Analysis of the Pareto Front of a Multi-objective Optimization Problem for a Fossil Fuel Power Plant

Joel H. Van Sickel, Student Member, IEEE, Paramasivam Venkatesh, and Kwang Y. Lee, Fellow, IEEE

Abstract—This paper examines the Pareto front of a simple fossil fuel power plant using a common third-order model. This front is first examined analytically. Then the power plant model is transferred over to a steady-state model using a static neural network and the front is estimated using various geometric and heuristic approaches. This paper is the first of the two stages to eliminate the need for a human to construct a single objective cost function from multiple objectives for use in the power plant optimization. To generate the Pareto front of the power plant, three different optimization techniques are explored, normal-boundary intersection utilizing differential equation, multi-objective particle swarm optimization, and multi-objective evolutionary algorithm optimization.

Index Terms—Multi-objective evolutionary algorithm, multi-objective particle swarm optimization, normal-boundary intersection, Pareto front, Pareto optimization, power plant, reference governor.

I. INTRODUCTION

MULTI objective optimization is an interesting field with numerous approaches. These approaches can be split into two categories. The first type aggregates the objectives quantitatively into a single objective, the second approach uses the concept of Pareto optimality to find a trade-off curve. Aggregation of multiple objectives can be achieved through different ranking schemes such as weighted summation [1] or goal programming [2] to create a single function to be optimized. This assumes a quantitative knowledge of the importance of the different objectives. It is useful, however, because it returns a single solution. Finding the trade off curve, in which numerous points are returned, provides a comprehensive view of the optimization problem, but the large number of points are hard to visualize when more than three objectives are used. Also, most applications need a specific solution, not a set of points, to function properly. This creates a need to know how to choose a point on the trade-off curve.

Currently, multi-objective optimization work done for developing reference governors [3,4] has used aggregation to create single objective functions from multiple objectives for the control of power plants. Unfortunately, there is no explicit way to weight the different objectives, and without looking at the Pareto front, it is unknown what trade offs are being made. Therefore, this paper will explore the Pareto front of a simple power plant. This paper is devoted to methods of finding the Pareto front of a power plant on the assumption that a mathematical model is not available; however, due to the convenient form of this power plant model, the Pareto front can also be derived analytically to be used as a comparison to other techniques. For larger power plants, it is unlikely that an analytic solution of the Pareto front will be available.

This paper will address the issue in the following manner. First, Pareto optimality and reference governors will be discussed briefly, with adequate references to allow an understanding of the rest of the paper. Following will be an overview of the power plant model being used. Then, the analytic solution of the power plant’s Pareto front will be provided with some details of the derivation along with a static neural network model of this power plant. Then three different possibilities for heuristic multi-objective optimization will be analyzed. These three techniques will be normal-boundary intersection with differential evolution, multi-objective particle swarm optimization, and multi-objective evolutionary algorithm optimization. This will eliminate the need for a mathematical model and makes the approaches data driven.

II. MULTI-OBJECTIVE OPTIMIZATION AND PARETO OPTIMALITY

Multi-objective optimization has always been important as it is not very often that there is an ideal solution that provides the best result for every aspect of a problem. As such, great effort has gone into trying to develop methods to provide better ways to make decisions where trade offs are inevitable. When the problem can be quantified, equations can be used to give numerical values to different objectives representing how well they have been achieved. Initially, standard single
objective optimization techniques were used, but they required that different objectives be aggregated [1,2]. This was only a simplification. To truly deal with multi-objective solutions in their entirety, the concept of Pareto optimality was put forth.

Pareto optimality deals with multi-objective optimization where it is desired to minimize the \( N \) objectives \( f_i(x) \), where \( i = 1,2,...,N \), \( x \) is a vector of length \( k \), and \( k \) is the number of variables. This optimization can be posed as:

\[
\text{minimize } F(x) = [f_1(x), f_2(x),..., f_n(x)]
\]

Subject to constraints:

\[
g_i(x) \leq 0 \\
\]

\[
h_i(x) = 0 \\
\]

**Terminology:** Vector \( F(a) \) is said to dominate vector \( F(b) \), \( F(a) \prec F(b) \), if and only if \( f_i(a) \leq f_i(b) \) for all \( i \) and \( f_i(a) < f_i(b) \) for at least one \( i \). Vector \( F(a) \) is considered Pareto optimal or non-dominated if there exists no other vector \( F(b) \) such that \( F(b) \prec F(a) \).

There can be numerous points in a set that are all Pareto optimal, and it is actually very rare to have a situation in which there is only one non-dominated point. An in depth discussion of Pareto optimality is provided by Kim & de Weck [5].

### III. REVERENCE GOVERNOR

A reference governor provides feedforward control actions and set points to a feedback control system as shown in Fig. 1. Typically, a heuristic optimization is used in conjunction with a steady-state neural network model to search for ideal feed forward controls. Currently, multiple objectives are aggregated by optimizing a cost function that uses either weighted sums system [3] or goal programming [4]. Unfortunately, creating this cost function is challenging and does not provide an idea of what trade-offs are being made between different set points. Generating a Pareto front of the power plant’s steady state operation will allow a more informed decision to be made and possibly allow the elimination of the cost function or provide a specific process to determine the cost function that is more intuitive to a human. However, for this to be useful in real power plants, it must be possible to generate the Pareto front without a mathematical model, thus this paper will determine whether the use of neural networks and search algorithms can generate an accurate Pareto front of the power plant.

![Reference Governor](image)

**Fig. 1. Traditional Reference Governor for power plant.**

### IV. POWER PLANT

The power unit used for in this paper is a 160 MW oil-fired boiler-turbine unit as modeled by Bell & Astrom [6]. It is a three-input three-output system with three differential equations (1). The three inputs \( u_1 \), \( u_2 \) and \( u_3 \) are all control valves that effect mass flow rates within the system. Input \( u_1 \) controls the mass flow rate, \( u_2 \) the steam to the turbine, and \( u_3 \) the feedwater to the drum. All three controls are normalized where for \( u_1 \), 0 - 1 corresponds to 0 - 14 kg/s, for \( u_2 \), 0 - 1 corresponds between closed and open, and for \( u_3 \), 0 - 1 corresponds to 0 - 140 kg/s. The three states \( E \), \( P \), and \( \rho_f \) are electric power in MW, drum steam pressure in kg/cm\(^3\), and fluid density in kg/m\(^3\), respectively.

#### A. Mathematical Model

\[
\frac{dP}{dt} = 0.9u_1 - 0.0018u_2P^{9/8} - 0.15u_3 \tag{1a}
\]
\[
\frac{dE}{dt} = ((0.73u_2 - 0.016)P^{9/8} - E)/10 \tag{1b}
\]
\[
\frac{d\rho_f}{dt} = (14u_3 - (1.1u_2 -0.19)P^{9/8})/85 \tag{1c}
\]

For the purposes of this paper, the steady-state equations are all that are important for deriving the Pareto front:

\[
E_{ss} = \frac{(0.73u_1 - 0.16)(0.9u_1 - 0.15u_3)}{0.0018u_2} \tag{2a}
\]
\[
P_{ss1} = \frac{14u_3}{1.1u_2 - 0.19} \tag{2b}
\]
\[
P_{ss2} = \left[\frac{0.9u_1 -0.15u_3}{0.0018u_2}\right]^{6/9} \tag{2c}
\]
Solving the steady-state equations for $u_1$, $u_2$, and $u_3$, based on pressure limits yields operating windows for a given unit load demand ($uld$). This has previously been done in [6], but the windows are provided in Fig. 2.

B. Neural Network Model

Incorporating any of these techniques will require the mathematical model to be replaced with a neural network or other type of identifier. There are numerous methods that could be used for this, but static neural networks have worked well for reference governors and will continue to be used for generation of the Pareto front. A reference governor uses a neural network that models the steady-state behavior of the power plant. This allows the use of a simple static neural network. Since a mathematical model is available, any amount of training data can be generated. For training data, a large sample of points ($u_1$, $u_2$, and $u_3$) meeting the constraints of the windows in Fig. 2 are used as input training data and (2a) and (2b) are used with the input data to generate the output data ($E_{ss}$ and $P_{ss1}$). This network was trained with 3 inputs to a hidden layer with 15 neurons, and 2 output neurons, using a standard feedforward neural network from Matlab’s neural network toolbox.

V. ANALYSIS OF PARETO FRONT

This simple power plant model makes a very useful introduction to Pareto fronts of power plants because of the relative ease of calculating the exact Pareto front. For this power plant model, there are four variables, $uld$, $u_1$, $u_2$, and $u_3$, three objective functions, and one constraint.

Objectives: For this power plant, three objectives will be minimized for the Pareto front: $\text{Error}$, $u_1$, and $-u_2$. $\text{Error}$ is defined as $|uld - E_{ss}|$ where $uld$ is the unit load demand that the power plant must track. Minimizing $u_1$ reduces fuel intake, and increasing $u_2$ reduces valve losses.

Multi-objective Optimization: The multi-objective optimization function is defined below:

$$x = [uld, u_1, u_2, u_3]$$

(3)

$$f_1(x) = \frac{|uld - (0.73u_1 - 0.16)(0.9u_2 - 0.15u_3)|}{0.0018u_2}$$

(4a)

$$f_2(x) = u_1$$

(4b)

$$f_3(x) = -u_2$$

(4c)

$$\text{minimize } F(x) = [f_1(x), f_2(x), f_3(x)]$$

$$\text{s.t. } \frac{14u_1}{1.1u_2 - 0.19} \left(\frac{0.9u_2 - 0.15u_3}{0.0018u_2}\right)^{8/9} = 0$$

(5)

The Pareto front of this power plant is fairly simple to calculate. First, it should be noted that the control values are bounded between 0 and 1. Over this range and with the constraint from (5), for any unit load demand from 0 to 160 MW, $u_1$ is minimized when $u_2$ is maximized. The math behind this is somewhat long, but easily shown graphically in Fig. 3. This is possible by using (2) to solve for $u_1$ in terms of $u_2$ and $E_{ss}$ as shown in (6).

$$u_1 = \frac{0.0027397E_{ss} - u_2}{u_2 - 0.21918} + ...$$

$$0.0015479(u_2 - 0.1727)\left(\frac{E_{ss}}{u_2 - 0.21918}\right)^{8/9}$$

(6)

It is also important to note that for a given $\text{Error}$, $u_1$ is minimized when $E_{ss} < uld$. This makes sense physically because fuel use is lower for lower power levels. This provides a convenient simplification to finding this Pareto front because for a specified $\text{Error}$, there is only one non-
dominated point. Therefore, using (2), the Pareto front can be calculated analytically for a given $uld$ as a function of $Error$, and is done as follows:

$$E_{ul}^* = uld - Error$$  \hspace{1em} (7)

$$u_2^* = \max(u_2) \quad \text{(Fig. 2)}$$  \hspace{1em} (8)

$$k_1^* = 0.0018 u_2^* \left( \frac{141}{1.1 u_2^* - 0.19} \right)^{9/8}$$  \hspace{1em} (9)

$$k_2^* = \frac{0.0018 E_{ul}^* - u_2^*}{0.73 u_2^* - 0.16}$$  \hspace{1em} (10)

$$u_1^* = \left( \frac{k_2^*}{k_1^*} \right)^{8/9}$$  \hspace{1em} (11)

$$u_2^* = \frac{k^*}{0.15 u_2^*} + 0.9$$  \hspace{1em} (12)

Generating a set of points using this procedure, the Pareto front is visualized for multiple values of $uld$ as shown in Figs. 4 and 5. This is a very simple Pareto front mathematically, but will be very useful for testing the heuristic algorithms and their capabilities before moving onto to larger power plants.

VI. HEURISTIC APPROXIMATION OF THE PARETO FRONT

Now that a neural network has been trained, it is possible to use and evaluate the different approaches. The first approach uses the normal-boundary intersection method, which actually solves multiple single objective functions to generate the Pareto front. This approach is interesting because it is geometrically inspired and does not require any type of Pareto ranking. The second two approaches both solve a single multi-objective optimization problem and use Pareto rankings to drive the search.

A. Normal-Boundary Intersection with Differential Evolution

Normal-boundary intersection (NBI) was developed by Das and Dennis [7] for solving multicriteria optimization problems. It is geometrically inspired and has been very successful in practice. This approach requires the knowledge of the solution to the optimization of the individual $f_i(x)$, which is usually available. In this paper, NBI has been combined with differential evolution (DE) in order to be used with the neural network.

NBI works by solving the following maximization problem.

$$\begin{align*}
\max_t \\
\text{s.t.} \quad \Phi \beta + \hat{n} = F(x) - F^* \\
h(x) &= 0 \\
g(x) &\leq 0
\end{align*}$$  \hspace{1em} (13)

Where $F^* = \{f_1^*, f_2^* \ldots f_n^*\}$, and $f_i^*$ is the solution to the minimization of $f_i(x)$, $x_i^*$ is the corresponding vector that produces that solution, $\Phi$ is defined as $\Phi(:,i) = F(x_i^*) - F^*$. $\beta$ is a vector of $n$ elements whose sum is 1, which is varied to generate the different points across the Pareto front, and (13) must be solved for every $\beta$ that is used. Finally, $\hat{n}$ is the unit normal vector to the plane containing all $x_i^*$, which is easily calculated for the three dimensional space of this problem using (14), and $t$ represents how far $\hat{n}$ can project from $\Phi \beta$ before crossing the Pareto front.

$$\hat{n} = \left( \frac{F(x_i^*) - F(x_j^*)}{\left\| F(x_i^*) - F(x_j^*) \right\|_2} \right) \times \left( \frac{F(x_j^*) - F(x_k^*)}{\left\| F(x_j^*) - F(x_k^*) \right\|_2} \right)$$  \hspace{1em} (14)

In words, NBI works by taking the plane formed by the $x_i^*$, and projecting vector $\hat{n}$ toward the point $F^*$, see Fig 6. Maximizing the value $\hat{n}$ for valid points in the search space naturally chooses points on the Pareto front. For oddly shaped Pareto fronts, this method can fail and converge to local Pareto points instead of global Pareto points.

Since there are multiple vectors that minimize $f_i(x)$ for this problem the following method was used to choose a specific $x_i^*$ for each $f_i^*$. For $f_i(x)$, choose a vector that produces zero error and has $u_2$ close to its maximum value. For $f_2(x)$, put a
bound on allowable error, which was chosen to 9 MW. For \( f_3(x) \), choose a vector with maximum \( u_2 \) which also produces a very low \( \text{Error} \). To enlarge the search space for better results, we need to make sure that \( x_{1i}^* \neq x_{1i}^* \).

DE is a simple type of evolutionary algorithm that uses the differences between population members to generate children, with probability \( CR \), the crossover rate, as in (15).

\[
\Phi \beta_i = u_2 \cdot \Phi i
\]

After every generation, the current particles are combined and then sorted by their Pareto rank to create a new set of non-dominated solutions found. A \( Pbest \) or \( Gbest \) because there is no ‘best’ point in a Pareto optimal set.

This particular implementation of MOPSO works by keeping a list \( A \), of all the non-dominated solutions found. After every generation, the current particles are combined with \( A \) and then sorted by their Pareto rank to create a new set of non-dominated points. \( A \) is then replaced with this new set.

\( Pbest \) is initialized as the particle’s current position, as in standard PSO, and updates to the particle’s new position only if the new point dominates \( Pbest \). \( Gbest \) is dealt with a little differently. Instead of having a single \( Gbest \), every particle chooses randomly from \( A \) which non-dominated point it will use for \( Gbest \), and this choice is made in every generation.

The results of MOPSO are shown in Fig. 9 and 10. This algorithm generates far more points on the Pareto front than normal-boundary intersection. This slows the algorithm down as the number of points on the Pareto front grow, increasing the number of comparisons required by the algorithm. The weighting constant.

To solve this problem, a simple DE algorithm [8] was used with (13) and 21 values of \( \beta \) with equal step sizes as shown in [7]. The DE algorithm uses 200 population members for a maximum of 200 generations, with \( CR = 0.6 \) and \( F = 0.5 \).

The results of using NBI with DE to estimate the Pareto front with a static neural network are provided in Fig. 7 and Fig. 8. Overall, the estimated accuracy of the front for different values of \( \text{uld} \) is fairly accurate, especially for \( u_1 \) and \( \text{Error} \). As can been seen, this is a very clean approach with little noise. The biggest difference in these results to the next two approaches (MOPSO and MOEA) is that only 21 points were generated for each value of unit load demand, which is enough to accurately generate the front.

**B. Multi-objective Particle Swarm Optimization**

Particle swarm optimization (PSO) is one of the most widely spread heuristic optimization techniques based on its simplicity and quick convergence. It has continued to attract interest in the area of multi-objective optimization. There are many different ways of modifying PSO to work with multiple objectives. For this work, the method by Alvarez [9] is used.

PSO uses a number of particles that start with random velocities. These velocities determine how the particles move through the search space, and are updated every generation using (16).

\[
v' = w \cdot v^p + c_1 r_1 (Pbest - x^p) + c_2 r_2 (Gbest - x^p)
\]  

(16)

For (16), \( v \) is the velocity vector, \( w \), \( c_1 \), and \( c_2 \), are weighting constants, and \( r_1 \), and \( r_2 \) are random variables between 0 and 1. For this work, all constants are set to 0.5.

The challenge in using PSO for multi-objective problems is to determine the values for the vectors \( Gbest \) and \( Pbest \). \( Gbest \) is typically the best point currently found from all of the particles and \( Pbest \) the best point found for a specific particle. Since multiple points can be Pareto optimal, it is not immediately apparent which of the non-dominated points should be used for \( Gbest \) or \( Pbest \) because there is no ‘best’ point in a Pareto optimal set.

This particular implementation of MOPSO works by keeping a list \( A \), of all the non-dominated solutions found. After every generation, the current particles are combined with \( A \) and then sorted by their Pareto rank to create a new set of non-dominated points. \( A \) is then replaced with this new set.

\( Pbest \) is initialized as the particle’s current position, as in standard PSO, and updates to the particle’s new position only if the new point dominates \( Pbest \). \( Gbest \) is dealt with a little differently. Instead of having a single \( Gbest \), every particle chooses randomly from \( A \) which non-dominated point it will use for \( Gbest \), and this choice is made in every generation.

The results of MOPSO are shown in Fig. 9 and 10. This algorithm generates far more points on the Pareto front than normal-boundary intersection. This slows the algorithm down as the number of points on the Pareto front grow, increasing the number of comparisons required by the algorithm. The
results are slightly more noisy than that of NBI with DE but still accurate. These results were achieved with only 30 particles over 100 generations.

where \( \beta \) is a weighting constant set to 0.015, \( \omega \) is a weighting constant set to 0.97, \( N_{gen} \) is the current generation number, and \( x_{\text{min}} \) and \( x_{\text{max}} \) are the bounds for dimension \( i \) of \( x \).

The results of MOEP are shown in Fig. 11 and Fig. 12. For these runs, 200 population members were used for 50 generations.

This method differs from MOPSO in two major ways; the first is that new points are generated through mutation and elitism rather than changing a particle’s velocity. The second large difference is that this algorithm uses the Pareto optimal points for its population members, while MOPSO keeps the list \( A \) of Pareto optimal points that have been discovered. Therefore, MOEP always provides a Pareto front with a specific number of points and has the same calculation time for every run, while MOPSO returns a variable number of Pareto points and typically varies between 100 and 160 seconds in calculation time.

C. Multi-objective Evolutionary Programming

Evolutionary Multi-objective evolutionary programming (MOEP) is probably one of the simplest methods of estimating the Pareto front. As detailed in [10], it takes a very simple optimization algorithm employing evolutionary programming with elitism that mutates parents by weighted normal random variables and uses no crossover.

The main difference to single objective optimization with evolutionary programming is that the fitness of each population member is its Pareto rank, the number of points that dominate the specific population member. All Pareto optimal points inherently have a Pareto rank of zero. The population members with the lowest Pareto rank move on to the next generation. This causes the population members to converge toward the Pareto front. The mutation is accomplished using (17) and (18). Equation (18) has been modified slightly from the referenced work.

\[
x^i_j = x^i_j + N(0, \sigma^2_j)
\]  

\[
\sigma_i = \beta \cdot \omega^{N_{gen}} (x_{\text{max}} - x_{\text{min}})
\]  

VII. RESULTS

All of the proposed methods satisfactorily generated the desired Pareto front for the steady state operation of this power plant. NBI-DE while being the slowest, was the most
consistent and most accurate. MOPSO was slightly less accurate, and somewhat faster. However, the MOPSO algorithm can have a limit placed on how many Pareto points it stores to decrease computation time if desired. MOEP was the fastest of the three, but slightly less accurate and the least consistent. Aggregated results for 10 runs are shown in Table I. Error for each Pareto point was calculated by finding the smallest distance between the Pareto point generated by the algorithm and the true Pareto front.

<table>
<thead>
<tr>
<th>Method</th>
<th>Average Time</th>
<th>Average Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBI with DE</td>
<td>145 s</td>
<td>0.2948</td>
</tr>
<tr>
<td>MOPSO</td>
<td>142 s</td>
<td>0.2989</td>
</tr>
<tr>
<td>MOEP</td>
<td>97 s</td>
<td>0.3019</td>
</tr>
</tbody>
</table>

For this application, MOEP would be chosen because it was the fastest, while the accuracy of all three algorithms was comparable. MOEP was also the easiest to implement of all of the algorithms, while NBI with DE was by far the most complicated to implement and required the most tuning of all of the algorithms.

MOPSO however seemed to be a very solid algorithm that could possibly surpass MOEP in computation time if the number of entries in A was limited to decrease the time required to analyze the Pareto optimality of all of the points. If a more sophisticated search algorithm was required, MOPSO definitely has the most opportunities to be modified and further tailored for a specific application.

NBI with DE still may be of use because it can easily focus on a specific area of the Pareto front. Once a desired operating point is known, it may serve as the fastest method for analyzing that specific point. If only one point needed to be analyzed, NBI with DE would take 1/20th the time it currently requires to analyze the whole front.

VIII. CONCLUSIONS

It has been shown that the combination of a static neural network and heuristic Pareto optimization algorithm can adequately generate the operating front of a simple power plant. This work will ideally be used to eliminate previous methods of multi-objective optimization that lump all goals into a single cost function. Further work will be focused on how to determine which section of the Pareto front will be used for operating the power plant based on a human operator’s preferences. This is a simple aspect if the operator is choosing the desired operating conditions for a single unit load demand, but it would be desirable that the operator not have to choose a point for every single unit load demand. It may be possible to have the operator only choose a few points, and interpolate, or perhaps develop a more sophisticated method that can take a single point and extrapolate what the operating conditions should then be for other ulds.

IX. REFERENCES


X. BIOGRAPHIES

Joel H. Van Sickel received his B.S. degree in Electrical Engineering from Grove City College in 2005. He is currently pursuing his Ph.D. in Electrical Engineering at the Pennsylvania State University. His interests include distributed artificial intelligence, distributed simulation, intelligent control, power generation, multi-agent systems and optimization techniques. He is a recipient of the PSU Graduate School Fellowship and the NDSEG Fellowship.
Paramasivam Venkatesh received his B.S.E.E and Masters degree in Power System Engineering with Distinction and Ph.D. in 1991, 1994 and 2003, respectively, from Madurai Kamaraj University, India. His area of interest is application of evolutionary computation techniques to power system problems and power system restructuring. He has received the Boyscast Fellowship award in the year 2006 from Department of Science and Technology, India for carrying out Post Doctoral Research Work at the Pennsylvania State University, U.S.A. Currently he is Assistant Professor in the Department of Electrical and Electronics Engineering, Thiagarajar college of Engineering, Madurai, India.

Kwang Y. Lee received his B.S. degree in Electrical Engineering from Seoul National University, Korea, in 1964, M.S. degree in Electrical Engineering from North Dakota State University, Fargo, in 1968, and Ph.D. degree in System Science from Michigan State University, East Lansing, in 1971. He has been with Michigan State, Oregon State, Univ. of Houston, the Pennsylvania State University, and Baylor University where he is currently a Professor and Chair of Electrical and Computer Engineering. His interests include power system control, operation, planning, and intelligent system applications to power systems. Dr. Lee is a Fellow of IEEE, Associate Editor of IEEE Transactions on Neural Networks, and Editor of IEEE Transactions on Energy Conversion. He is also a registered Professional Engineer.