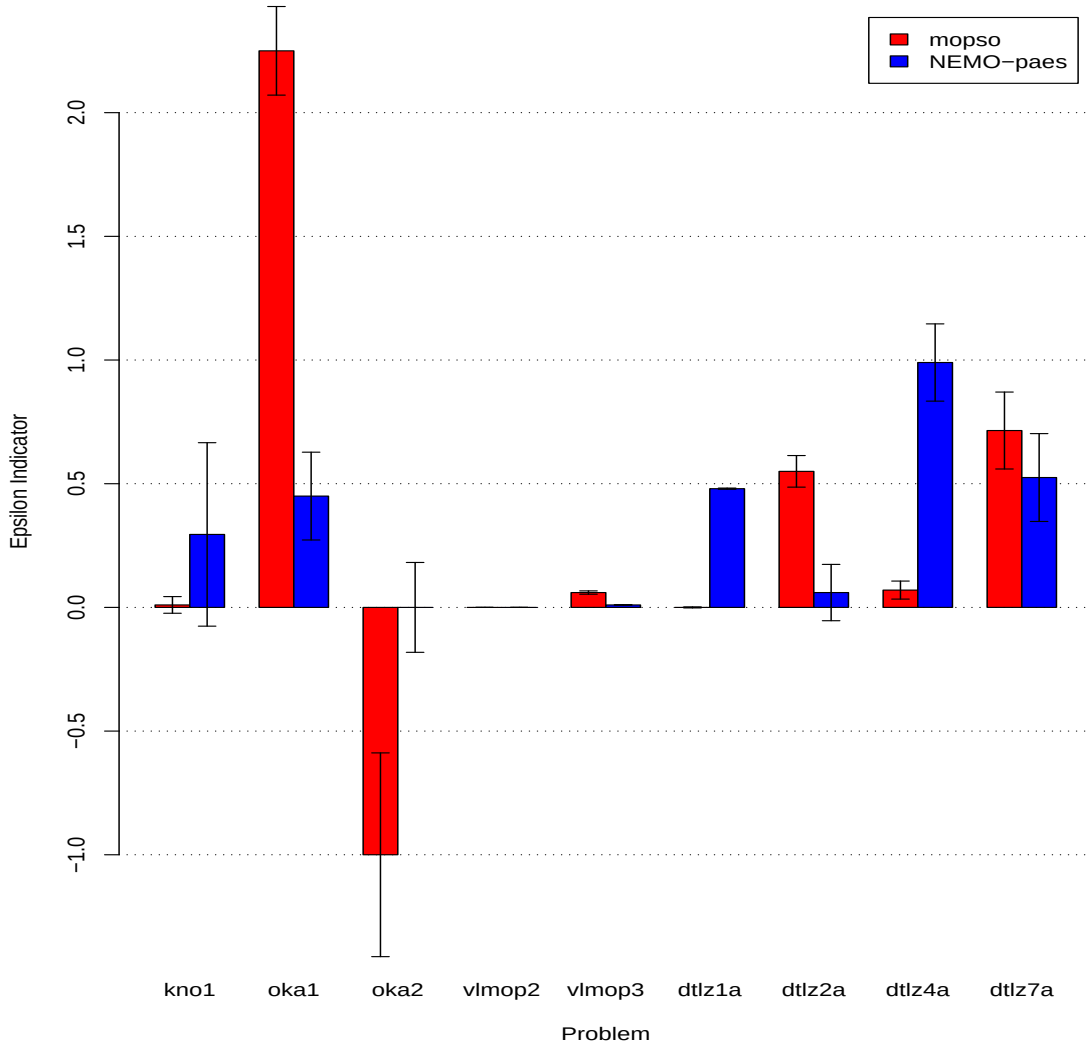


Figure 5.24: $\epsilon+$ Indicators for MOPSO and NEMO$_{PAES}$ with Standard Error Bars.

Problem	MOPSO		NEMO$_{PAES}$		p-value
	Median	IQR	Median	IQR	
kno1	**0.01**	0	0.3	0.24	2.19E-009
oka1	2.25	1.95	0.45	2.12	0.41
oka2	**-1**	0	0	0	6.08E-005
vlmop2	0	0	0	0	NaN
vlmop3	0.06	0.03	**0.01**	0.01	1.61E-011
dtlz1a	**0**	0.01	0.48	0.01	7.38E-012
dtlz2a	0.55	0.7	**0.06**	1.28	0
dtlz4a	**0.07**	0.07	0.99	0.07	0
dtlz7a	0.72	0.34	0.53	1.14	0.62
sdflp	170.02	148.99	**1.91**	168.64	0

Table 5.26: $\epsilon+$ Indicators for MOPSO and NEMO$_{PAES}$

5.4.7 PAES versus NEMO$_{NSGA-II}$

In this comparison, NSGA-II was used as the underlying EMO algorithm to train NEMO and compared against PAES. Tables 5.27-5.29 show the average Pareto optimal front size, spacing indicator, and hypervolume indicator, and Table 5.30 shows the median and interquartile ranges for the binary $\epsilon+$ indicator. Likewise, Figures 5.25-5.28 display the average and standard errors of the optimal front sizes, spacing, and hypervolume indicators, and the median and standard errors of the binary $\epsilon+$ indicators.

NEMO$_{NSGA-II}$ outperforms PAES in 100% of the problems in terms of the size of the Pareto optimal set. It outperforms PAES in terms of the spacing indicator on 40% of the problems, while being outperformed on only 10% of the problems. For the hypervolume indicator, NEMO$_{NSGA-II}$ outperforms PAES in 100% of the problems in the test suite.

Finally, on the binary $\epsilon+$ indicator, NEMO$_{NSGA-II}$ outperforms PAES on 90% of the problems, and, on DTLZ2a, its performance is conclusive.

Problem	PAES		NEMO$_{NSGA-II}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	90.8	46.53	**10397.27**	153.92	3.01E-011
oka1	10.03	11.47	**452.27**	108.65	4.91E-011
oka2	1.17	0.46	**46.13**	47.29	4.07E-012
vlmop2	1050.27	33.96	**8425.47**	147.38	3.01E-011
vlmop3	1532.2	316.78	**9465.67**	315.62	3.02E-011
dtlz1a	1.7	2.35	**10997.23**	2.65	1.85E-011
dtlz2a	1.33	2.34	**10997.67**	2.34	9.85E-012
dtlz4a	4.63	4.44	**8058**	1439.77	2.74E-011
dtlz7a	471.77	117.47	**4948.57**	552.63	3.02E-011
sdflp	1053.37	50.2	**4217.73**	397.14	3.01E-011

Table 5.27: Pareto Size Indicators for PAES and NEMO$_{NSGA-II}$

5.4.8 PAES versus NEMO$_{MOPSO}$

In this comparison, MOPSO was used as the underlying EMO algorithm to train NEMO and compared against PAES. Tables 5.31-5.33 show the average Pareto optimal front size, spacing indicator, and hypervolume indicator, and Table 5.34 shows the median and interquartile ranges for the binary $\epsilon+$ indicator. Figures 5.29-5.32 display the average and standard errors of the optimal front sizes, spacing, and hypervolume indicators, and the median and standard errors of the binary $\epsilon+$ indicators.

NEMO$_{MOPSO}$ outperforms PAES in 100% of the problems in terms of the size of the Pareto optimal set. It outperforms PAES in terms of the spacing indicator on 50% of the problems, while being outperformed on only 10% of the problems. For the hypervolume

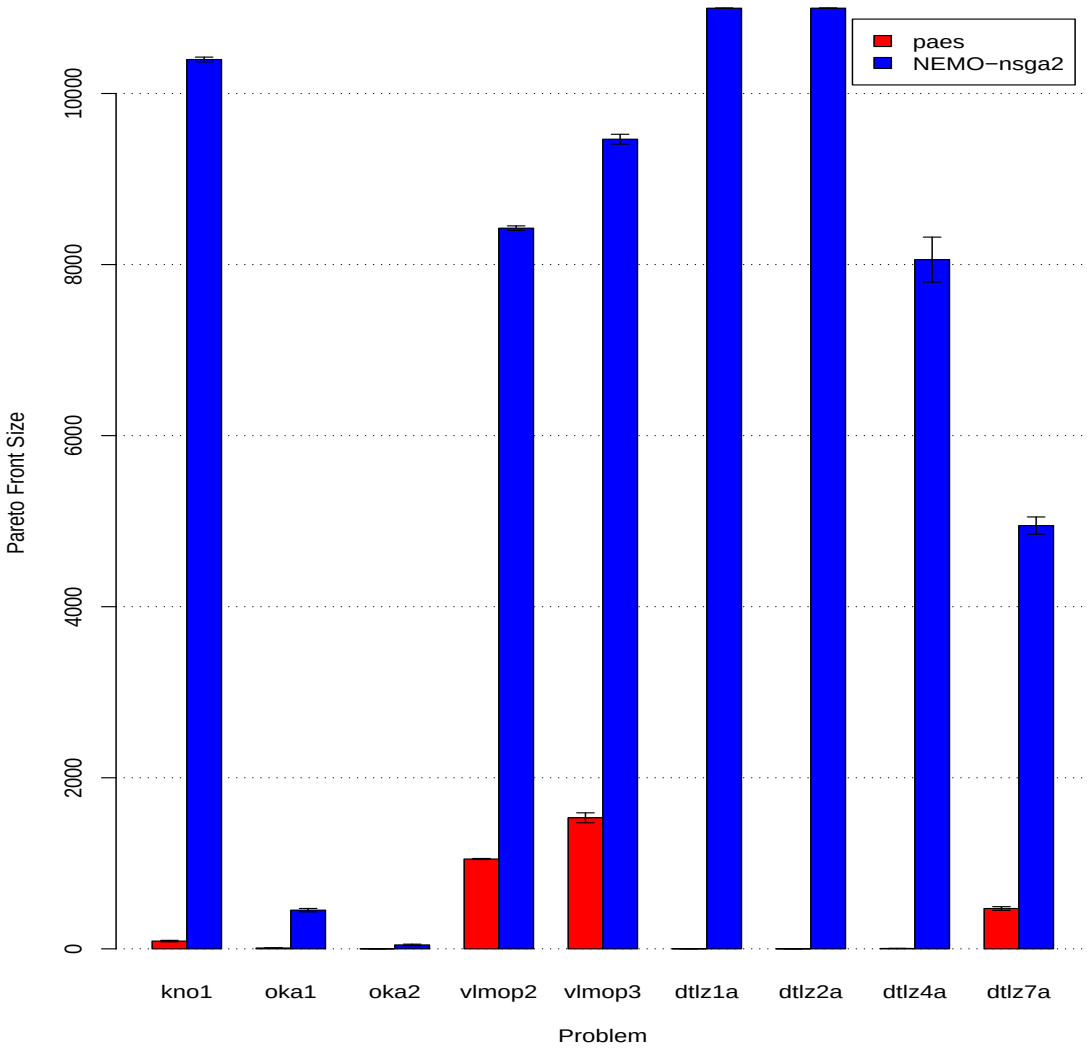


Figure 5.25: Pareto Size Indicators for PAES and NEMO$_{NSGA-II}$ with Standard Error Bars.

Problem	PAES		NEMO$_{NSGA-II}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	0.63	2.34	**0**	0	4.09E-011
oka1	0.43	0.68	0.07	0.1	0.2
oka2	**0.13**	0.68	0.5	0.51	4.52E-011
vlmop2	0	0	0	0	NaN
vlmop3	0.01	0	**0**	0	8.70E-014
dtlz1a	0	0	0	0	0.16
dtlz2a	0.05	0.14	**0.01**	0	0
dtlz4a	0.03	0.16	**0.01**	0	8.19E-010
dtlz7a	0.01	0	0.01	0	NaN
sdflp	1.82	2	2	1.09	0.41

Table 5.28: Spacing Indicators for PAES and NEMO$_{NSGA-II}$

Problem	PAES		NEMO$_{NSGA-II}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	1649.12	793.33	**369790.1**	5840.04	3.02E-011
oka1	9.94	12.95	**533.73**	204.29	8.13E-011
oka2	0.84	1.99	**74.6**	79.76	3.57E-011
vlmop2	90.68	3.8	**659.74**	8.7	3.02E-011
vlmop3	644.47	192.35	**6640.39**	579.65	3.02E-011
dtlz1a	0.01	0.01	**457.84**	63.9	1.42E-011
dtlz2a	0.04	0.07	**629.47**	175.4	1.78E-011
dtlz4a	0.08	0.18	**535.18**	225.24	2.76E-011
dtlz7a	114.52	33.2	**783.95**	130.35	3.02E-011
sdflp	99226666.67	5360193.85	**534733333.33**	64608600.54	3.00E-011

Table 5.29: Hypervolume Indicators for PAES and NEMO$_{NSGA-II}$

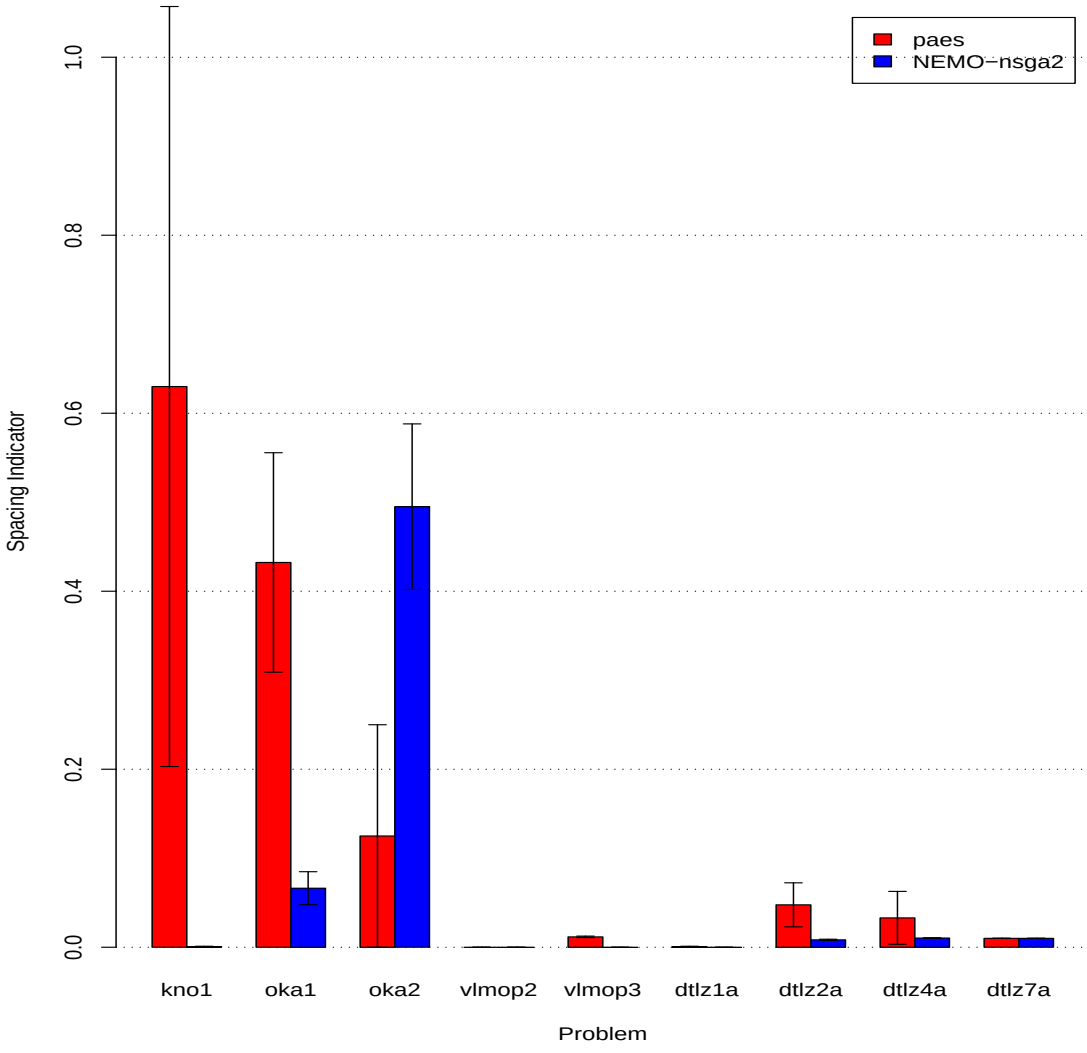


Figure 5.26: Spacing Indicators for PAES and NEMO$_{NSGA-II}$ with Standard Error Bars.

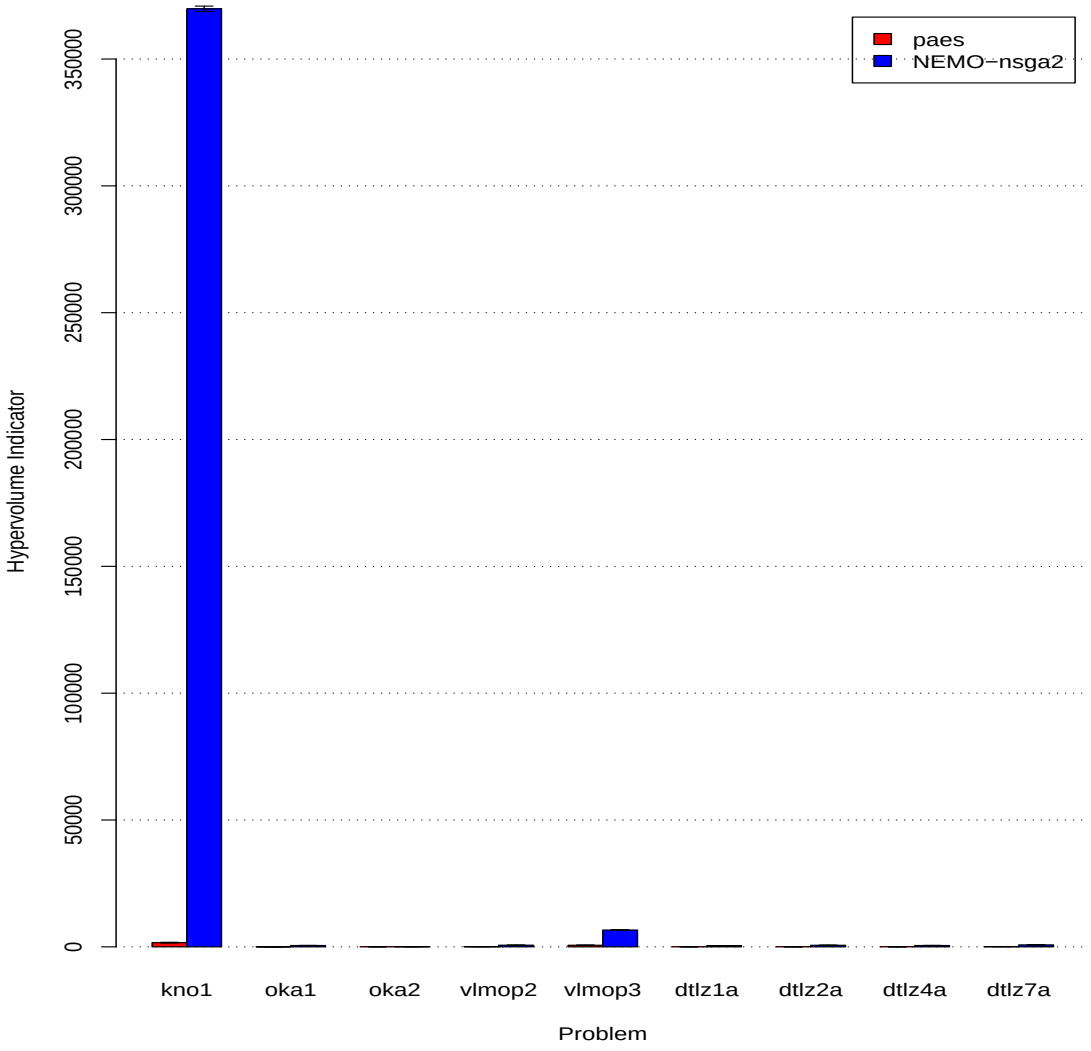


Figure 5.27: Hypervolume Indicators for PAES and NEMO$_{NSGA-II}$ with Standard Error Bars.

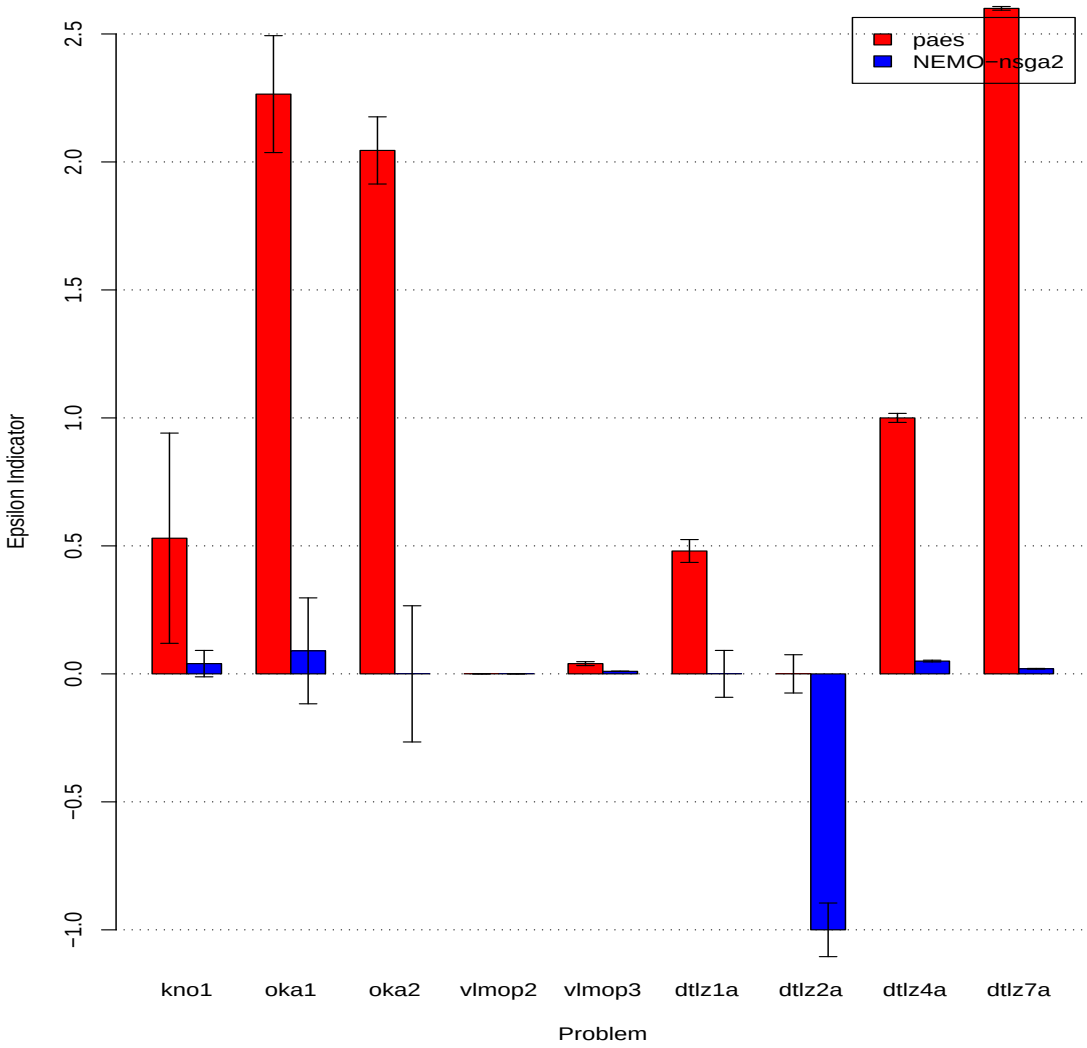


Figure 5.28: $\epsilon+$ Indicators for PAES and NEMO$_{NSGA-II}$ with Standard Error Bars.

Problem	PAES		NEMO$_{NSGA-II}$		p-value
	Median	IQR	Median	IQR	
kno1	0.53	0.21	**0.04**	0.04	9.24E-009
oka1	2.27	1.95	**0.09**	2.19	0
oka2	2.05	1.46	**2.65E-006**	0.11	0
vlmop2	0	0	0	0	NaN
vlmop3	0.04	0.03	**0.01**	0	4.95E-010
dtlz1a	0.48	0.49	**0**	1.01	1.95E-005
dtlz2a	0	0.77	**-1**	1.01	0
dtlz4a	1	0	**0.05**	0.02	1.52E-012
dtlz7a	2.6	0.05	**0.02**	0.01	1.28E-011
sdflp	170.06	0.1	**2.42**	8.46	1.15E-007

Table 5.30: $\epsilon+$ Indicators for PAES and NEMO$_{NSGA-II}$

indicator, NEMO$_{NSGA-II}$ outperforms PAES in 90% of the problems in the test suite. Finally, on the binary $\epsilon+$ indicator, NEMO$_{NSGA-II}$ outperforms PAES on 90% of the problems, and, on OKA2 and DTLZ1a, its performance is conclusive.

5.4.9 PAES versus NEMO$_{PAES}$

In this comparison, PAES was used as the underlying EMO algorithm to train NEMO and compared against standard PAES. Tables 5.35-5.37 show the average Pareto optimal front size, spacing indicator, and hypervolume indicator, whereas Table 5.38 shows the median and interquartile ranges for the binary $\epsilon+$ indicator. Figures 5.33-5.36 display the average and standard errors of the optimal front sizes, spacing, and hypervolume indicators, and the median and standard errors of the binary $\epsilon+$ indicators.

NEMO$_{PAES}$ outperforms PAES in 90% of the problems in terms of the size of the Pareto optimal set. It outperforms PAES in terms of the spacing indicator on 50% of the

Problem	PAES		NEMO$_{MOPSO}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	31.8	71.6	**10060.5**	2916.42	4.75E-010
oka1	15.97	15.06	**29.3**	21.37	0
oka2	0.17	0.46	**28.23**	57.68	9.55E-011
vlmop2	59.67	29.18	**10939.6**	29.17	2.96E-011
vlmop3	647.63	790.02	**10352.07**	789.89	3.02E-011
dtlz1a	19.5	45.23	**10939.93**	187.54	8.25E-012
dtlz2a	8.77	11.63	**10990.23**	11.63	2.78E-011
dtlz4a	4.37	3.12	**1317.3**	2005.47	0
dtlz7a	595.9	167.76	**3916.77**	3093.48	9.48E-006
sdflp	1233.97	46.2	**2597.03**	418.57	3.01E-011

Table 5.31: Pareto Size Indicators for PAES and NEMO$_{MOPSO}$

Problem	PAES		NEMO$_{MOPSO}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	1.1	1.07	**0.01**	0.03	6.94E-007
oka1	0.31	0.38	0.22	0.25	0.81
oka2	**0**	0	0.3	0.5	5.35E-006
vlmop2	0.02	0.01	**0**	0	9.20E-013
vlmop3	0.04	0.03	**0**	0	1.10E-012
dtlz1a	0	0	6.20E-008	3.40E-007	0.54
dtlz2a	0.09	0.1	**0**	0	0
dtlz4a	0.02	0.05	0.01	0.01	0.78
dtlz7a	0.01	0	0.01	0.01	5.77E-005
sdflp	3.29	2.16	**1.62**	1.42	0.01

Table 5.32: Spacing Indicators for PAES and NEMO$_{MOPSO}$

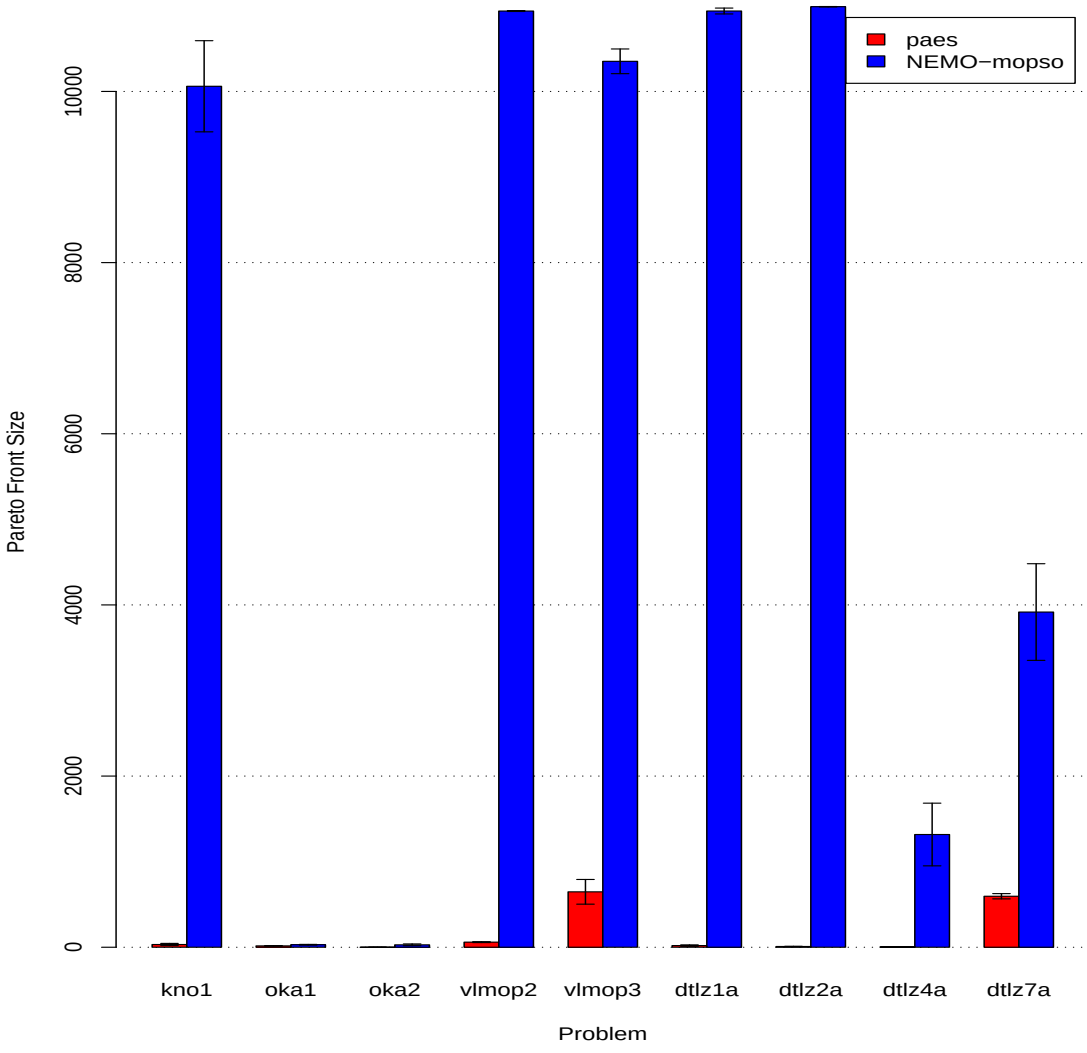


Figure 5.29: Pareto Size Indicators for PAES and NEMO$_{MOPSO}$ with Standard Error Bars.

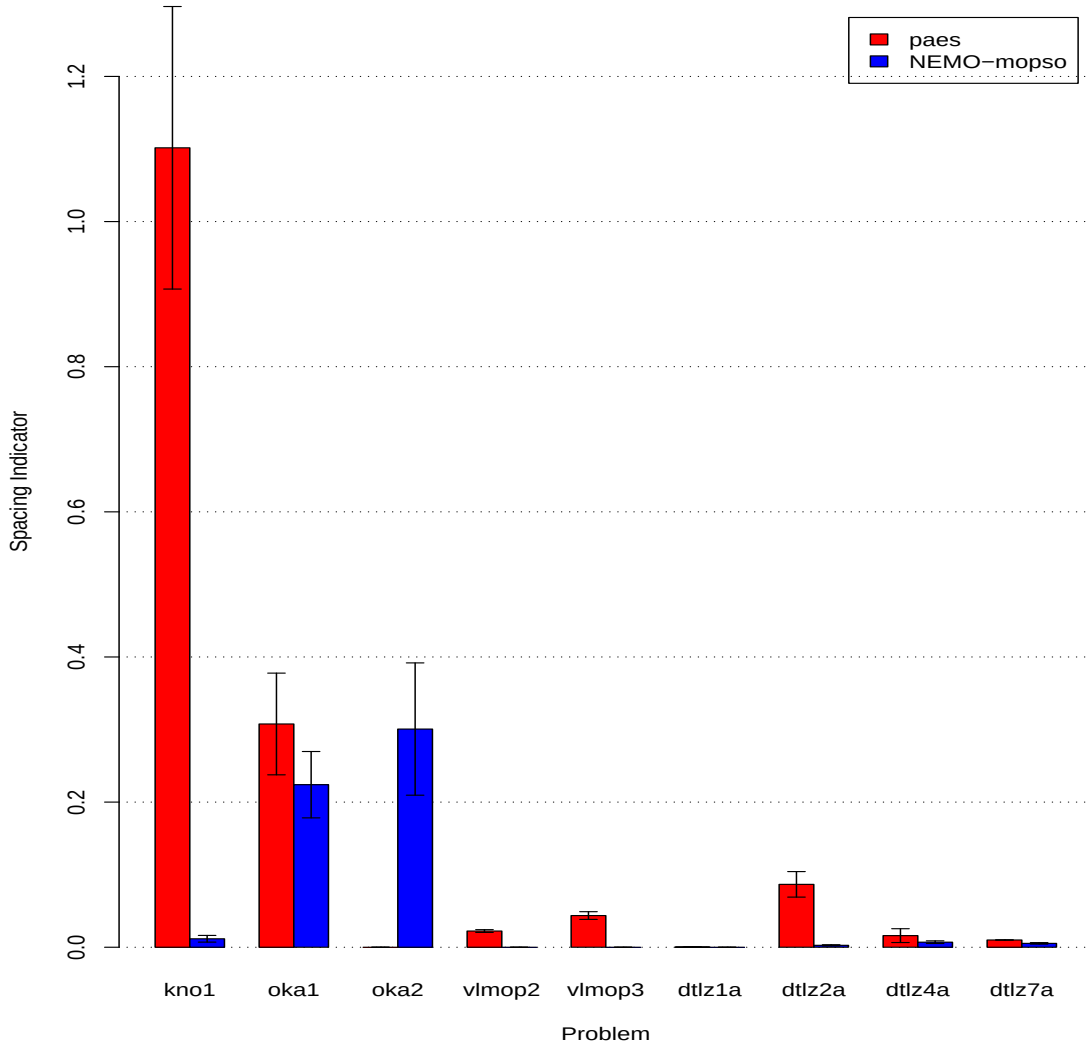


Figure 5.30: Spacing Indicators for PAES and NEMO$_{MOPSO}$ with Standard Error Bars.

Problem	PAES		NEMO$_{MOPSO}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	728.93	1951.34	**339228.54**	103995.7	5.81E-010
oka1	10.17	9.86	**15.44**	10.04	0.01
oka2	0.03	0.14	**4.81**	13.01	0
vlmop2	4.3	2.01	**876.69**	10.17	3.02E-011
vlmop3	402.55	343.97	**5944.66**	1253.2	3.02E-011
dtlz1a	0.15	0.35	**384.34**	93.13	1.10E-011
dtlz2a	0.37	0.58	**332.32**	217.45	2.90E-011
dtlz4a	0.04	0.05	**84.31**	152.2	0.01
dtlz7a	59.91	60.68	53.13	82.93	0.2
sdflp	122066666.67	5570447.47	**314700000**	56847739.77	2.97E-011

Table 5.33: Hypervolume Indicators for PAES and NEMO$_{MOPSO}$

Problem	PAES		NEMO$_{MOPSO}$		p-value
	Median	IQR	Median	IQR	
kno1	2.14	1.85	**0.04**	0.17	2.52E-006
oka1	2.29	2.02	**0.35**	2.08	0.01
oka2	0	0	**-1**	0	4.53E-008
vlmop2	0.05	0.02	**0**	0	9.62E-013
vlmop3	0.03	0.03	**0.01**	0.01	1.59E-009
dtlz1a	0	0.48	**-1**	1	1.69E-005
dtlz2a	0.58	0.34	**0.29**	0.37	0
dtlz4a	0.83	1.73	**0.09**	0.15	0.01
dtlz7a	0.87	0.94	**0.69**	0.59	0.05
sdflp	**3.33**	168.16	170.02	160.55	0

Table 5.34: $\epsilon+$ Indicators for PAES and NEMO$_{MOPSO}$

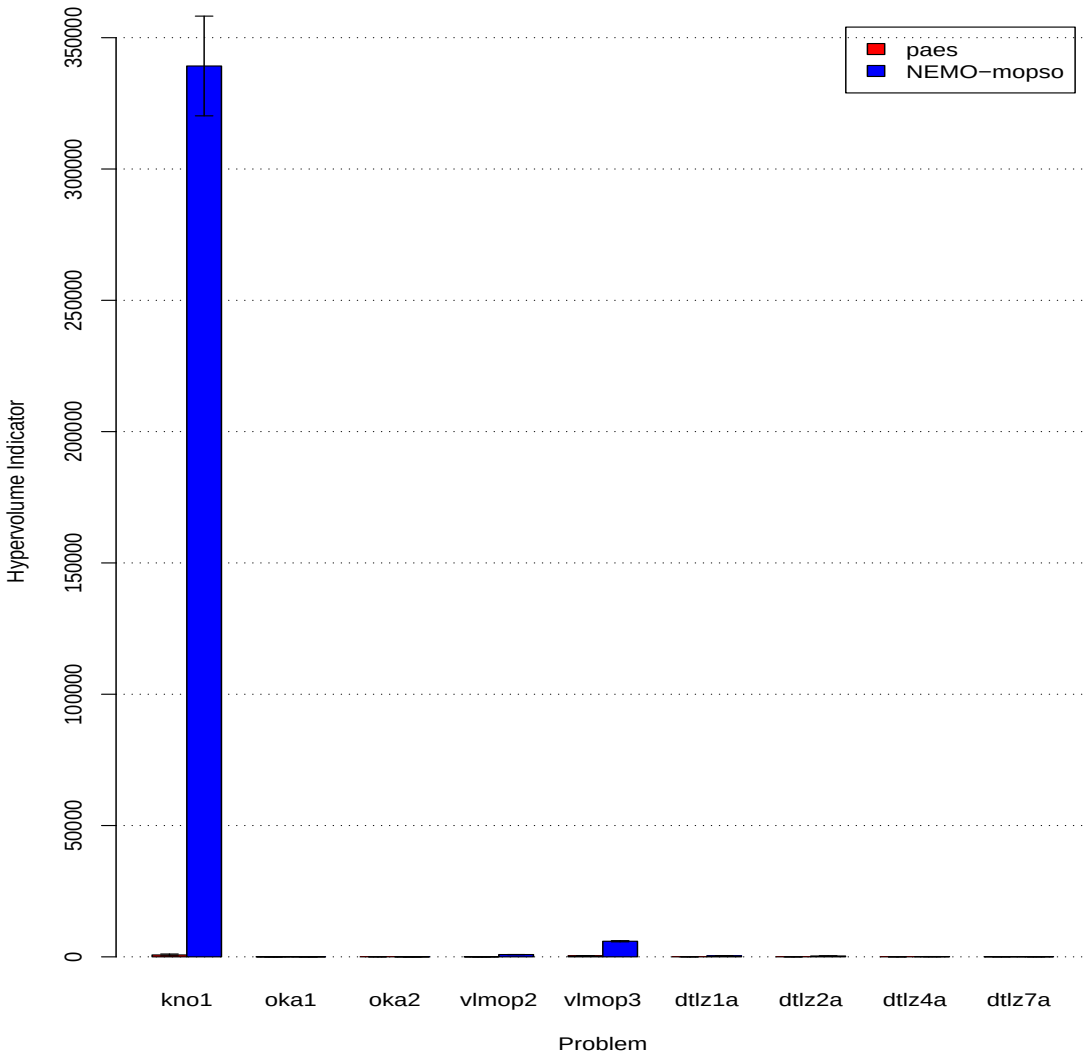


Figure 5.31: Hypervolume Indicators for PAES and NEMO$_{MOPSO}$ with Standard Error Bars.

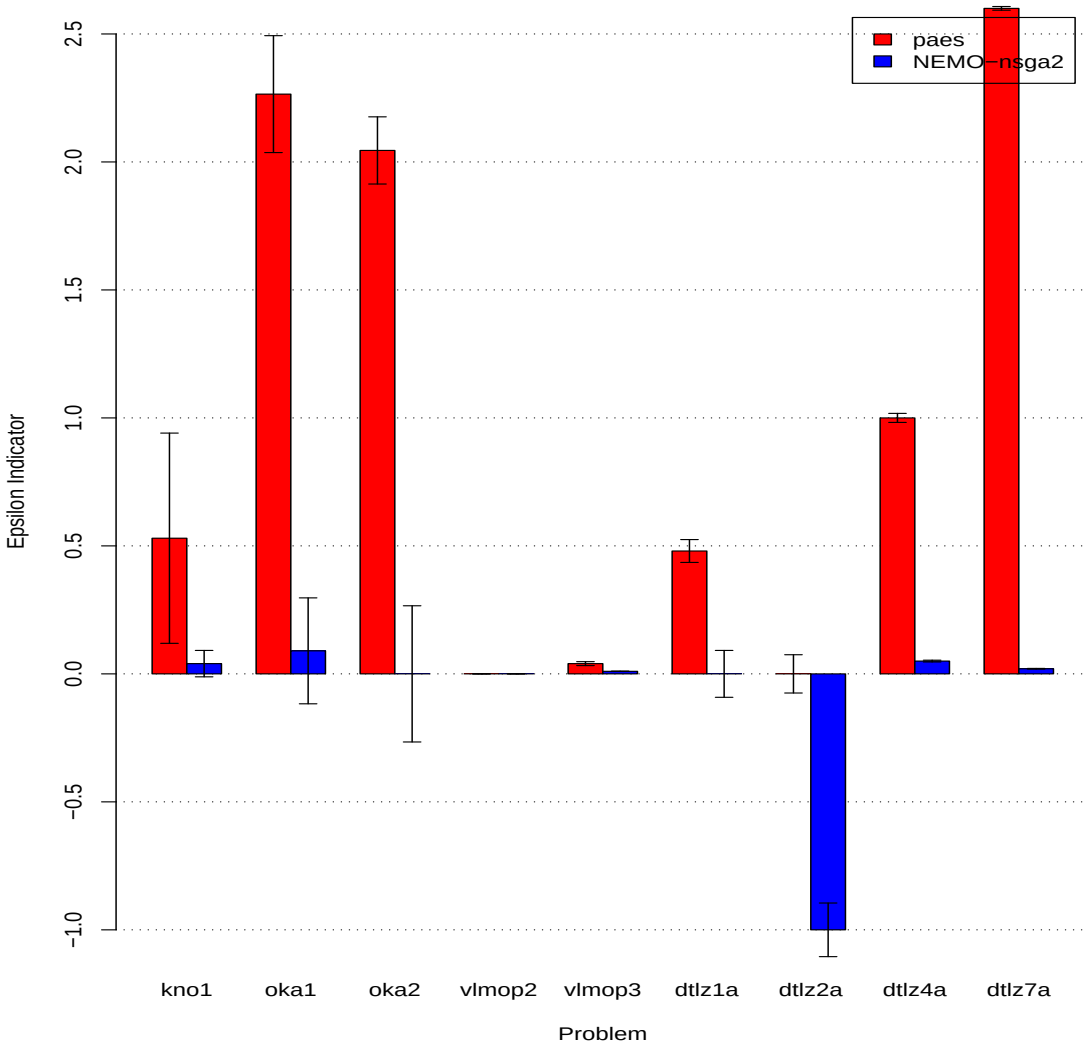


Figure 5.32: $\epsilon+$ Indicators for PAES and NEMO$_{MOPSO}$ with Standard Error Bars.

problems, while being outperformed on only 20% of the problems. For the hypervolume indicator, NEMO$_{PAES}$ outperforms PAES in 70% of the problems in the test suite. Finally, on the binary $\epsilon+$ indicator, NEMO$_{PAES}$ outperforms PAES on 80% of the problems, and, on DTLZ4a, its performance is conclusive.

Problem	PAES		NEMO$_{PAES}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	173.43	109.92	**9488.5**	2208.73	3.01E-011
oka1	748.6	1239.38	2255.7	3977.04	0.12
oka2	13.2	18.65	**375.77**	1802.26	0.05
vlmop2	1121.9	39.89	**8487.3**	104.74	3.00E-011
vlmop3	883.03	1029.59	**10964.3**	142.8	2.50E-011
dtlz1a	2.6	1.28	**1830.17**	1215.96	2.07E-011
dtlz2a	17.4	21.19	**10998.6**	0.86	2.08E-011
dtlz4a	1.03	2.55	**68.77**	47.63	4.57E-011
dtlz7a	515.2	85.42	**5627.33**	1578.23	3.02E-011
sdflp	1325.2	49.04	**5014.7**	391.87	3.01E-011

Table 5.35: Pareto Size Indicators for PAES and NEMO$_{PAES}$

5.5 Conclusions

The results from these experiments reveal several important facts. First, the preliminary results from the previous chapters are shown to hold up to statistical scrutiny. The three experiments – NSGA-II versus NEMO$_{NSGA-II}$, MOPSO versus NEMO$_{MOPSO}$, and PAES versus NEMO$_{PAES}$ – reveal that the NEMO approach is very effective when used on a given EMO algorithm to increase the size of the Pareto optimal set. The NEMO

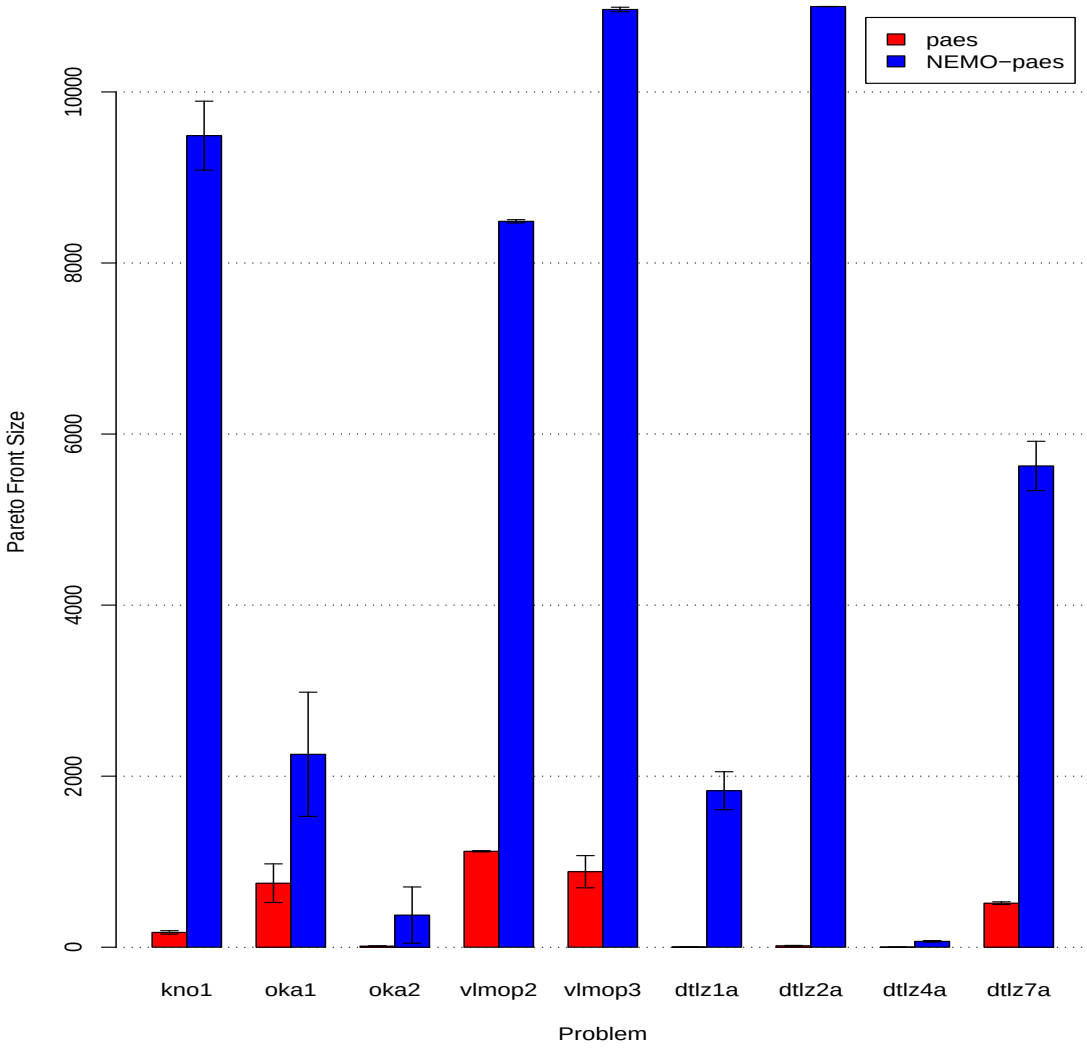


Figure 5.33: Pareto Size Indicators for PAES and NEMO$_{PAES}$ with Standard Error Bars.

Problem	PAES		NEMO$_{PAES}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	0.12	0.06	**0.01**	0.01	1.35E-011
oka1	0.21	0.31	0.31	0.3	0.27
oka2	0.18	0.56	0.05	0.07	0.62
vlmop2	0	0	0	0	NaN
vlmop3	0.02	0.01	**0**	0	2.05E-012
dtlz1a	**0**	0.01	0.01	0.01	0.03
dtlz2a	0.08	0.08	**0**	0	9.24E-007
dtlz4a	**0.04**	0.17	0.1	0.09	2.35E-005
dtlz7a	0.01	0	**0**	0	2.51E-011
sdflp	3.87	1.7	**2.43**	0.1	6.72E-005

Table 5.36: Spacing Indicators for PAES and NEMO$_{PAES}$

Problem	PAES		NEMO$_{PAES}$		p-value
	Mean	St. Dev.	Mean	St. Dev.	
kno1	3220.31	1983.64	**334196.11**	86365.77	3.02E-011
oka1	3384.32	6026.43	9868.94	18793.45	0.3
oka2	6.18	15.38	462.68	2427.21	0.49
vlmop2	98.91	4.09	**647.98**	8.77	3.02E-011
vlmop3	530.87	471.19	**6016.88**	283.83	3.02E-011
dtlz1a	17.03	14.25	**31409.09**	43608.72	3.01E-011
dtlz2a	0.27	0.48	**262.03**	73.22	2.95E-011
dtlz4a	2.12E-008	1.15E-007	3.30E-007	1.81E-006	0.99
dtlz7a	11.47	7.22	**129.02**	104.45	3.02E-011
sdflp	136300000	5298991.45	**594100000**	55494237.04	2.96E-011

Table 5.37: Hypervolume Indicators for PAES and NEMO$_{PAES}$

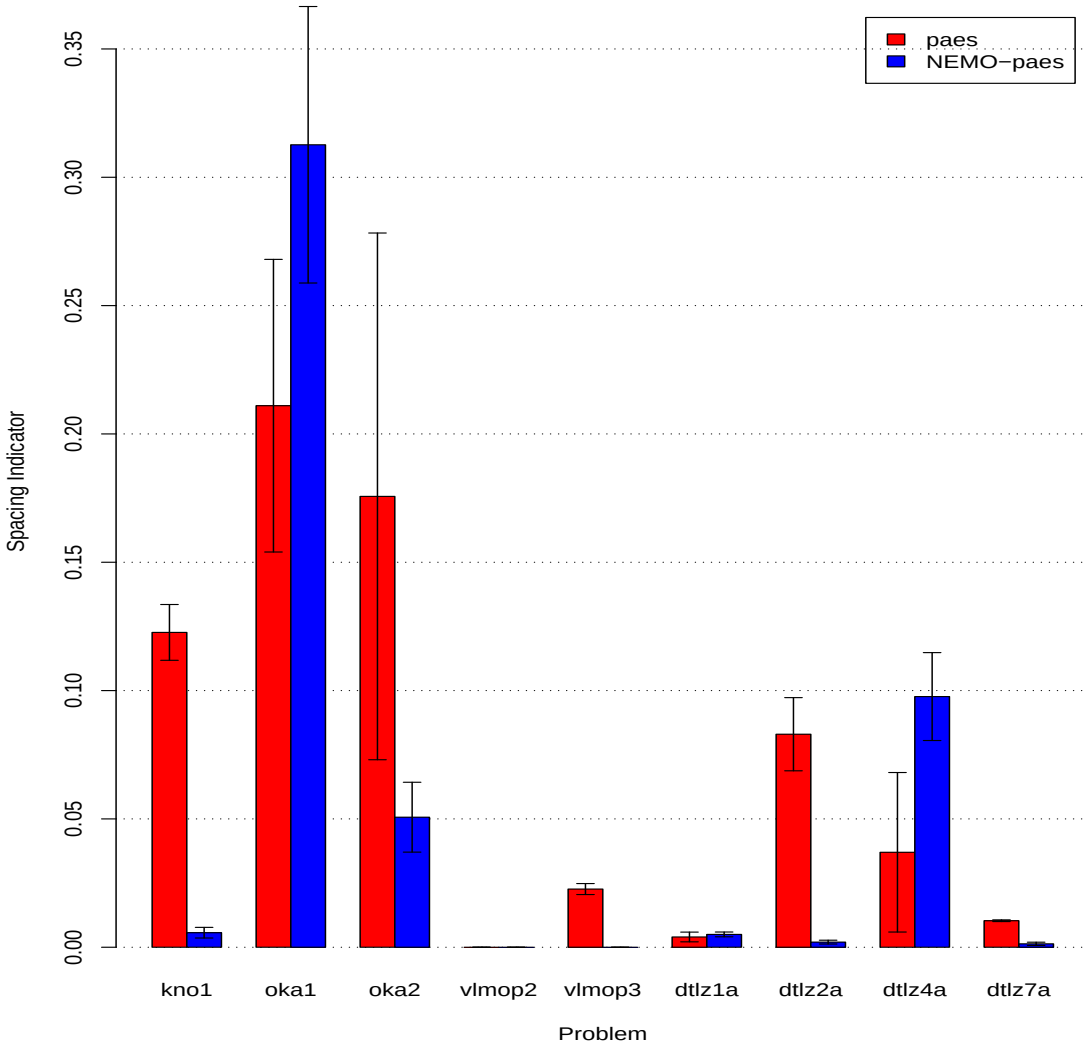


Figure 5.34: Spacing Indicators for PAES and NEMO$_{PAES}$ with Standard Error Bars.

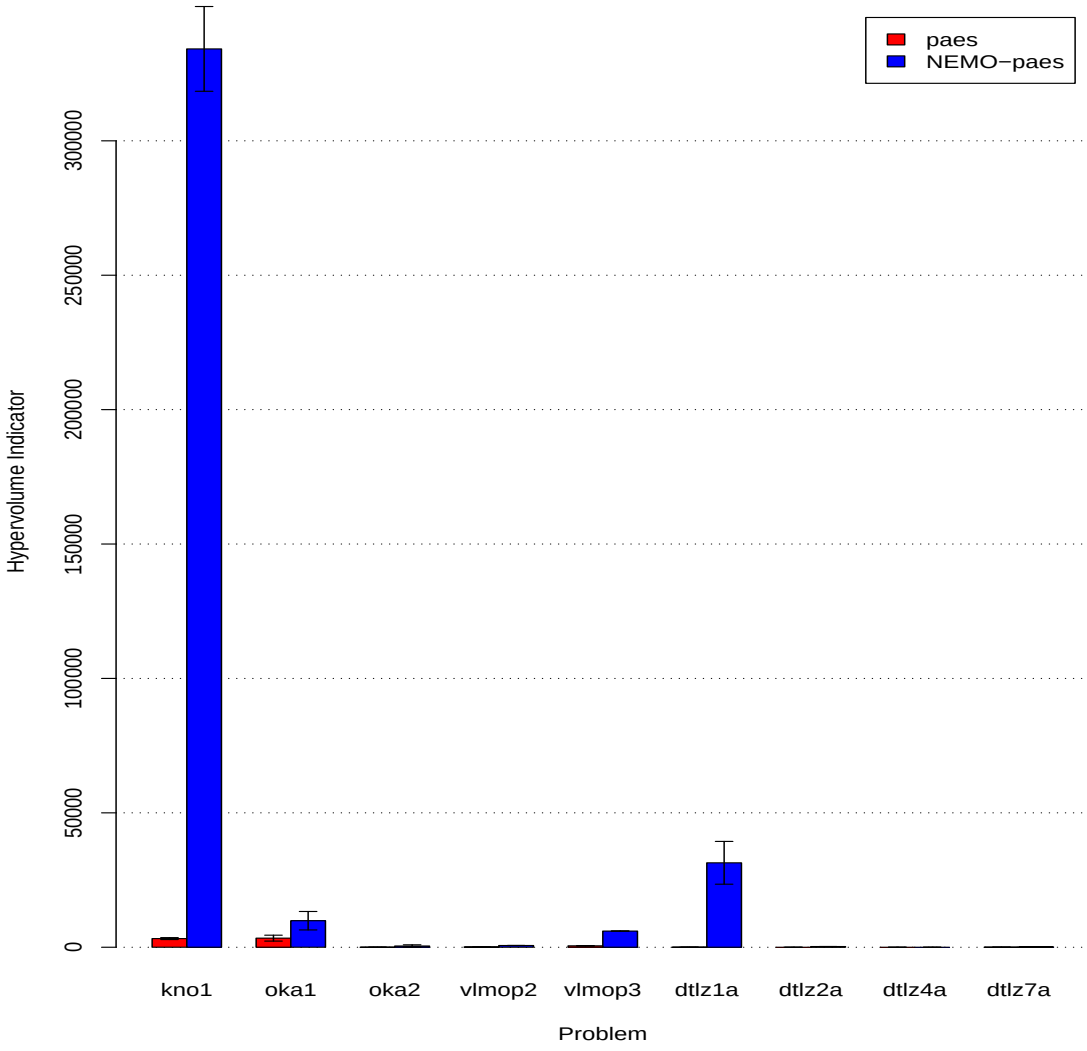


Figure 5.35: Hypervolume Indicators for PAES and NEMO$_{PAES}$ with Standard Error Bars.

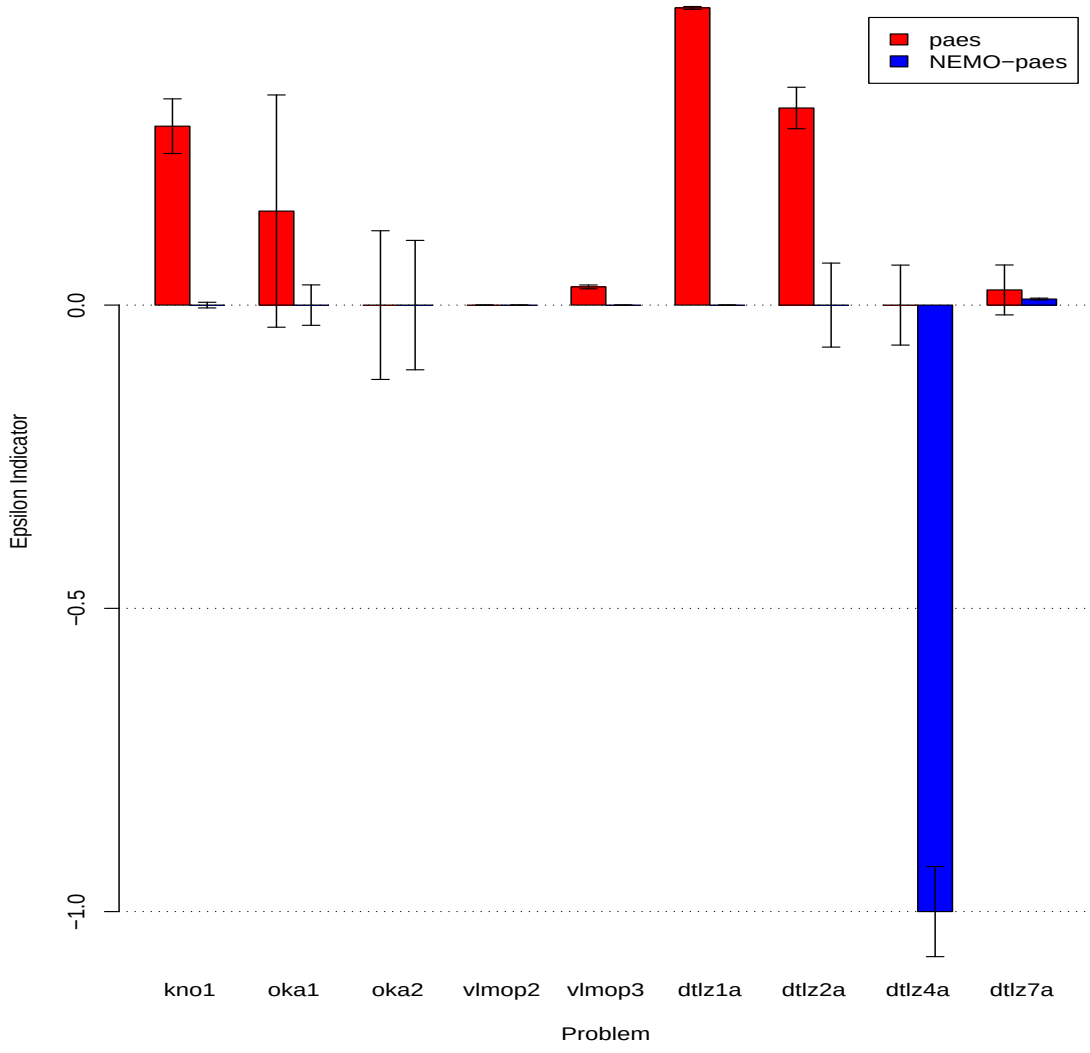


Figure 5.36: $\epsilon+$ Indicators for PAES and NEMO$_{PAES}$ with Standard Error Bars.

Problem	PAES		NEMO$_{PAES}$		p-value
	Median	IQR	Median	IQR	
kno1	0.3	0.12	**0**	0.01	1.50E-011
oka1	0.16	2.24	**0**	0	7.16E-010
oka2	0	0.13	0	0.75	0
vlmop2	0	0	0	0	NaN
vlmop3	0.03	0.02	**0**	0	1.46E-012
dtlz1a	0.49	0.02	**0**	0	8.74E-013
dtlz2a	0.33	0.22	**0**	0	5.99E-009
dtlz4a	0	0	**-1**	0	5.76E-010
dtlz7a	0.03	0.11	**0.01**	0.01	1.06E-009
sdflp	3.8	8	**0**	0	1.21E-012

Table 5.38: $\epsilon+$ Indicators for PAES and NEMO$_{PAES}$

approach in these instances also appears to do well on all indicators except for the binary $\epsilon+$ indicator, which was, however, rarely conclusive.

Second, the NSGA-II versus NEMO$_{MOPSO}$ and NSGA-II versus NEMO$_{PAES}$ experiments reveal that the NEMO approach is capable of taking a less successful EMO algorithm and making it competitive with NSGA-II. This is especially true of PAES, which has been shown to be outperformed by NSGA-II in head-to-head comparisons [6]. In this work, however, NEMO$_{PAES}$ is shown to be very competitive with NSGA-II in the sizes of the Pareto optimal sets, the spacing indicator, and the hypervolume indicator. This in itself is a remarkable statement about the power of the NEMO approach.

Finally, these experiments reveal that the NEMO approach has difficulty yielding competitive values for the $\epsilon+$ indicator. In many of the experiments, the NEMO algorithm performed well on all indicators except for the $\epsilon+$ indicator. Further investigation is warranted to determine whether this is a limitation of the NEMO approach or whether the

$\epsilon+$ indicator is simply not salient enough to be used as a basis for comparison without additional indicators.

Chapter 6

Conclusions and Future Work

The neural enhancement technique described in this work was developed in an effort to learn the active areas of decision space where Pareto optimal solutions could be found. As was discussed in the introduction and literature review, this technique is applied directly to multiobjective optimization algorithms and relies heavily on neural network concepts. While several researchers have made use of artificial intelligence or pattern recognition approaches to assist with multiobjective optimization, the exhaustive literature survey conducted by the author revealed no studies (except for [65], for which this work is an extension) that attempt to solve the problem of learning promising areas of decision space.

In light of this absence of research, the first effort put forth in the current work was to justify the use of the neural network technique by proving its effectiveness. It was shown that the NEMO approach could produce many times more solutions than a very successful existing EMO approach (NSGA-II). While this was a promising result, the experiment left many unanswered questions. First and foremost, would it be possible to train the NEMO in an acceptable period of time so as to make the approach computationally competitive with the existing EMO approach? Second, would the NEMO approach perform well when evaluated based on other indicators that take into account the shape and spread of the Pareto optimal frontier, rather than just its size?

A final concern with the results of that first experiment involved the manner in which the NEMO and EMO approaches were compared. In that experiment, the EMO approach was given 25600 function evaluations, from which a training set was extracted. Then, the

NEMO was trained on that set and was given an additional 25600 function evaluations to expand its Pareto optimal frontier. Finally, its results were compared to those produced by the EMO (essentially, the training set). This seemed extremely "unfair" to the EMO since the NEMO was given a "head start."

To address all of these concerns, a second experiment was carried out. In this experiment, five different training algorithms for NEMO were evaluated and compared using multiple indicators from the multiobjective optimization literature. Additionally, the training time was collected for each training approach. To alleviate the "unfair" comparison mentioned above, a new methodology was enacted. In this experiment, the EMO was given 25600 function evaluations in order to produce a training set, but it was then given an additional 25600 function evaluations in which to optimize that set even further. Likewise, the NEMO was given 25600 function evaluations to improve and expand upon the training set. The results from each after 51200 total function evaluations were then compared.

This second experiment generated extremely positive results. First, it led to the creation of heuristic training methods for the NEMO approach that produced training times that were well in the range of the existing EMO approaches. Additionally, these heuristic methods also generated NEMOs that generally outperformed the EMO in terms of relative yield, spacing, and hypervolume. In terms of the binary $\epsilon+$ indicator, the EMO approach performed better, but often this better performance was not conclusive due to the nature and limits of the $\epsilon+$ indicator.

There were several relevant open issues generated from this experiment. First, while the second experiment used 5 runs in an effort to generate statistical measures of success,

it seemed necessary to repeat the EMO/NEMO comparison many times in order to determine statistical significance. Additionally, it remained to be seen whether the NEMO with heuristic training approach could be used with other EMO algorithms with equal success. Consequently, it was important to determine whether relatively "poor" EMO algorithms could be bolstered by NEMO to make them competitive with "good" EMO algorithms.

Therefore, a final experiment was carried out that made use of the heuristically-trained NEMO using training sets generated from NSGA-II (as before), MOPSO, and PAES. The nine different pairwise comparisons were made ($\text{NEMO}_{NSGA-II}$/NSGA-II, $\text{NEMO}_{NSGA-II}$/MOPSO, etc.), each using results collected across 30 different runs. When analyzing the results in this experiment, several interesting observations were made. First, the results from the previous experiments (essentially the $\text{NEMO}_{NSGA-II}$/NSGA-II comparison) were verified under statistical scrutiny. Second, the MOPSO and PAES EMO algorithms were both enhanced by the use of the NEMO approach. (This observation applies to the NEMO_{MOPSO}/MOPSO and NEMO_{PAES}/PAES comparisons.)

Most importantly, however, this final experiment revealed that the NEMO approach could make less powerful EMO approaches competitive with powerful EMO algorithms. In particular, it has been shown in [6] that NSGA-II outperforms PAES. However, we show that NEMO_{PAES} performs very well when compared to NSGA-II, making it certainly competitive if not dominant. This is an exciting result that underscores the power of the NEMO method.

It remains to be seen whether a different learning system might outperform the general regression neural network when used as the NEMO learner. It seems unlikely that a backpropagation feed-forward neural network would be successful, given the prohibitive

computation time needed for training. However, it's possible that an optimized radial basis function network might provide a legitimate alternative. In such a case, it would be interesting to discover whether the particular strengths of the RBF network would provide gains on certain types of multiobjective optimization problems. However, there may be additional learning approaches that are equally or more effective. A thorough investigation of this area is warranted.

Another very important direction for future work lies in finding the ideal point at which to engage the NEMO approach. It should be clear that if the EMO is halted prematurely, the NEMO will be trained on a poor representative Pareto optimal frontier. If given additional function evaluations, the EMO will be much more likely to find solutions that dominate those found by the NEMO. This is because NEMO operates "horizontally," expanding the Pareto optimal frontier, whereas the EMO operates "vertically," attempting to find solutions that are more optimal than the existing set. However, if the EMO is halted near the actual Pareto optimal frontier, the NEMO can be engaged immediately to generate a comprehensive sampling of the frontier. It is important to determine the point at which the switch from EMO to NEMO should occur, including any metrics or indicators that might provide some insight in this area.

If such a set of indicators could be found, it may then be possible to incorporate the NEMO approach into an existing EMO algorithm so that it could expand solutions on-line in order to give the EMO a better chance to find the Pareto optimal frontier. However, without such indicators, the NEMO approach would almost certainly be a liability, "wasting" function evaluations to fill out a sub-optimal Pareto frontier. It may, instead, be

possible to use a NEMO approach periodically to generate promising areas of the search space for the EMO to explore.

The results presented in this work, plus these future directions, reveal the wealth of opportunity that the NEMO approach provides for multiobjective optimization. With the multi-tasking, multi-cultural, multi-billion members of this 21st century society, there is no doubt that single objective optimization will be unable to solve the complex problems that globalization has created. Multiobjective optimization arises at the dawn of this era as the answer to these difficult challenges, but it has challenges of its own to overcome. It is in answer to some of those challenges that NEMO is offered, in the hopes of pushing the limits of understanding a little further and making the world a little better, which is a multiobjective problem in itself.

Bibliography

[1] C. A. C. Coello, "A short tutorial on evolutionary multiobjective optimization," in Proceedings of First International Conference on Evolutionary Multi-Criterion Optimization, E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, and D. Corne, Eds. Springer-Verlag, 2001, pp. 21–40.

[2] E. Zitzler, M. Laumanns, and S. Bleuler, "A tutorial on evolutionary multiobjective optimization," in Workshop on Multiple Objective Metaheuristics (MOMH 2002). Berlin, Germany: Springer-Verlag, 2002.

[3] H. Eschenauer, J. Koski, and A. Osyczka, Multicriteria Design Optimization. Berlin: Springer-Verlag, 1990.

[4] C. A. C. Coello and M. Lechunga, "MOPSO: A proposal for multiple objective particle swarm optimization," in Proceedings of IEEE World Congress on Computational Intelligence, 2002, pp. 1051–1056.

[5] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," Evolutionary Computation, vol. 2, no. 3, pp. 221–248, 1994.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, apr 2002.

[7] J. Knowles, "ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems," IEEE Transactions on Evolutionary Computation, vol. 10, no. 1, pp. 50–66, feb 2005.

[8] A. Garrett, G. Dozier, and K. Deb, "NEMO: neural enhancement for multiobjective optimization," in Proceedings of the 2007 Congress on Evolutionary Computation. IEEE Press, sep 2007, pp. 3108–3113.

[9] I. Das and J. E. Dennis, "A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems," Structural and Multidisciplinary Optimization, vol. 14, no. 1, pp. 63–69, aug 1997.

[10] C. A. C. Coello, "20 years of evolutionary multi-objective optimization: What has been done and what remains to be done," in Computational Intelligence: Principles and Practice, G. Y. Yen and D. B. Fogel, Eds. IEEE Computational Intelligence Society, 2006, pp. 73–88.

[11] T. Bäck, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: Comments on the history and current state," IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 3–17, apr 1997.

[12] Z. Michalewicz and D. B. Fogel, How to Solve It: Modern Heuristics. Springer, 2004.

[13] D. B. Fogel, "What is evolutionary computation?" IEEE Spectrum, vol. 37, no. 2, pp. 26–32, Feb. 2000.

[14] J. H. Holland, Adaptation in Natural and Artificial Systems. Ann Arbor, MI: University of Michigan Press, 1975.

[15] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA: Addison-Wesley Publishing Company, Inc., 1989.

[16] S. Forrest, "Genetic algorithms: principles of natural selection applied to computation," Science, vol. 60, pp. 872–878, Aug. 1993.

[17] M. D. Vose, The Simple Genetic Algorithm: Foundations and Theory. MIT Press, 1999.

[18] L. J. Fogel, A. J. Owens, and M. J. Walsh, Artificial intelligence through simulated evolution. New York: Wiley, 1966.

[19] D. B. Fogel, "An introduction to simulated evolutionary optimization," IEEE Transactions on Neural Networks, vol. 5, no. 1, pp. 3–14, Jan. 1994.

[20] T. Bäck, F. Hoffmeister, and H.-P. Schwefel, "A survey of evolution strategies," in Proceedings of the 4th International Conference on Genetic Algorithms, R. K. Belew and L. B. Booker, Eds. Morgan Kaufman, 1991, pp. 2–9.

[21] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Proceedings of the IEEE Conference on Neural Networks, Perth, Australia, 1995, pp. 1942–1948.

[22] J. Kennedy, "The particle swarm: Social adaptation of knowledge," in Proceedings of the International Conference on Evolutionary Computation, Indianapolis, IN, 1997, pp. 303–308.

[23] M. Clerc, "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization," in Proceedings of the International Conference on Evolutionary Computation, Washington, DC, 1999, pp. 1951–1957.

[24] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in Proceedings of the Congress on Evolutionary Computation, Washington, DC, 2000, pp. 84–88.

[25] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in Proceedings of the 1st International Conference on Genetic Algorithms, L. Erlbaum, Ed., 1985, pp. 93–100.

[26] C. A. C. Coello, "Recent trends in evolutionary multiobjective optimization," in Evolutionary Multiobjective Optimization: Theoretical Advances And Applications, A. Abraham, L. Jain, and R. Goldberg, Eds. Springer-Verlag, 2005, pp. 7–32.

[27] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," IEEE Computer, vol. 29, no. 3, pp. 31–44, mar 1996.

[28] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," Cornell Aeronautical Laboratory, Psychological Review, vol. 65, no. 6, pp. 386–408, 1958.

[29] M. L. Minsky and S. Papert, Perceptrons. MIT Press, 1969.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in Parallel Data Processing, D. Rumelhart and J. McClelland, Eds. MIT Press, 1986, pp. 318–362.

[31] T. Poggio and F. Girosi, "Networks for approximation and learning," Proceedings of the IEEE, vol. 78, no. 9, pp. 1484–1487, 1990.

[32] D. F. Specht, "A general regression neural network," IEEE Transactions on Neural Networks, vol. 2, no. 6, pp. 568–576, 1991.

[33] T. Kohonen, Self-Organizing Maps. Berlin, Heidelberg: Springer, 1995.

[34] G. Carpenter and S. Grossberg, "The ART of adaptive pattern recognition by a self-organizing neural network," IEEE Computer, vol. 21, no. 3, pp. 77–88, 1988.

[35] I. A. Basheer and M. Hajmeer, "Artificial neural networks: fundamentals, computing, design, and application," Journal of Microbiological Methods, vol. 43, pp. 3–31, 2000.

[36] C. A. C. Coello, "An updated survey of GA-based multiobjective optimization techniques," ACM Computing Surveys, vol. 32, no. 2, pp. 109–143, jun 2000.

[37] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization," in Proceedings of the 5th International Conference on Genetic Algorithms, S. Forrest, Ed., 1993, pp. 416–423.

[38] ——, "An overview of evolutionary algorithms in multiobjective optimization," Evolutionary Computation, vol. 3, no. 1, pp. 1–16, 1995.

[39] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard, "Some guidelines for genetic algorithms with penalty functions," in Proceedings of the 3rd International Conference on Genetic Algorithms, J. D. Schaffer, Ed. Morgan Kaufmann, 1989, pp. 191–197.

[40] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in Proceedings of the Second International Conference on Genetic Algorithms, 1987, pp. 41–49.

[41] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in Proceedings of the First IEEE Conference on Evolutionary Computation, 1994, pp. 82–87.

[42] J. Knowles and D. Corne, "The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation," in Proceedings of 1999 Congress on Evolutionary Computation, 1999, pp. 98–105.

[43] F. Glover and M. Laguna, Tabu Search. Norwell, MA: Kluwer, 1997.

[44] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," Science, vol. 220, no. 4598, pp. 671–680, 1983.

[45] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach," IEEE Transactions on Evolutionary Computation, vol. 3, no. 4, pp. 257–271, nov 1999.

[46] J. N. Morse, "Reducing the size of the nondominated set: Pruning by clustering," Computer Operations Research, vol. 7, no. 1–2, 1980.

[47] P. Hajela and C.-Y. Lin, "Genetic search strategies in multicriterion optimal design," Structural Optimization, vol. 4, pp. 99–107, Aug. 1992.

[48] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," in Proceedings of EUROGEN 2002, Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, Eds., 2002, pp. 95–100.

[49] A. P. Engelbrecht, Computational Intelligence: An Introduction. John Wiley and Sons, Ltd., 2002.

[50] D. W. Corne, J. D. Knowles, and M. J. Oates, "The pareto envelope-based selection algorithm for multiobjective optimization," in Proceedings of the Parallel Problem Solving from Nature VI Conference, M. Shoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds. Springer, 2000, pp. 839–848.

[51] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," Evolutionary Computation, vol. 8, no. 2, pp. 173–195, 2000.

[52] C. A. Coello Coello and G. Toscano Pulido, "A micro-genetic algorithm for multiobjective optimization," in First International Conference on Evolutionary Multi-Criterion Optimization, E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, and D. Corne, Eds. Springer-Verlag LNCS No. 1993, 2001, pp. 126–140.

[53] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," IEEE Transactions on Evolutionary Computation, vol. 8, no. 3, pp. 256–279, jun 2004.

[54] S. Mostaghim and J. Teich, "Covering pareto-optimal fronts by subswarms in multi-objective particle swarm optimization," in Proceedings of 2004 Congress on Evolutionary Computation, 2004, pp. 1404–1411.

[55] O. Schiitze, S. Mostaghim, M. Dellnitz, and J. Teich, "Covering pareto sets by multilevel evolutionary subdivision techniques," in Proceedings of 2nd International Conference on Evolutionary Multi-Criterion Optimization, Theory ond Applications, 2003, pp. 118–132.

[56] D. Jones, M. Schonlau, and W. Welch, "Efficient global optimization of expensive black-box functions," Journal of Global Optimization, vol. 13, pp. 455–492, 1998.

[57] T. M. Mitchell, Machine Learning. McGraw Hill, 1997.

[58] D. E. Rumelhart, B. Widrow, and M. A. Lehr, "The basic ideas in neural networks," Communications of the ACM, vol. 37, no. 3, pp. 87–92, 1994.

[59] A. Bryson and Y. Ho, Applied Optimal Control. Blaisdell Publishing, 1969.

[60] J. L. McClelland and D. E. Rumelhart, Explorations in parallel distributed processing: A handbook of models, programs, and exercises. MIT Press, 1988.

[61] D. R. Hush and B. G. Horne, "Progress in supervised neural networks," IEEE Signal Processing Magazine, vol. 10, no. 1, pp. 8–39, Jan. 1993.

[62] I. Hatzakis and D. Wallace, "Dynamic multi-objective optimization with evolutionary algorithms: A forward-looking approach," in Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO '06), 2006, pp. 1201–1208.

[63] A. Gaspar-Cunha and A. Vieira, "A hybrid multi-objective evolutionary algorithm using an inverse neural network," in Proceedings of Hybrid Metaheuristics Workshop at ECAI 2004, 2004, pp. 25–30.

[64] A. Gaspar-Cunha and J. A. Covas, "RPSGAe: A multiobjective genetic algorithm with elitism: Application to polymer extrusion," Lecture Notes in Economics and Mathematical Systems, 2002.

[65] H. Yapicioglu, G. Dozier, and A. E. Smith, "Neural network enhancement of multi-objective evolutionary search," in Proceedings of the 2006 Congress on Evolutionary Computation, 2006, pp. 1909–1915.

[66] K. Deb, "NSGA-II source code in C," http://delta.cs.cinvestav.mx/~ccoello/EMOO/EMOOsoftware.html.

[67] T. Okabe, Y. Jin, M. Olhofer, and B. Sendhoff, "On test functions for evolutionary multi-objective optimization," in Proceedings of Parallel Problem Solving from Nature (PPSN VIII), 2004, pp. 792–802.

[68] D. A. V. Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithm test suites," in Proceedings of the 1999 ACM symposium on Applied computing, 1999, pp. 351–357.

[69] ——, "On measuring multiobjective evolutionary algorithm performance," in Proceedings of the 2000 Congress on Evolutionary Computation, 2000, pp. 204–211.

[70] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," IEEE Transactions on Evolutionary Computation, vol. 7, no. 2, pp. 117–132, Apr. 2003.

[71] T. Okabe, Y. Jin, and B. Sendhoff, "A critical survey of performance indices for multi-objective optimisation," in Proceedings of the 2003 Congress on Evolutionary Computation, 2003, pp. 878–885.

[72] E. Zitzler, "Evolutionary algorithms for multiobjective optimization: methods and applications," Ph.D. dissertation, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1999.

[73] R. Sarker and C. A. C. Coello, "Assessment methodologies for multiobjective evolutionary algorithms," Evolutionary Computation, pp. 177–195, feb 2002.

[74] J. Knowles, "PAES source code in C," http://dbkgroup.org/knowles/multi/.

[75] M. S. Lechuga and G. T. Pulido, "MOPSO source code in C," http://delta.cs.cinvestav.mx/~ccoello/EMOO/EMOOsoftware.html.

Appendices

Appendix A

Results from Training Algorithm Experiments

These tables display the results from the five training algorithm experiments from Chapter 4. Each of them contain the average and standard deviation across all five runs, as well as the p-values associated with the spacing, hypervolume, and binary $\epsilon+$ indicators. In each of those indicators, if $p < 0.05$, the average is boldfaced and underlined.

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
kno1	1	12.41	18.24	0.02	0	19462.2	430559	0.04	0.15
kno1	2	15.09	17.54	0.02	0	21270.8	411739	0.06	0.89
kno1	3	13.27	17.32	0.02	0	20381.3	428533	0.04	0.08
kno1	4	13.84	17.71	0.02	0	20820.9	434878	0.05	0.03
kno1	5	15.03	16.92	0.02	0	21254.9	423308	0.05	0.21
Average		13.928	17.549	0.02	**0**	20638	**425803**	0.048	0.272
St. Dev.		1.15214	0.487729	0	0	752.14	8894.51	0.0083666	0.352165
p-value				0.00397675		0.0079365		0.141238	
oka1	1	11.16	0.11	0.09	0.02	798.55	70.27	0.31	0.09
oka1	2	5.67	0.21	0.09	0.22	805.18	119.07	0.07	0.18
oka1	3	9.27	0.22	0.06	0.21	635.27	98.43	2.26	0.19
oka1	4	8.52	1.54	0.03	0.07	3739.59	1626.73	2.23	0.42
oka1	5	10.86	1.55	0	0.08	46.19	548.72	2.27	0.22
Average		9.096	0.73	0.054	0.12	1204.96	492.644	1.428	0.22
St. Dev.		2.20632	0.748886	0.0391152	0.0897218	1450.57	663.826	1.13341	0.121861
p-value				0.401965		0.420635		0.222222	
oka2	1	0.13	0.19	0.48	1.04	56.83	2.79	2.65e-06	0.41
oka2	2	0.13	0.27	0.68	1.12	27.14	3.87	0	0.4
oka2	3	0.14	0.21	0.28	0.82	35.47	4.72	0	0.41
oka2	4	0.17	0.24	0.59	1.29	30.76	2.66	2.65e-06	0.53
oka2	5	0.14	0.35	0.79	1.08	17.08	3.18	0.04	0.41
Average		0.142	0.252	**0.564**	1.07	**33.456**	3.444	**0.00800106**	0.432
St. Dev.		0.0164317	0.0626099	0.195525	0.169115	14.7117	0.854652	0.0178880	0.0549545
p-value				0.0079365		0.0079365		0.0106623	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
vlmop2	1	151.05	4.8	0	9.59e-05	172.39	668.38	0	0
vlmop2	2	166.22	4.79	0	9.74e-05	175.85	666	0	0
vlmop2	3	168.27	4.88	0	9.72e-05	171.92	670.42	0	0
vlmop2	4	150.2	4.9	0	9.68e-05	172.78	668.07	0	0
vlmop2	5	149.66	4.87	0	9.68e-05	170.7	666.31	0	0
Average		157.08	4.848	**0**	9.682e-05	172.728	**667.836**	0	0
St. Dev.		9.32078	0.0496991	0	5.76194e-07	1.91243	1.78377	0	0
p-value				0.00729036		0.0079365		NaN	
vlmop3	1	208.55	4.44	0.01	0	2664.43	8818.84	0.01	0.02
vlmop3	2	196	3.76	0.01	0	2645.68	9722.25	0	0.02
vlmop3	3	187.25	4.2	0.01	0	2665.84	10136.8	0.01	0.01
vlmop3	4	197.72	4	0.01	0	2570.03	9900.39	0.01	0.01
vlmop3	5	203.91	4.54	0.01	0	2579.12	8553.37	0.01	0.02
Average		198.686	4.188	0.01	**0**	2625.02	**9426.33**	0.008	0.016
St. Dev.		8.11663	0.318622	0	0	46.8427	697.885	0.00447214	0.00547723
p-value				0.00397675		0.0079365		0.0558293	
dtlz1a	1	34.08	0	0	8.24e-07	201126	75.97	0	5.13
dtlz1a	2	48.5	0	0	0.14	35800000	12765.4	0	66.22
dtlz1a	3	32.06	0	0	8.66e-07	177832	75.97	0	5.13
dtlz1a	4	34.64	0	0	5.12e-10	194033	75.97	0	5.13
dtlz1a	5	36.02	0	0	0	5996.98	0.05	0	9.39
Average		37.06	0	**0**	0.0280003	**7275798**	2598.67	**0**	18.2
St. Dev.		6.55156	0	0	0.0626097	15945717	5683.47	0	26.9073
p-value				0.0253699		0.0200082		0.00669438	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
dtlz2a	1	1472.8	0	0.01	0.22	624.26	0.91	0.02	0.23
dtlz2a	2	1669.23	0	0.01	0.22	628.39	0.91	0.01	0.22
dtlz2a	3	1792.59	0	0.01	0.22	626.14	0.91	0.04	0.23
dtlz2a	4	1530.09	0	0.01	0.22	604.02	0.91	0.01	0.23
dtlz2a	5	1452.7	0	0.01	0.22	623.12	0.91	0.02	0.22
Average		1583.48	0	**0.01**	0.22	**621.186**	0.91	**0.02**	0.226
St. Dev.		144.295	0	0	0	9.8018	0	0.0122474	0.00547723
p-value				0.00397675		0.00749496		0.0104178	
dtlz4a	1	1459.44	0	0.01	1.33e-11	524.88	0.04	0	1.06
dtlz4a	2	1368.28	0	0.01	0	525.61	0.02	1.39e-06	1.06
dtlz4a	3	1746.44	0	0.01	1.27e-11	509.45	0.04	0	1.06
dtlz4a	4	1386.06	0	0.01	0	533.43	0.02	0.01	1.06
dtlz4a	5	1502.78	0	0.01	4.25e-08	515.87	0.04	0	1.06
Average		1492.6	0	0.01	**8.5052e-09**	**521.848**	0.032	**0.00200028**	1.06
St. Dev.		152.023	0	0	1.90037e-08	9.3131	0.0109545	0.00447198	0
p-value				0.00729036		0.0109095		0.00669438	
dtlz7a	1	384.45	0	0.01	0.37	826.02	1.1	0	0.28
dtlz7a	2	372.22	0	0.01	0.37	742.31	0.94	0	0.29
dtlz7a	3	389.08	0	0.01	0.37	841.57	1.05	0	0.28
dtlz7a	4	367.44	0	0.01	0.37	892.92	1.31	0.01	0.28
dtlz7a	5	379.67	0	0.01	0.37	808.86	1.03	0.01	0.28
Average		378.572	0	**0.01**	0.37	**822.336**	1.086	**0.004**	0.282
St. Dev.		8.80971	0	0	0	54.6612	0.137949	0.00547723	0.00447214
p-value				0.00397675		0.0079365		0.00856192	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
sdflp	1	74.03	Inf	0	2.32	0	6.32e+08	0	Inf
sdflp	2	74.99	Inf	0	0.18	0	7.81e+08	0	Inf
sdflp	3	68.56	Inf	0	0.21	0	7.27e+08	0	Inf
sdflp	4	72.7	Inf	0	2.09	0	6.52e+08	0	Inf
sdflp	5	79.59	Inf	0	0.06	0	9.35e+08	0	Inf
Average		73.974	Inf	0	0.972	0	745400000	0	Inf
St. Dev.		3.98464	NaN	0	1.12990	0	121590707	0	NaN
p-value				0.00749496		0.00749496		Inf	

Table A.1: Evolve I/O and Single Sigma Indicators

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
kno1	1	16.36	18.24	0.02	0	19462.2	428537	0.04	0.16
kno1	2	17.14	17.54	0.02	0	21270.8	411739	0.06	0.89
kno1	3	17.72	17.32	0.02	0	20381.3	428706	0.04	0.06
kno1	4	18.58	17.71	0.02	0	20820.9	434931	0.05	0.03
kno1	5	18.27	16.92	0.02	0	21254.9	423562	0.05	0.22
Average		17.614	17.55	0.02	**0**	20638	**425495**	0.048	0.272
St. Dev.		0.890494	0.49	0	0	752.14	8682.83	0.0083666	0.353794
p-value				0.00397675		0.0079365		0.170587	
oka1	1	12.34	0.54	0.09	0.03	1881.62	594.66	0.36	0.42
oka1	2	5.78	11.57	0.09	0	4518.99	28735.5	3.5e-06	0.67
oka1	3	10.44	19.06	0.06	0	2933.96	33502.8	4.16e-06	1.84
oka1	4	9.64	1.57	0.03	0.07	3775.46	1601.57	2.23	0.42
oka1	5	11.05	0.26	0	0.09	1309.98	115.76	2.3	0.39
Average		9.85	6.6	0.054	0.038	2884	12910.1	0.978002	0.748
St. Dev.		2.47978	8.4	0.0391152	0.0408656	1318.40	16716.4	1.18428	0.620862
p-value				0.591208		0.84127		0.675174	
oka2	1	0.09	8.12	0.48	0.25	295.43	2171.74	0.01	2.64
oka2	2	0.08	11.37	0.68	0	26.65	160.2	-0.01	3.35
oka2	3	0.09	8.88	0.28	0	25.5	100.82	-0.05	3.02
oka2	4	0.14	76.37	0.59	0.01	86.76	4755.86	-0.02	3.14
oka2	5	0.08	19.88	0.79	0.17	76.66	1392.24	0.34	2.17
Average		0.096	24.92	0.564	**0.086**	102.2	**1716.17**	**0.054**	2.864
St. Dev.		0.0250998	29.14	0.195525	0.116748	111.601	1909.36	0.161338	0.465972
p-value				0.0119252		0.031746		0.0079365	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
vlmop2	1	216.14	4.8	0	9.59e-05	172.39	668.39	0	0
vlmop2	2	225.75	4.79	0	9.75e-05	175.78	666.16	0	0
vlmop2	3	219.5	4.88	0	9.72e-05	171.92	670.42	0	0
vlmop2	4	211.48	4.9	0	9.68e-05	172.78	668.07	0	0
vlmop2	5	216.09	4.9	0	9.68e-05	170.15	666.03	0	0
Average		217.792	4.85	**0**	9.684e-05	172.604	**667.814**	0	0
St. Dev.		5.28475	0.05	0	6.02495e-07	2.04045	1.81017	0	0
p-value				0.00729036		0.0079365		NaN	
vlmop3	1	272.31	4.44	0.01	0	2664.43	8818.86	0.01	0.02
vlmop3	2	247.03	4.37	0.01	0	2636.05	8347.01	0	0.01
vlmop3	3	249.95	4.55	0.01	0	2634.8	7405.36	0.01	0.02
vlmop3	4	279.13	4.55	0.01	0	2565.57	8246.04	0.01	0.02
vlmop3	5	269.31	4.54	0.01	0	2579.12	8553.37	0.01	0.02
Average		263.546	4.49	0.01	**0**	2615.99	**8274.13**	**0.008**	0.018
St. Dev.		14.2348	0.08	0	0	41.8456	532.815	0.00447214	0.00447214
p-value				0.00397675		0.0079365		0.0209213	
dtlz1a	1	28.98	4.22	0	3.1e-10	115.49	1131.31	0	1.95e-05
dtlz1a	2	45.8	4.18	0	3.68e-08	115.63	1132.2	0	7.82e-05
dtlz1a	3	29.95	4.77	0	3.31e-09	103.25	1131.8	0	8.53e-05
dtlz1a	4	31.09	4.37	0	1.48e-10	110.85	1131.25	0	1.95e-05
dtlz1a	5	33.89	4.26	0	1.12e-09	113.84	1131.35	0	3.68e-05
Average		33.942	4.36	**0**	8.3376e-09	111.812	**1131.58**	**0**	4.786e-05
St. Dev.		6.87943	0.24	0	1.59608e-08	5.15875	0.408497	0	3.18323e-05
p-value				0.00749496		0.0079365		0.00729036	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
dtlz2a	1	1162.88	1.59	0.01	0	447.45	2085.62	0.02	0.58
dtlz2a	2	1316.06	1.58	0.01	6.14e-06	448.95	2066.8	0.01	0.59
dtlz2a	3	1613.16	1.59	0.01	0	460.95	2111.05	0.02	0.57
dtlz2a	4	1187.58	1.63	0.01	7.42e-05	430.12	2057.41	0.02	0.59
dtlz2a	5	1236.02	1.59	0.01	6.31e-06	456.11	2102.96	0.02	0.58
Average		1303.14	1.6	0.01	**1.733e-05**	448.716	**2084.77**	**0.018**	0.582
St. Dev.		182.901	0.02	0	3.19434e-05	11.7453	22.8786	0.00447214	0.0083666
p-value				0.00729036		0.0079365		0.00923674	
dtlz4a	1	1177.44	1.24	0.01	0	558.32	779	0.01	0.25
dtlz4a	2	1069.48	1.61	0.01	0	439.89	879.66	0.01	0.4
dtlz4a	3	1374.08	1.68	0.01	0	428.84	645.88	0.02	0.25
dtlz4a	4	1081.72	1.64	0.01	0	447.86	686.54	0.01	0.69
dtlz4a	5	1203.84	1.65	0.01	0	430.1	682.57	0.01	0.69
Average		1181.31	1.56	0.01	**0**	461.002	**734.73**	**0.012**	0.456
St. Dev.		122.584	0.18	0	0	54.9512	94.7389	0.00447214	0.222216
p-value				0.00397675		0.0079365		0.00923674	
dtlz7a	1	305.94	2.51	0.01	9.77e-05	645.08	1046.92	0	1.72
dtlz7a	2	333.89	2.57	0.01	0	574.57	1282.63	0	1.63
dtlz7a	3	339.38	2.23	0.01	0	899	1376.93	4.4e-11	1.48
dtlz7a	4	313.48	2.56	0.01	9.85e-05	698.19	1242.01	0	1.72
dtlz7a	5	346.92	2.52	0.01	0	725.65	1361.57	0	1.62
Average		327.922	2.48	0.01	**3.924e-05**	708.498	**1262.01**	**8.8e-12**	1.634
St. Dev.		17.4614	0.14	0	5.37323e-05	121.089	132.484	1.96774e-11	0.098387
p-value				0.00669438		0.0079365		0.00946735	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
sdflp	1	79.23	Inf	0	2.35	0	6.11e+08	0	Inf
sdflp	2	82.19	Inf	0	2.18	0	7.1e+08	0	Inf
sdflp	3	81.45	Inf	0	0.22	0	7.27e+08	0	Inf
sdflp	4	88.11	Inf	0	2.09	0	6.52e+08	0	Inf
sdflp	5	86.91	Inf	0	2	0	8.36e+08	0	Inf
Average		83.578	Inf	0	1.768	0	707200000	0	Inf
St. Dev.		3.77496	NaN	0	0.874969	0	85572776	0	NaN
p-value				0.00749496		0.00749496		Inf	

Table A.2: Heuristic I/O and Evolve Single Sigma Indicators

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
kno1	1	15.89	0.1	0.02	2.78e-05	19462.2	2834.98	0.01	5.22
kno1	2	16.28	0.097	0.02	5.59e-05	21270.8	2727.24	0.02	6.02
kno1	3	17.27	0.096	0.02	1.76e-05	20381.3	2910.33	0.01	4.92
kno1	4	17.59	0.097	0.02	8.7e-07	20820.9	2889.3	0.02	4.46
kno1	5	17.97	0.092	0.02	2.09e-05	21254.9	2828.88	0.02	5
Average		17	0.096	0.02	**2.4614e-05**	**20638**	2838.15	**0.016**	5.124
St. Dev.		0.8821	0.003	0	2.00981e-05	752.14	71.1043	0.00547723	0.572259
p-value				0.00749496		0.0079365		0.0109095	
oka1	1	12.5	17.66	0.09	9.75e-05	4183.03	32559.8	-0.48	3.83
oka1	2	5.89	18.39	0.09	0	5322.52	48700.1	-0.56	3.98
oka1	3	10.61	13.64	0.06	0	2948.77	14011	-0.05	1.84
oka1	4	9.99	7.06	0.03	0	2614.18	5570.1	-0.03	1.89
oka1	5	11.06	1.36	0	0	15.72	8.1	-0.2	1.81
Average		10.01	11.62	0.054	**1.95e-05**	3016.84	20169.8	**-0.264**	2.67
St. Dev.		2.48180	7.29	0.0391152	4.36033e-05	1990.58	20156.1	0.244397	1.12900
p-value				0.0441713		0.150794		0.0079365	
oka2	1	0.09	261.9	0.48	0	279.63	8578.79	-1.03	4.33
oka2	2	0.08	366.7	0.68	0	107.69	10297.1	-0.95	4.69
oka2	3	0.13	323.5	0.28	0	151.96	12393.3	-1.16	4.88
oka2	4	0.17	289.5	0.59	0	152.47	9717.75	-0.88	4.96
oka2	5	0.09	421.8	0.79	0	71.91	6297.18	-0.63	4.53
Average		0.112	332.7	0.564	**0**	152.732	**9456.82**	**-0.93**	4.678
St. Dev.		0.0376829	63.4	0.195525	0	78.525	2244.51	0.197358	0.256652
p-value				0.00749496		0.0079365		0.0079365	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
vlmop2	1	218.36	4.79	0	3.99e-05	172.39	860.17	0	0.39
vlmop2	2	226.56	4.73	0	3.97e-05	175.43	856.28	0	0.4
vlmop2	3	224.97	4.81	0	3.98e-05	171.92	859.42	0	0.4
vlmop2	4	213.2	4.83	0	3.95e-05	171.93	853.48	0	0.39
vlmop2	5	213.38	4.83	0	3.92e-05	169.43	852	0	0.39
Average		219.294	4.8	**0**	3.962e-05	172.22	**856.27**	**0**	0.394
St. Dev.		6.28483	0.04	0	2.77489e-07	2.13841	3.57595	0	0.00547723
p-value				0.00749496		0.0079365		0.0065017	
vlmop3	1	269.8	4.44	0.01	0	11167	14052.9	0	1.23
vlmop3	2	244.34	4.37	0.01	0	11454.4	14088.5	0	1.23
vlmop3	3	243.61	4.55	0.01	0	10957.6	13638.6	0	1.23
vlmop3	4	262.47	4.55	0.01	0	11302.8	14824.4	0	1.23
vlmop3	5	262.02	4.54	0.01	0	10858.2	13894.4	0	1.23
Average		256.448	4.49	0.01	**0**	11148	**14099.8**	**0**	1.23
St. Dev.		11.8005	0.08	0	0	244.152	442.229	0	0
p-value				0.00397675		0.0079365		0.00397675	
dtlz1a	1	30.84	4.22	0	6.13e-11	115.49	1131.28	0	1.95e-05
dtlz1a	2	45.64	4.18	0	1.07e-10	115.55	1131.49	0	1.95e-05
dtlz1a	3	30.09	4.77	0	1.88e-10	103.24	1131.69	0	6.85e-05
dtlz1a	4	31.28	4.37	0	8.41e-11	110.85	1131.26	0	1.95e-05
dtlz1a	5	32.64	4.26	0	2.04e-09	113.84	1131.32	0	3.6e-05
Average		34.098	4.36	**0**	4.9608e-10	111.794	**1131.41**	**0**	3.26e-05
St. Dev.		6.5185	0.24	0	8.644e-10	5.14821	0.181852	0	2.13026e-05
p-value				0.00749496		0.0079365		0.00669438	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
dtlz2a	1	1263.83	1.59	0.01	6.13e-06	447.45	2077.58	0.01	0.6
dtlz2a	2	1317.97	1.58	0.01	6.16e-06	448.95	2066.83	0.01	0.59
dtlz2a	3	1595.53	1.59	0.01	6.18e-06	460.95	2107.93	0.02	0.59
dtlz2a	4	1393.7	1.63	0.01	6.2e-06	430.12	2054.71	0.01	0.59
dtlz2a	5	1282.77	1.59	0.01	6.2e-06	456.11	2103.96	0.02	0.59
Average		1370.76	1.6	0.01	**6.174e-06**	448.716	**2082.2**	**0.014**	0.592
St. Dev.		135.103	0.02	0	2.96648e-08	11.7453	23.1776	0.00547723	0.00447214
p-value				0.00729036		0.0079365		0.00856192	
dtlz4a	1	1232.91	1.62	0.01	0	438.8	877.96	0.01	0.69
dtlz4a	2	1077.58	1.61	0.01	0	439.89	584.27	0.01	0.69
dtlz4a	3	1389.55	1.68	0.01	0	428.84	765.54	0.01	0.69
dtlz4a	4	1086.53	1.64	0.01	0	447.86	702.74	0.01	0.69
dtlz4a	5	1205.44	1.65	0.01	0	430.1	400.61	0.01	0.69
Average		1198.40	1.64	0.01	**0**	437.098	666.224	**0.01**	0.69
St. Dev.		127.369	0.03	0	0	7.80511	182.549	0	0
p-value				0.00397675		0.150794		0.00397675	
dtlz7a	1	324.81	2.51	0.01	0	884.33	1642.18	0	1.72
dtlz7a	2	343.77	2.57	0.01	9.84e-05	574.57	915.58	0	1.71
dtlz7a	3	334.34	2.44	0.01	9.63e-05	701.17	1093.68	2e-12	1.71
dtlz7a	4	311.61	2.56	0.01	9.76e-05	698.19	1198.54	0	1.72
dtlz7a	5	353.75	2.52	0.01	9.75e-05	681.34	1127.08	0	1.73
Average		333.656	2.52	0.01	**7.796e-05**	707.92	**1195.41**	**4e-13**	1.718
St. Dev.		16.3616	0.05	0	4.35874e-05	111.523	270.618	8.94427e-13	0.0083666
p-value				0.00749496		0.0079365		0.00923674	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
sdflp	1	82.72	Inf	0	0.02	0	1.34e+09	0	Inf
sdflp	2	83.84	Inf	0	0.02	0	1.35e+09	0	Inf
sdflp	3	79.53	Inf	0	0.02	0	1.32e+09	0	Inf
sdflp	4	89.34	Inf	0	0.02	0	1.27e+09	0	Inf
sdflp	5	89.5	Inf	0	0.02	0	1.32e+09	0	Inf
Average		84.986	Inf	0	0.02	0	1.32e+09	0	Inf
St. Dev.		4.34596	NaN	0	0	0	30822070	0	NaN
p-value				0.00397675		0.00729036		Inf	

Table A.3: Heuristic I/O and Evolve Multiple Sigmas Indicators

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
kno1	1	0.06	18.24	0.02	0	19462.2	416652	0.04	0.14
kno1	2	0.08	17.54	0.02	0	21270.8	410043	0.06	0.87
kno1	3	0.09	17.32	0.02	0	20381.3	427874	0.04	0.05
kno1	4	0.09	17.71	0.02	0	20820.9	434539	0.05	0.01
kno1	5	0.09	16.92	0.02	0	21254.9	418882	0.05	0.22
Average		0.082	17.55	0.02	**0**	20638	**421598**	0.048	0.258
St. Dev.		0.0130384	0.49	0	0	752.14	9646.44	0.0083666	0.351667
p-value				0.00397675		0.0079365		0.288844	
oka1	1	0.08	0.18	0.09	0.01	766.48	44.28	0.32	0.42
oka1	2	0.02	0.3	0.09	0.05	1798.47	437.12	0.19	0.38
oka1	3	0.05	0.3	0.06	0.09	2024.26	364.63	2.29	0.41
oka1	4	0.05	0.4	0.03	0.07	1966.83	616.08	2.23	0.42
oka1	5	0.08	0.35	0	0.07	1363.74	128.21	2.3	0.39
Average		0.056	0.31	0.054	0.058	**1583.96**	318.064	1.466	0.404
St. Dev.		0.0250998	0.08	0.0391152	0.0303315	525.176	232.464	1.10677	0.0181659
p-value				1		0.0079365		0.675174	
oka2	1	0	26.07	0.48	0.01	334.72	7697.9	-6e-05	2.96
oka2	2	0	12	0.68	0	26.65	172.25	-0.01	3.35
oka2	3	0	19.21	0.28	0	34.08	361.72	-0.04	3
oka2	4	0	4.97	0.59	0.15	14.36	78.22	0.28	3.14
oka2	5	0	196.6	0.79	0.02	111.32	27523.1	-0.01	3
Average		0	51.77	0.564	**0.036**	104.226	**7166.64**	**0.043988**	3.09
St. Dev.		0	81.35	0.195525	0.0642651	134.343	11833.7	0.132783	0.160624
p-value				0.0119252		0.0555556		0.0116673	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
vlmop2	1	0.98	3.76	0	0	172.39	561.49	0	0
vlmop2	2	1.06	3.74	0	0	176.19	569.02	0	0
vlmop2	3	1.02	3.86	0	0	171.95	575.43	0	0
vlmop2	4	0.99	3.89	0	0	172.99	573.75	0	0
vlmop2	5	0.97	3.8	0	0	170.77	563.48	0	0
Average		1.004	3.81	0	0	172.858	**568.634**	0	0
St. Dev.		0.0364692	0.06	0	0	2.03242	6.12594	0	0
p-value				NaN		0.0079365		NaN	
vlmop3	1	1.31	4.2	0.01	0	2653.55	7242.62	0.01	0.01
vlmop3	2	1.28	4.04	0.01	0	2652.53	6908.2	0.01	0.01
vlmop3	3	1.28	4.24	0.01	0	2666.58	6920.45	0.01	0.01
vlmop3	4	1.34	4.21	0.01	0	2583.81	6775.09	0.01	0.01
vlmop3	5	1.34	4.41	0.01	0	2558.02	7384.96	0.01	0.01
Average		1.31	4.22	0.01	**0**	2622.9	**7046.26**	0.01	0.01
St. Dev.		0.03	0.13	0	0	48.6383	255.782	0	0
p-value				0.00397675		0.0079365		NaN	
dtlz1a	1	0.11	4.22	0	4.62e-07	115.66	1132.45	0	0
dtlz1a	2	0.17	4.18	0	1.46e-06	116.8	1143.09	0	0
dtlz1a	3	0.13	4.77	0	6.13e-07	103.29	1131.59	0	0
dtlz1a	4	0.13	4.37	0	1.15e-05	121.03	1233.13	0	0
dtlz1a	5	0.14	4.26	0	6.06e-05	170.04	1689.31	0	0
Average		0.136	4.36	**0**	1.4927e-05	125.364	**1265.91**	0	0
St. Dev.		0.0219089	0.24	0	2.59483e-05	25.8336	240.458	0	0
p-value				0.00749496		0.0079365		NaN	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
dtlz2a	1	5.34	1.59	0.01	0	458.42	1744.09	0.02	0.12
dtlz2a	2	6.03	1.58	0.01	0	448.95	1679.39	0.02	0.15
dtlz2a	3	7	1.59	0.01	0	460.95	1742.37	0.02	0.1
dtlz2a	4	5.72	1.63	0.01	0	439.72	1702.93	0.03	0.11
dtlz2a	5	5.39	1.59	0.01	0	456.11	1693.31	0.02	0.14
Average		5.896	1.6	0.01	**0**	452.83	**1712.42**	**0.022**	0.124
St. Dev.		0.676927	0.02	0	0	8.58585	29.3523	0.00447214	0.0207364
p-value				0.00397675		0.0079365		0.00970079	
dtlz4a	1	5.41	0.36	0.01	0.01	453.48	151.58	0.02	0.24
dtlz4a	2	5	0.35	0.01	0.01	460.1	149.9	0.01	0.24
dtlz4a	3	6.8	0.42	0.01	0.01	433.7	164.97	0.02	0.22
dtlz4a	4	5.17	0.32	0.01	0.01	492.6	163.09	0.02	0.24
dtlz4a	5	6.58	0.36	0.01	0.01	439.72	147.48	0.02	0.25
Average		5.792	0.36	0.01	0.01	**455.92**	155.404	**0.018**	0.238
St. Dev.		0.836224	0.04	0	0	23.0490	8.0357	0.00447214	0.0109545
p-value				NaN		0.0079365		0.008784	
dtlz7a	1	1.22	0.96	0.01	0.01	660.75	739.88	0.01	0.16
dtlz7a	2	1.16	0.99	0.01	0.01	699.96	823.73	0.01	0.12
dtlz7a	3	1.67	0.94	0.01	0.01	667.22	710.63	0.01	0.17
dtlz7a	4	1.13	0.996	0.01	0.01	734.58	785.7	0.01	0.11
dtlz7a	5	1.22	0.93	0.01	0.01	660.63	634.82	0.01	0.13
Average		1.28	0.96	0.01	0.01	684.628	738.952	**0.01**	0.138
St. Dev.		0.221472	0.03	0	0	32.3240	72.4899	0	0.0258844
p-value				NaN		0.222222		0.00749496	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
sdflp	1	0.42	Inf	0	2.31	0	6.34e+08	0	Inf
sdflp	2	0.39	Inf	0	2.36	0	6.16e+08	0	Inf
sdflp	3	0.42	Inf	0	0.24	0	7.62e+08	0	Inf
sdflp	4	0.41	Inf	0	2.31	0	6.05e+08	0	Inf
sdflp	5	0.42	Inf	0	0.23	0	7.6e+08	0	Inf
Average		0.412	Inf	0	1.49	0	675400000	0	Inf
St. Dev.		0.0130384	NaN	0	1.14584	0	78827660	0	NaN
p-value				0.00729036		0.00749496		Inf	

Table A.4: Heuristic I/O and Heuristic Single Sigma Indicators

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
kno1	1	0.06	18.24	0.02	0	19462.2	426036	0.04	0.15
kno1	2	0.08	17.54	0.02	0	21270.8	409695	0.06	0.87
kno1	3	0.08	17.32	0.02	0	20381.3	425499	0.04	0.06
kno1	4	0.09	17.71	0.02	0	20820.9	432499	0.05	0.02
kno1	5	0.08	16.92	0.02	0	21254.9	417180	0.05	0.21
Average		0.078	17.55	0.02	**0**	20638	**422182**	0.048	0.262
St. Dev.		0.0109545	0.49	0	0	752.14	8849.31	0.0083666	0.347951
p-value				0.00397675		0.0079365		0.170587	
oka1	1	0.06	0.51	0.09	0.14	766.37	172.71	0.31	0.72
oka1	2	0.03	3.07	0.09	0	1546.46	1952.5	0.18	2.41
oka1	3	0.06	10.29	0.06	0	2542.74	18879.1	2.26	0.65
oka1	4	0.05	9.04	0.03	0.04	4220.01	18526.7	2.2	0.6
oka1	5	0.06	10.27	0	0.02	92.14	11343.3	2.27	0.59
Average		0.052	6.64	0.054	0.04	1833.54	10174.9	1.444	0.994
St. Dev.		0.0130384	4.54	0.0391152	0.0583095	1616.47	8867.53	1.09582	0.79324
p-value				0.524518		0.222222		1	
oka2	1	0	6.6	0.48	0.01	41.61	112.75	0.02	3.05
oka2	2	0	12.47	0.68	0	26.65	130.18	-0.01	3.35
oka2	3	0	30.62	0.28	0	50.76	589.98	-0.05	3.37
oka2	4	0	0.63	0.59	0.53	14.36	7.14	0.28	3.14
oka2	5	0	224.31	0.79	0	76.16	5127.17	-5.96e-10	1.68
Average		0	54.93	0.564	**0.108**	41.908	1193.44	**0.048**	2.918
St. Dev.		0	95.35	0.195525	0.235945	23.682	2210.44	0.132174	0.705386
p-value				0.0344536		0.150794		0.0079365	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
vlmop2	1	0.94	3.93	0	0	172.39	560.98	0	0
vlmop2	2	1.05	3.85	0	0	176.19	558.56	0	0
vlmop2	3	0.98	4.07	0	0	171.92	571.77	0	0
vlmop2	4	0.95	4.03	0	0	172.87	564.65	0	0
vlmop2	5	0.91	4.03	0	0	170.78	563.64	0	0
Average		0.966	3.98	0	0	172.83	**563.92**	0	0
St. Dev.		0.0531977	0.09	0	0	2.03196	4.99017	0	0
p-value				NaN		0.0079365		NaN	
vlmop3	1	1.27	4.44	0.01	0	2664.43	7629.78	0.01	0.02
vlmop3	2	1.25	4.3	0.01	0	2680.45	6935.73	0	0.01
vlmop3	3	1.22	4.15	0.01	0	2692.73	5029.64	0.01	0.05
vlmop3	4	1.27	4.41	0.01	0	2565.49	6802.92	0.01	0.01
vlmop3	5	1.27	4.08	0.01	0	2518.74	4756.18	0.02	0.05
Average		1.256	4.28	0.01	**0**	2624.37	**6230.85**	0.01	0.028
St. Dev.		0.0219089	0.16	0	0	77.5361	1264.78	0.00707107	0.0204939
p-value				0.00397675		0.0079365		0.144698	
dtlz1a	1	0.11	4.22	0	4.06e-07	115.64	1132.27	0	0
dtlz1a	2	0.17	4.18	0	1.81e-06	117	1145.03	0	0
dtlz1a	3	0.11	4.77	0	6e-07	103.31	1131.78	0	0
dtlz1a	4	0.11	4.37	0	1.19e-05	121.84	1241.28	0	0
dtlz1a	5	0.13	4.26	0	4.71e-05	163.28	1621.78	0	0
Average		0.126	4.36	**0**	1.23632e-05	124.214	**1254.43**	0	0
St. Dev.		0.0260768	0.24	0	1.99974e-05	22.8819	210.389	0	0
p-value				0.00749496		0.0079365		NaN	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
dtlz2a	1	5.17	1.59	0.01	0	572.24	2179.57	0.02	0.19
dtlz2a	2	5.78	1.58	0.01	0	448.95	1745.74	0.02	0.15
dtlz2a	3	6.75	1.59	0.01	0	467.11	1817.43	0.02	0.17
dtlz2a	4	5.52	1.63	0.01	0	720.62	2723.83	0.03	0.17
dtlz2a	5	5.19	1.59	0.01	0	456.11	1753.36	0.02	0.17
Average		5.682	1.6	0.01	**0**	533.006	**2043.99**	**0.022**	0.17
St. Dev.		0.648205	0.02	0	0	116.253	419.914	0.00447214	0.0141421
p-value				0.00397675		0.0079365		0.008784	
dtlz4a	1	5.16	0.99	0.01	0.01	485.41	475.8	0.02	0.23
dtlz4a	2	4.83	0.998	0.01	0.01	439.89	375.24	0.02	0.12
dtlz4a	3	6.58	1.56	0.01	0.01	432.7	495.03	0.02	0.09
dtlz4a	4	5.05	0.94	0.01	0.01	448.72	343.72	0.02	0.36
dtlz4a	5	6.38	0.89	0.01	0.01	432.39	220.29	0.02	0.1
Average		5.6	1.08	0.01	0.01	447.822	382.016	**0.02**	0.18
St. Dev.		0.815138	0.27	0	0	22.042	110.945	0	0.115109
p-value				NaN		0.547619		0.00749496	
dtlz7a	1	1.16	1.0	0.01	0.01	781.96	817.73	0.01	0.41
dtlz7a	2	1.11	1.33	0.01	0.01	928.14	1234.17	0.01	0.33
dtlz7a	3	1.33	0.98	0.01	0.01	844.32	838.98	0.01	0.15
dtlz7a	4	1.25	1.98	0.01	0.01	969.05	1465.9	0.01	0.14
dtlz7a	5	1.28	1.55	0.01	0.01	898.46	1049.05	0.01	0.16
Average		1.226	1.37	0.01	0.01	884.386	1081.17	**0.01**	0.238
St. Dev.		0.0896103	0.42	0	0	73.1041	274.096	0	0.123976
p-value				NaN		0.420635		0.00749496	

Continued on Next Page...

Problem	File	Training Time (s)	Yield Ratio	NSGA Spacing	NEMO Spacing	NSGA Hypervolume	NEMO Hypervolume	NSGA-NEMO Binary $\epsilon+$	NEMO-NSGA Binary $\epsilon+$
sdflp	1	0.42	Inf	0	2.14	0	8.02e+08	0	Inf
sdflp	2	0.42	Inf	0	2.14	0	7.7e+08	0	Inf
sdflp	3	0.44	Inf	0	2.11	0	8.18e+08	0	Inf
sdflp	4	0.42	Inf	0	0.17	0	7.65e+08	0	Inf
sdflp	5	0.45	Inf	0	2.01	0	9.63e+08	0	Inf
Average		0.43	Inf	0	1.714	0	823600000	0	Inf
St. Dev.		0.0141421	NaN	0	0.864772	0	80989505	0	NaN
p-value				0.00729036		0.00749496		Inf	

Table A.5: Heuristic I/O and Heuristic Multiple Sigmas Indicators