ABSTRACT
In previous work, a neural network was used to increase the number of solutions found by an evolutionary multiobjective optimization algorithm. In this paper, various approaches are applied in the training of the neural network to determine whether an approach exists that can provide reasonable results in a reasonable time. To this end, two heuristic training algorithms are developed. When evaluated on a suite of ten benchmark multiobjective optimization problems, these heuristic techniques perform very well and, on average, produce many more solutions than the evolutionary multiobjective optimization approach alone.

Categories and Subject Descriptors
G.1.6 [Numerical Analysis]: Optimization—multiobjective optimization

General Terms
Multiobjective optimization

Keywords
neural networks, multiobjective optimization, genetic algorithms

1. INTRODUCTION
Optimization problems are commonly found in real world applications. However, many optimization problems involve multiple, often conflicting, objectives. 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. The more elements in such a set, the more options that are available as possible solutions.

Many multiobjective optimization techniques have been developed over the years, 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 [3], and the nondominated sorting genetic algorithm [6,13] 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. In [7], a neural network system was developed that can, in essence, learn the areas where nondominated solutions are found so that many such solutions can be generated without the need for a large number of function evaluations. This work focuses on finding an efficient training algorithm for such a neural network system.

2. MULTIOBJECTIVE OPTIMIZATION

A multiobjective optimization (MO) problem is defined as follows:

Minimize $[f_1(\vec{x}), f_2(\vec{x}), \cdots, f_k(\vec{x})]$
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. We wish to find the values $x_1^*, x_2^*, \ldots, x_n^*$ that yield the optimum values for all the objective functions.

2.1 Optimization Approaches

Multiobjective optimization problems are very difficult because they often deal with conflicting objectives. Many different approaches have been applied to MO problems. Aggregation-based approaches [1,15] use a weighted sum of the objective values as the new objective in a single-objective optimization problem. Criterion-based approaches [2,15] consider only one objective of a MO problem at a time. In the simplest case, the objectives are ranked in order of importance, optimizing each one in turn without degrading the values of the previous objectives. Finally, Pareto-based approaches [2,15] use the concept of Pareto dominance to compare multiobjective solutions.

In a Pareto-based approach, 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 [2]. The image of the Pareto optimal set under the objective functions is called the Pareto optimal frontier (or often just the Pareto optimal front) [2]. Finally, since the global Pareto optimal set is often unknown, most multiobjective optimization approaches actually produce a Pareto set approximation [15]. In this work, the term “Pareto frontier” 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.

2.2 NSGA-II

The nondominated sorting genetic algorithm (NSGA) [13] was first introduced in 1994 by Srinivas and Deb as an evolutionary multiobjective optimizer. The NSGA algorithm operated like a simple genetic algorithm [8, 9] 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 [5].

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 [8], which allows good individuals to continue to survive and reproduce.

3. NEURAL ENHANCEMENT

To perform the neural enhancement for multiobjective optimization (NEMO) approach, a general regression neural network (GRNN) [12] is 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. In essence, 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.

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:
\[
d(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}
\]
Here, \(||\cdot||\) represents Euclidean distance, and \( \sigma \) 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 \( \sigma \) is used for all training inputs. Training a GRNN consists of merely finding the value of the smoothing parameter \( \sigma \) that provides the minimum error.

4. METHODOLOGY

Given a multiobjective optimization problem with \( k \) decision variables, \( d_1, \ldots, d_k \), NEMO first partitions the decision variables into two sets, \( I \) and \( O \). 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 in this work). Then, values 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 in set \( O \). Finally, the generated solution is passed to the multiobjective function to discover the objective values associated with it. These values determine whether the generated solution is nondominated.

4.1 Test Suite

In order to thoroughly test the neural enhancement system and the training approaches, all nine multiobjective benchmark problems were taken from [10]. These problems were KNO1, OKA1 [11], OKA2 [11], VLMOP2 [14], VLMOP3 [14], DTLZ1a, DTLZ2a, DTLZ4a, and DTLZ7a. Finally, the semi-desirable facility location problem (SD-FLP) [7] was included as an example of a real-world problem.

4.2 Performance Assessment

To evaluate the performance of the various training approaches, the relative yield, or yield ratio, was calculated. 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. Thus, the yield ratio is a binary indicator that is simply the yield of one MO algorithm divided by the yield of another. In this way, the yield ratio is a measure of performance. The yield ratio is a measure of performance.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Number of Decision Variables</th>
<th>Number of Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>kno1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>okl1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>okl2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>vlmop2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>vlmop3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>dtlz1a</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>dtlz2a</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>dtlz4a</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>dtlz7a</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>sdflp</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1: Multiobjective Optimization Test Suite
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 $B$.

Additionally, the training time for NEMO was recorded
for each training approach. This is the measure of 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 because
doing so would unnecessarily punish NEMO for generating
many solutions. Therefore, only the actual training time is
taken into account. 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.

5. 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, yield-
ing 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 addi-
tional 100 generations (for a total of 51200 function evalu-
ations). 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.

The implementation used for NSGA-II was taken from the
EMOO repository [4] and was implemented in C by Kalyan-
moy 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.

5.1 Evolve I/O and Evolve Single Sigma

A steady-state genetic algorithm (GA) [8] 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.
The GA was created with a population size of 100, uniform
crossover, and Gaussian mutation on the real-coded segment of the
chromosome (i.e., the $\sigma$ parameter). The crossover usage rate
was set to 1.0, the mutation rate for both types of mutations
were set to 0.1, and the mutation range for the Gaussian mutation was set to 1.0.
The allowed range of the $\sigma$ chromosome was given to be [0.01, 20.0] (which was
determined experimentally to produce decent performance).

To evaluate a given candidate $\sigma$ 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.

5.2 Heuristic I/O and Evolve Single Sigma

A very simple heuristic was used to determine the partic-
ular 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 re-
maining decision variables.

A steady-state genetic algorithm (GA) was used to evolve
the value of the GRNN’s $\sigma$ parameter. The GA was cre-
ated with a population size of 100, uniform crossover, and
Gaussian mutation on the real-coded segment of the chro-
mosome (i.e., the $\sigma$ parameter). 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. As before, the allowed range
of the $\sigma$ chromosome was given to be [0.01, 20.0] (which was
determined experimentally to produce decent performance).

To evaluate a given candidate $\sigma$ 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.

5.3 Heuristic I/O and Evolve Multiple Sigmas

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 smoothing parameter (i.e.,
$\sigma$ value) 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 $\sigma$ value was allowed to range between 0.01
and 20.0, inclusive. And, as before, evaluating a given set of
candidate $\sigma$ values was accomplished by using a 9:1 train-
ing/testing ratio and calculating the mean squared error on
the test set.

5.4 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 $\sigma$ values for the GRNN. 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 $\sigma$ value for element $p$.
This is repeated for all points in the Pareto optimal set.
Those calculated $\sigma$’s are then averaged together to arrive at
a single $\sigma$ for the GRNN.

5.5 Heuristic I/O and Heuristic Multiple Sigmas

In this final experiment, the heuristic was used to deter-
mine the I/O mask, and the heuristic in the previous section
was used to find a $\sigma$ value for each point in the Pareto op-
timal set. However, the $\sigma$ values found were used without
being averaged together.

6. RESULTS

Figures 1 and 2 provide graphical depictions of the results
of each training approach: 5.1 Evolve I/O and Evolve Single Sigma, 5.2 Heuristic I/O and Evolve Single Sigma, 5.3 Heuristic I/O and Evolve Multiple Sigmas, and 5.4 Heuristic I/O and Heuristic Single Sigma.
Table 2: NEMO Solutions for SDFLP Compared with NSGA-II (half)

<table>
<thead>
<tr>
<th>Approach</th>
<th>Average</th>
<th>St. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSGA-II (half)</td>
<td>1502.2</td>
<td>13.86</td>
</tr>
<tr>
<td>EIO-ESS</td>
<td>7079.2</td>
<td>1944.13</td>
</tr>
<tr>
<td>HIO-ESS</td>
<td>6315.4</td>
<td>751.04</td>
</tr>
<tr>
<td>HIO-EMS</td>
<td>12560.2</td>
<td>97.95</td>
</tr>
<tr>
<td>HIO-HSS</td>
<td>5617.8</td>
<td>352.1</td>
</tr>
<tr>
<td>HIO-HMS</td>
<td>6543</td>
<td>381.91</td>
</tr>
</tbody>
</table>

The results of this approach on the first run are depicted in Figure 3. In these graphs, the NSGA-II solutions are depicted in red while the NEMO solutions are in blue. Additionally, 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.

7. CONCLUSIONS AND FUTURE WORK

In this paper, the NEMO approach is shown to be capable of comparable training time to that of NSGA-II using a heuristic training algorithm. Just as importantly, using this efficient algorithm provides many more additional solutions than NSGA-II in the same number of function evaluations. On most problems in the test suite, NEMO was capable of finding more than four times as many additional solutions than NSGA-II. Additionally, on one particular problem (SDFLP), NSGA-II was unable to produce any additional solutions while NEMO produced many.

Clearly, the use of neural enhancement with NSGA-II can produce many more solutions for multiobjective optimization problems than NSGA-II alone. However, while those solution sets are bigger, it is not clear from this work whether they are necessarily “better” than those produced by NSGA-II. Additional work must be done to evaluate the efficient sets produced by each approach using additional indicators from the MO literature. It would also be interesting to explore whether other multiobjective approaches could be enhanced by NEMO, making them more successful than rival approaches that do not make use of neural enhancement.
8. REFERENCES


<table>
<thead>
<tr>
<th>Problem</th>
<th>EIO-ESS</th>
<th>HIO-ESS</th>
<th>HIO-EMS</th>
<th>HIO-HSS</th>
<th>HIO-HMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>Training Time</td>
<td>Yield Ratio</td>
<td>Training Time</td>
<td>Yield Ratio</td>
<td>Training Time</td>
</tr>
<tr>
<td>kno1</td>
<td>13.928</td>
<td>17.549</td>
<td>17.614</td>
<td>17.55</td>
<td>17</td>
</tr>
<tr>
<td>okal1</td>
<td>9.096</td>
<td>0.73</td>
<td>9.85</td>
<td>6.6</td>
<td>10.01</td>
</tr>
<tr>
<td>okal2</td>
<td>0.142</td>
<td>0.252</td>
<td>0.096</td>
<td>24.92</td>
<td>0.112</td>
</tr>
<tr>
<td>vlmop2</td>
<td>157.08</td>
<td>4.848</td>
<td>217.792</td>
<td>4.85</td>
<td>219.294</td>
</tr>
<tr>
<td>vlmop3</td>
<td>198.686</td>
<td>4.188</td>
<td>263.546</td>
<td>4.49</td>
<td>256.448</td>
</tr>
<tr>
<td>dtlz1a</td>
<td>37.06</td>
<td>0</td>
<td>33.942</td>
<td>4.36</td>
<td>34.098</td>
</tr>
<tr>
<td>dtlz2a</td>
<td>1583.48</td>
<td>0</td>
<td>1303.14</td>
<td>1.6</td>
<td>1370.76</td>
</tr>
<tr>
<td>dtlz4a</td>
<td>1492.6</td>
<td>0</td>
<td>1181.31</td>
<td>1.56</td>
<td>1198.40</td>
</tr>
<tr>
<td>dtlz7a</td>
<td>378.572</td>
<td>0</td>
<td>327.922</td>
<td>2.48</td>
<td>333.656</td>
</tr>
<tr>
<td>sd1lp</td>
<td>73.974</td>
<td>N/A</td>
<td>83.578</td>
<td>N/A</td>
<td>84.986</td>
</tr>
</tbody>
</table>