OPTIMIZATION ALGORITHMS FOR LOADING MILITARY DIESEL GENERATORS

BY

NATHAN PETERSON

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Advisers:

Professor Peter Sauer
Melanie Johnson, Construction Engineering Research Laboratory (CERL)

Abstract

The economic load dispatch (ELD) problem challenges the designer to adequately provide for electrical load demand while minimizing operational costs. The military has a unique set of constraints for meeting the ELD problem to provide power to soldiers in forward operating bases. The constraints include the use of military diesel gensets that remain disconnected from each other and are loaded below a user-defined real power threshold (for a reliability safety cushion). In addition, the system must be simple enough to be constructed with minimal training and require no reconfiguration once established. As a result, a simple tool to quickly assign loads to isolated military diesel generators is required. To meet this need, this study compares the use of several optimization algorithms including particle swarm optimization (PSO), bat algorithm (BA), cuckoo search (CS), first fit decreasing (FFD) bin packing, and an exhaustive search (ES) method. It is found that at large enough search spaces, the optimization algorithms can discover reasonably optimal solutions while substantially decreasing search time. For this application, FFD has more optimal average solutions as well as faster run time compared to the other algorithms.

Acknowledgments

I would like to express my gratitude to Professor Peter Sauer and Mrs. Melanie Johnson for their mentorship, as well as my sincere appreciation to Dawith Ha and Tom Lee for their coding support.

Contents

1. Introduction

When it comes to supplying electrical power to soldiers in the field where grid power is unavailable, the United States military nearly always uses diesel generators. However, due to training constraints, the soldiers in charge of electrical equipment are typically given just weeks of training to learn the broad field of electrical engineering. In addition, when lives are at stake, soldiers are forced to treat reliability as the top priority. However, this leads to drastically oversized generators, wet stacking problems, and very inefficient systems. This occurs at a time when carbon emissions should be limited, when the price of fuel in the field can be as high as $500 per gallon according to former Commandant of the Marine Corps, Gen. James T. Conway, and when the price in lives lost can be even greater. According to a 2009 Deloitte report [1], in Afghanistan there is about one casualty in every 24 resupply convoys resulting in a linear increase in casualties in relation to the ever-increasing demand of fuel as can be seen in Figure 1. As a result, a simple-to-use and effective method for soldiers to calculate how to load military diesel generators can save energy, money, and lives.

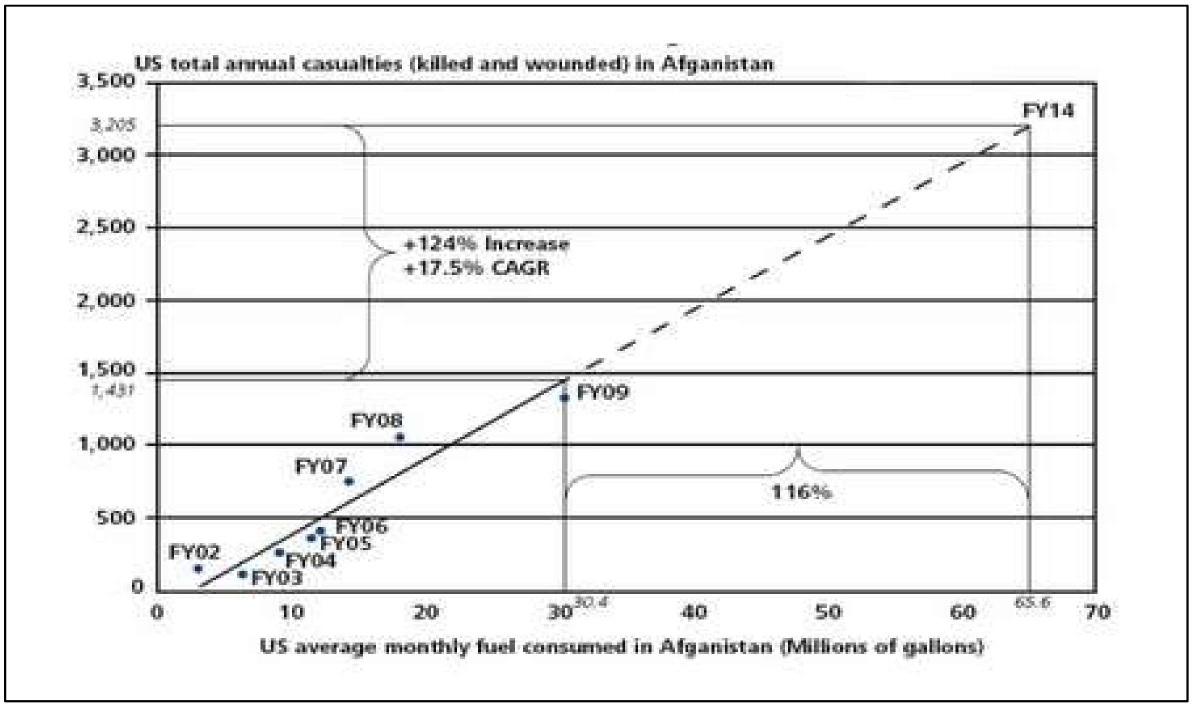

Figure 1: US Annual Casualties in Afghanistan vs. Average Monthly Fuel Consumption

The resulting puzzle to solve is the economic load dispatch (ELD) problem. The ELD problem challenges the user to optimize the output of available generators such that the electric load is met while minimizing financial costs and adhering to system limitations. In simple terms, the goal is to minimize

$$F_T = \sum_{i=1}^{n} F_i(P_i) \tag{1.1}$$

subject to

$$0 \leq P_i \leq P_i^{max} \tag{1.2}$$

$$P_D = \sum_{i=1}^{n} P_i \tag{1.3}$$

where $F_i(P_i)$ is the operation cost of unit i when its output power is P_i, n is the number of diesel generator units, P_i^{max} is the maximum power output power for unit i, and P_D is the total power demand. According to Equation (1.4), the operation cost of unit i can be further defined as

$$F_i(P_i) = (\text{fuel price}) \sum_{t=0}^{T} \text{hourly consumption(loading)} \tag{1.4}$$

where fuel consumption, as modeled by a second-order polynomial for most diesel generators, is defined in Equation (1.5) as

$$C_i(P_i) = \ a_i + b_i P_i + c_i P_i^2 \tag{1.5}$$

where a_i, b_i, and c_i are non-negative constants of the *i*-th generating unit.

The limitations for this application require the use of military diesel gensets that remain disconnected from each other and remain loaded below a user-defined real power threshold (for a reliability safety cushion). In addition, the system must be simple enough to be constructed with minimal training and require no reconfiguration once established. It is assumed the loads will closely follow a predictable load schedule. Since fuel prices dominate costs in this application, transmission and distribution costs, ramp rate limitations, maintenance costs, and emission costs are not considered for this model, but it is likely that maintenance costs and emission costs will be lower than current methods and can be added in further studies.

There are many different optimization algorithms to consider when resolving the ELD problem. Direct method algorithms used for searching every possible point in the search space are far too computationally and time intense to be considered outside of a very limited scope. A common second class of optimization algorithms includes gradient methods like gradient descent, the Newton-Raphson method, and the Frank-Wolfe algorithm, which use the second order derivative of the convex search space to find minima. These methods are fast and scale well, but require smooth, differentiable gradients and are liable to merely find local minima. Due to the nature of this application working with discrete generators, the ELD problem search space is nonlinear, discontinuous, and nondifferentiable. Furthermore, this problem is classified as NP-hard [2], meaning it cannot be solved in polynomial time, and therefore the time required to solve the problem increases exponentially with problem size. To solve this problem at any useful scale, a more sophisticated method must be taken.

Heuristic-based approaches are a good alternative. Heuristics are problem-specific techniques used to quickly determine good solutions when other methods are too slow or imprecise. One such heuristic category comprises bin packing algorithms, used and explained later in this study. Another good alternative is metaheuristics, which are problem-independent techniques that sample a set of solutions when the search space is too large to sample completely. Many popular and successful metaheuristic algorithms mimic behaviors found in nature, like the ant colony optimization algorithm. Like heuristic methods, these approaches may not deliver the global minimum, but if appropriately selected and calibrated they can provide reasonably good and fast solutions that are otherwise infeasible.

This study includes a literature review of current evaluation methods and the comparison of five different optimization algorithms: particle swarm optimization (PSO), bat algorithm (BA), cuckoo search (CS), first fit decreasing bin packing (FFD), and an exhaustive search (ES) method.

2. Literature Review

The ELD problem has tested many different classical optimization techniques. These methods include the Newton method [3], the lambda iteration method (LIM) [3]–[8], the gradient method [9], and the gradient projection algorithm (GPA) [10]. These methods typically have two limitations. The first restriction is their dependence on monotonically increasing cost functions possessing derivatives. The ELD problem typically possess a nonconvex, nonlinear, and sometimes discontinuous mathematical search space due to added real world complexities like transmission losses, valve point loading effects (VPL), prohibited operating zones (POZ), multiple fuel options (MF options) and, in this application's case, discrete disconnected diesel generators. This leads to the high probability of falling into a local minimum instead of the global optimum solution. In addition, these methods suffer the "curse of dimensionality" and become exponentially complex to calculate with the addition of multiple dimensions as is common in the ELD problem. As a result, these traditional methods were found inadequate for the ELD problem beyond very limited and rudimentary applications. The one alternative proposed solution [11] that at least solves the first limitation requires decomposing the search space into differentiable regions before applying these techniques. Nonetheless, this requires knowledge of the search space and extra preprocessing while still being limited by low dimensionality limitations.

As a result, many metaheuristic techniques are used in the literature to solve the nonlinear and dimensionality issues of the ELD problem including the PSO algorithm [4], [5], [7], [8], [12]–[37]; the genetic algorithm (GA) [4], [5], [29], [31], [38], [39], [7], [8], [12], [18], [20], [24], [26], [28]; the CS algorithm [9], [25], [27], [40]–[46]; the BA [13]–[18], [31], [32], [40], [41], simulated annealing (SA) [8], [35], [49]; ant colony optimization (ACO) [6], [38], [39]; firefly algorithm (FA) [24], [27], [43], [47]; artificial bee colony algorithm (ABC) [5], [28], [47]; bacterial foraging algorithm (BFA) [21], [50], [51]; gravitational search algorithm (GSA) [7], [8], [15], [28]–[31], [49]; and many others. Most of these algorithms are inspired by nature. To add complexity and tunability, many of these algorithms are enhanced or combined with each other to make entirely new algorithms with different characteristics. To avoid the rabbit hole of near infinite combinations and potential tweaks of each algorithm, this study will first look at the standard version of PSO, CS, and BA for this particular application of the ELD problem. Upon completion of this study, one can use the results to further narrow their focus in search of useful algorithm augmentations.

PSO is a jack-of-all-trades algorithm that has been around longer than most swarm-based metaheuristics and has proven robust and effective in a very large variety of fields. In the ELD realm, it has been employed in a wide assortment applications from thermal generators and cogeneration plants [4], to emissions dispatch [14], [28], [32] and uncertain wind source renewables [22], with numerous constraints like transmission losses, VPL, POZ, and MF options. Moreover, it works well from at least three generators [4] to at least 320 [26]. PSO has repeatedly beaten GA in optimization and completion time in comparison tests [4], [18], [14], [20], [24], [26], [39]–[41] as well as FA [24], [43], SA [35], ABC [43], BFA [51], and machine learning techniques [16], [36] under various test conditions. Nevertheless, PSO is repeatedly beaten by BA [19]–[23], [39] and CS [26], [27], [40], [41], [43] algorithms. One study [16] suggests that PSO has trouble with higher dimensionalities, while another [17] demonstrates better capabilities by using a Lévy distribution similar to the CS algorithm, and another [13] illustrates a superior PSO variation that uses constriction factors to assist PSO in local search. The evolutionary particle swarm optimization (EPSO) algorithm [32] is proposed to augment PSO by adding evolutionary programming to enhance its search capability. As a result of combination, tournament, sorting, and election processes, EPSO was found to achieve faster convergence, lower fuel costs, and fewer emissions. While it may not be optimal in its basic form, PSO can be seen as an old reliable method and frequently serves as a benchmark for competition.

CS has likewise demonstrated robustness in ELD application. It has been used in a combination of gensets, wind power, solar power, and battery storage microgrid systems [9]; it has beaten PSO in 320 unit systems [26]; it beats competitors in a 40 genset 10500 MW simulation test [43]; and it has done well calculating electric vehicle and wind farm variables [46]. Like the other metaheuristics considered, it consistently outperforms GA [26], [40], [41], [44] and PSO/PSO variations [26], [27], [40], [41], [43], while also beating FA [27], [43], GSA [29], [30], and ABC [43]. In the literature, no algorithm was found to outperform CS, but one study [25] suggests that its performance may start to fall off on larger systems compared to PSO because its “nests” must be changed and recalculated every iteration. Nonetheless, it outperforms PSO in generator systems as big as 320 units [26].

The literature also shows BA’s versatility and superiority over GA [20], [24], [39] and PSO [19]–[23], [39], as well as FA [24], ABC [21], and machine learning techniques [39]. Unfortunately, no literature was found directly comparing the BA and CS. There are multiple augmentations to BA, like the novel bat algorithm (NBA) [19], [21] which compensates for the

Doppler effect and allows bats to hunt in wider habitats than normal BA; the enhanced bat algorithm (EBA) [20] which adjusts the velocity of the bats, similar to PSO, for better local search capabilities; and chaotic bat algorithm (CBA) [23] which tunes the emission pulse rates of the bats by a sinusoidal map to improve convergence consistency over normal BA by one order of magnitude and over PSO by two orders of magnitude. These reasons and many others make BA very difficult to surpass.

Other metaheuristics were brought up in the literature, but were generally difficult to find and performed worse than the previously discussed algorithms. The GA was beaten by virtually every metaheuristic contender. FA was likewise beaten by everything except for GA [24] and PSO [27] once, and was only able to beat BA and ABC when given seven times the amount of completion time [47]. Unlike others, FA does not automatically adjust its tuning parameters as the algorithm progresses, leading to poorer performance. As a result, FA is a poor contender. The few instances of SA were beaten by PSO [35] and GSA [8], [49] and typically demonstrated very slow completion periods. Not much is found regarding ABC in regard to the ELD problem, except that it can outperform GA [5] like the other metaheuristics but both PSO and CS are shown to outdo it [43]. BFA beats GA [51], but both beats [21] and loses to PSO [51], so it is not a strong candidate. This is likely due to its biased random walk, which is inadequate to search the large multidimensional search space of the ELD problem. With the limited literature on ACO in ELD problems, it was found too inconsistent to consider here. While ACO's convergence is guaranteed, the time it takes is somewhat uncertain. It tops PSO [5], [28] and GA [5] as well as BA once [47] and GSA [28], but in other literature it also loses to PSO [43], CS [43], and GSA [49]. GSA shows promising characteristics similar to those of other algorithms chosen for this study because it is more common in ELD literature and it consistently outstrips PSO [7], [8], [15], [28], [30], [31], [49] and GSA [7], [8], [29], [31], [49]. However, since it loses to CS [29], [30], it was not deemed a significant candidate. A comparison of all of the studied metaheuristic candidates is in Table 1.

Table 1: Algorithm Comparison, Winners (Left) vs. Losers (Top)

	PSO	**GA**	**CS**	**BA**	**SA**	**GSA**	**BFA**	**FA**	**ABC**	**ACO**
PSO	**X**	[4], [18], [14], [20], [24], [26], [39], [40], [41]		[24]	[35]		[51]	[24], [43]	[43]	
GA		**X**						[24]		
CS	[26], [27], [29], [40], [41], [43]	[26], [29], [40], [41], [44]	**X**			[29], [30]		[27], [43]	[43]	
BA	[19], [20], [21], [22], [23], [39]	[20], [24], [39]		**X**			[21]	[24]		
SA					**X**					
GSA	[7], [8], [15], [28], [30], [49], [31]	[7], [8], [29], [49], [31],			[8], [49]	**X**			[49]	
BFA	[21]	[51]					**X**			
FA	[27]			[47]*				**X**	[47]*	
ABC	[5], [28]	[5]		[47]		[28]			**X**	
ACO	[39]	[39]								**X**

*Trial allowed to run 7 times as long as competing algorithms

Due to the nature of this study's application, another appropriate algorithm investigated by this study falls under the category of a bin packing heuristic algorithm. This algorithm is related because, in this scenario, electrical loads must be assigned to individual, disconnected generators. Since diesel generator efficiencies typically increase with the electrical loading, using the minimum number of generators, or bins, assures the maximum net fuel economy of the system, assuming each generator has the same efficiency versus loading curve. Since this is a unique set of requirements, there are few references to it in ELD problem literature. One exception [52] studies the strip packing problem, a close relation to the bin packing problem, in scheduling different priority loads in a smart electrical grid. While scheduling loads does not provide a solution robust enough to satisfy the requirements of the military at this time, this research demonstrates the usefulness of the FFD bin packing algorithm in this field. Another review of bin packing algorithms [53] has demonstrated that the FFD algorithm had the most optimal solution among six bin packing algorithms including max rest, first fit, next fit, next fit decreasing, and best fit across 11 different benchmarking data sets. This finding is consistent with another source [54] that was unable to determine a better algorithm than the first fit decreasing algorithm. Even in the two-dimensional bin packing application of the problem [55], the FFD family of algorithms was the top contender among next fit, first fit, split fit, floor-ceiling no rotation, knapsack, and size alternating stack (SAS) algorithms.

As a result of the above literature search, the PSO algorithm, the BA, the CS algorithm, and the FFD bin packing algorithm were estimated to demonstrate strong solutions to the ELD problem. Therefore, their performance will be compared for the application of the problem addressed in this thesis.

3. Description of Research

This research will first deeply investigate each of the five algorithms selected, followed by a description of the testing procedure, and finally a comparison of the testing results.

3.1 The Algorithms

3.1.1 The Particle Swarm Optimization Algorithm

Designed by James Kennedy and Russel Eberhart in 1995 [56], the PSO algorithm is a biologically inspired, stochastic, metaheuristic swarm algorithm that mimics the social and individual dynamic movements of many birds, insects, and fish. In this algorithm, a swarm consists of agents, called particles, that move around the mathematical search space probing for the best solution. Each particle adjusts its position in the search space based on its own experience as well as the experience of the swarm. A particle's new position and velocity can be given in Equation (3.1) and Equation (3.2) as

$$x_{k+1}^{i} = x_{k}^{i} + v_{k+1}^{i} \tag{3.1}$$

$$v_{k+1}^{i} = w_k v_{k+1}^{i} + c_1 r_1 \left(p_k^i - x_k^i\right) + c_2 r_r \left(p_k^g - x_k^i\right) \tag{3.2}$$

where x_{k+1}^{i} is the particle's new position, x_k^i is the particle's old position, v_{k+1}^{i} is the particle's new velocity, v_k^i is a particle's old velocity, p_k^i is the particle's individual best position, p_k^g is the particle swarm's global best position, w_k is an inertial weight constant, c_1 is weighting constant for local personal information, c_2 is a weighting constant for global social information, and r_1 and r_2 are random numbers between zero and one. If this equation is improperly tuned, a low velocity will create a slow algorithm, but too high will produce instability. The sum of c_1 and c_2

is usually empirically equal to four, and the total number of particles is usually chosen to be between 10 and 50. As can be seen from the velocity equation, a particle's velocity is controlled by three terms where the first is its inertia, the second is its personal influence, and the third is its swarm influence. The way these three forces affect particle velocity and position is visualized in Figure 2. Using the tip to tail method of vector addition, it is easy to determine the resulting velocity for the particle. Viewing the velocity equation in a different way, the first term is applied for diversification, meaning to search for new solutions, while the second and third terms are applied for intensification, or to better explore known solutions within a given region.

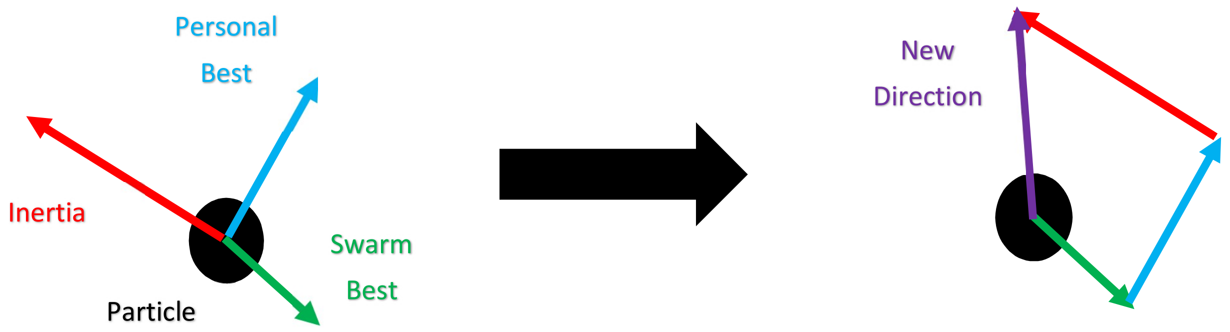

Figure 2: Particle Swarm Optimization Force Vectors

The algorithm starts by setting w_k, c_1, c_2, and the maximum number of iterations, k, as well as randomly initializing the particles in the search space with a position and velocity. Next, the fitness of each particle is evaluated according to the cost function, f_k^i, at each position, x_k^i. As defined in Equation (3.3), if the resulting function is better than the particle's previous best, then its result and position are recorded as a personal best.

$$f_k^i \leq f_{best}^i \begin{cases} \text{true}, & f_{best}^i = f_k^i \text{ and } p_k^i = x_k^i \\ false, & f_{best}^i = f_{best}^i \text{ and } p_k^i = p_k^i \end{cases} \tag{3.3}$$

Similarly, if the resulting function is better than the swarm's previous best, then its result and position are recorded as a swarm best according to Equation (3.4).

$$f_k^i \leq f_{best}^g \begin{cases} \text{true}, & f_{best}^g = f_k^i \text{ and } p_k^g = x_k^i \\ false, & f_{best}^g = f_{best}^g \text{ and } p_k^g = p_k^g \end{cases} \tag{3.4}$$

Next, the algorithm checks if the maximum number of iterations have been met. If so, the solutions are stored in f_{best}^g and p_k^g. If the maximum iterations have not been met, the velocity of each particle is calculated and the positions and velocities are updated according to Equation (3.1) and Equation (3.2) respectively, and the cycle starts again evaluating each particle's new fitness. This overall process is visualized in Figure 3.

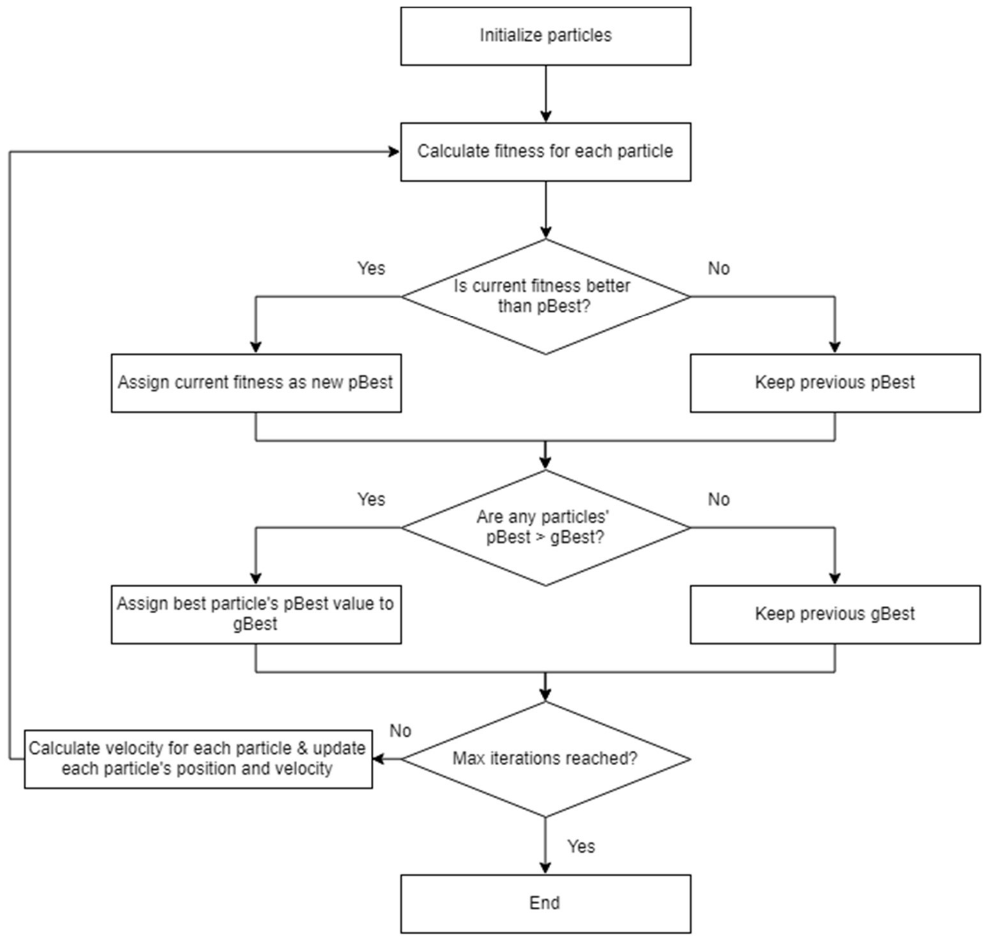
Initialize particles
Calculate fitness for each particle
Yes
Is current fitness better than pBest?
No
Assign current fitness as new pBest
Keep previous pBest
Yes
Are any particles' pBest > gBest?
No
Assign best particle's pBest value to gBest
Keep previous gBest
Calculate velocity for each particle & update each particle's position and velocity
No
Max iterations reached?
Yes
End

Figure 3: Particle Search Optimization Algorithm Flow Diagram

While it is better than many other algorithms, PSO can still converge on local minima instead of the global minimum. It quickly localizes minima but possesses weak local search capabilities to quickly find the local absolute minimum. In addition, it has high computational complexity. Nonetheless, PSO offers many benefits. The search space does not need to be differentiable like many other iterative methods like gradient descent, and it can solve high dimensional, nonconvex, and discontinuous problems. It is a simple, efficient global search algorithm compared to traditional optimization algorithms, and it makes no assumptions about the problem, which is why it is widely used in a large range of fields and applications, including ELD.

3.1.2 The Cuckoo Search Algorithm

The CS algorithm is a nature-inspired stochastic metaheuristic algorithm developed by Xin-She Yang in 2009 [57] that mimics brood parasitism of some cuckoo birds wherein they lay their eggs in the nest of another species. Cuckoo bird eggs are either discarded by a wary mother bird, or they are nurtured until they hatch and frequently dominate resources from the hatchlings of the parent species. In this algorithm, the nests represent potential solutions, and an egg in a nest represents a new solution. The goal of the algorithm is to replace the old worse nests/solutions with new and better nests/solutions.

There are three assumptions taken in the CS algorithm:

1. Every cuckoo lays exactly one egg in one nest per iteration.
2. Nests with better quality eggs are carried on to the next round.

3. The number of nests is fixed and not alterable, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in [0,1]$.

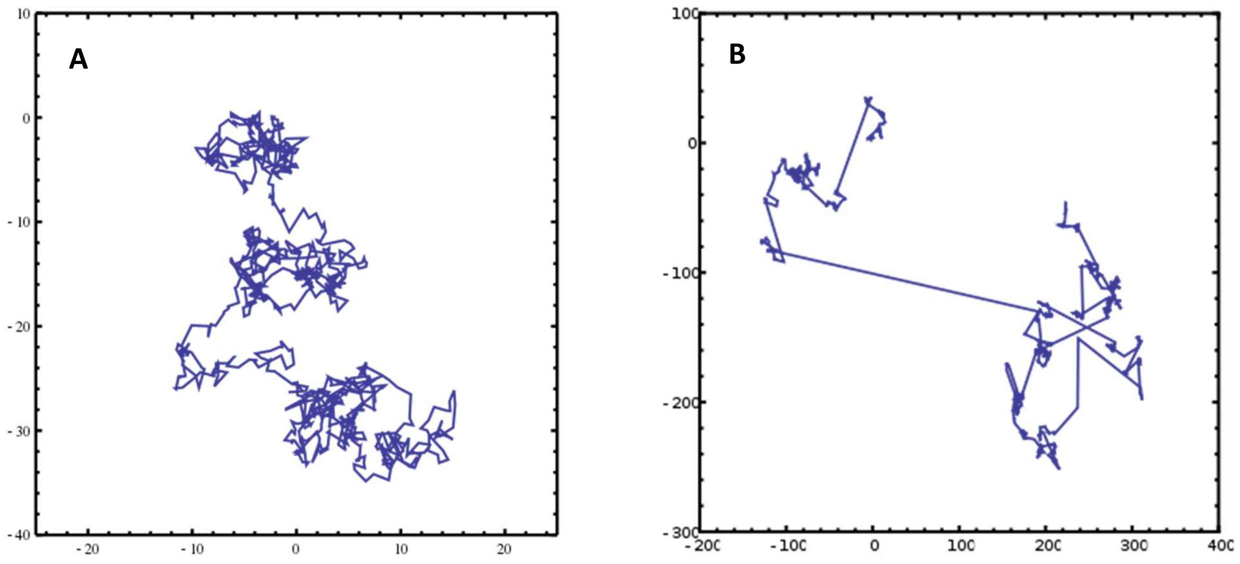

Figure 4: (A) Brownian Walk Versus (B) Lévy Flight

To discover a new nest, a random walk is made via Lévy flight as opposed to the usual Brownian walk, as it has a larger step length in the long term, allowing for more efficient exploration of the global search space, as seen in Figure 4. Special note should be taken of the much wider axis scales for Lévy flight in Figure 4B. The new Lévy flight step is calculated via Equation (3.5):

$$x_i^{t+1} = x_i^t + \alpha \oplus \text{Lévy}(\lambda) \tag{3.5}$$

where α is the step size and α > 0. It is common for α = 1. The ⊕ means entry-wise multiplication, or the Exclusive OR operation. The Lévy flight step length is taken from a Lévy distribution in Equation (3.6):

$$\text{Lévy} \sim u = t^{-\lambda}, \qquad (1 < \lambda < 3) \tag{3.6}$$

which has an infinite variance with an infinite mean.

The algorithm starts by randomly initializing n host nests and evaluating the fitness of each potential solution. Only if the new solution is better than the previous global best solution, is it recorded. Next, a fraction of the worst nests is replaced according to probability p_a, and if the maximum generations have not been met yet, the process starts again. This process can be seen in Figure 5.

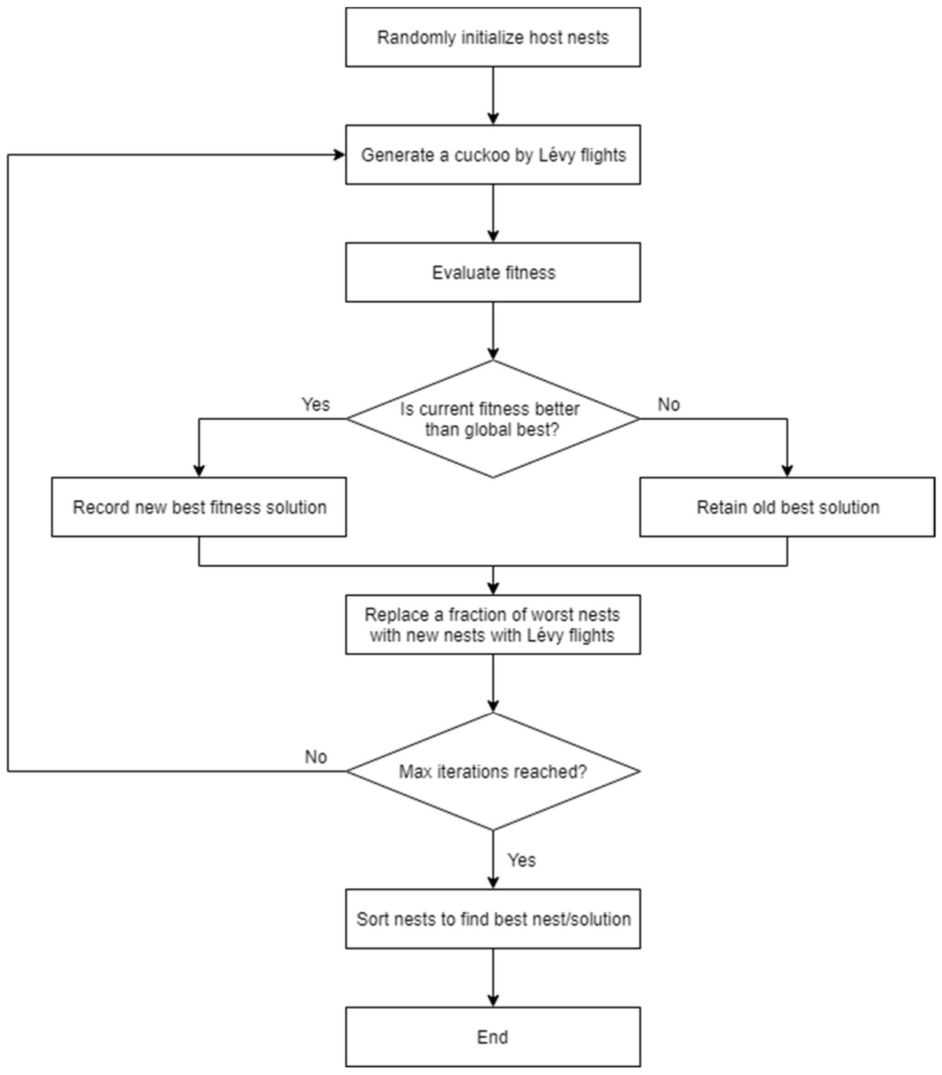
Randomly initialize host nests
Generate a cuckoo by Lévy flights
Evaluate fitness
Is current fitness better than global best?
Yes
No
Record new best fitness solution
Retain old best solution
Replace a fraction of worst nests with new nests with Lévy flights
Max iterations reached?
No
Yes
Sort nests to find best nest/solution
End

Figure 5: Cuckoo Search Algorithm Flow Diagram

The benefits of this algorithm are many. Like the other metaheuristics evaluated, it can relatively quickly solve problems that are nonlinear, discontinuous, and nonconvex. Assuming α=1 in most cases, it has only two tuning constants, probability p_a and number of host nests n, making it the simplest option considered, allowing it to be very generic for many problems. Nonetheless, this may be problematic if more control or optimization is desired. This algorithm has strong global convergence capabilities proven by Markovian probability theory. Exploration is rapid and very random due to Lévy flights, ensuring a strong global search capacity, but like many other metaheuristics, it may have poor local search capabilities close to the optimal solution.

3.1.3 The Bat Algorithm

The BA is another biologically inspired stochastic metaheuristic algorithm developed by Xin-She Yang in 2010 [58] which is based on the echolocation of microbats to navigate their surroundings and hunt prey as shown in Figure 6. Similarly, in this algorithm, bats fly around the search space in search for potential solutions.

There are three idealistic rules of the BA [58]:

1. "All bats use echolocation to sense distance, and they know the difference between food/prey and background barriers in some magical way.
2. Bats fly randomly with a velocity v_i and position x_i with a frequency f_{min}, varying wavelength λ and loudness A_0 to search for prey. They can automatically adjust the

wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission $r \in [0,1]$, depending on the proximity to their target.

3. Although the loudness can vary in many ways, we assume that the loudness varies from a large (positive) A_0 to a minimum constant value A_{min}. "

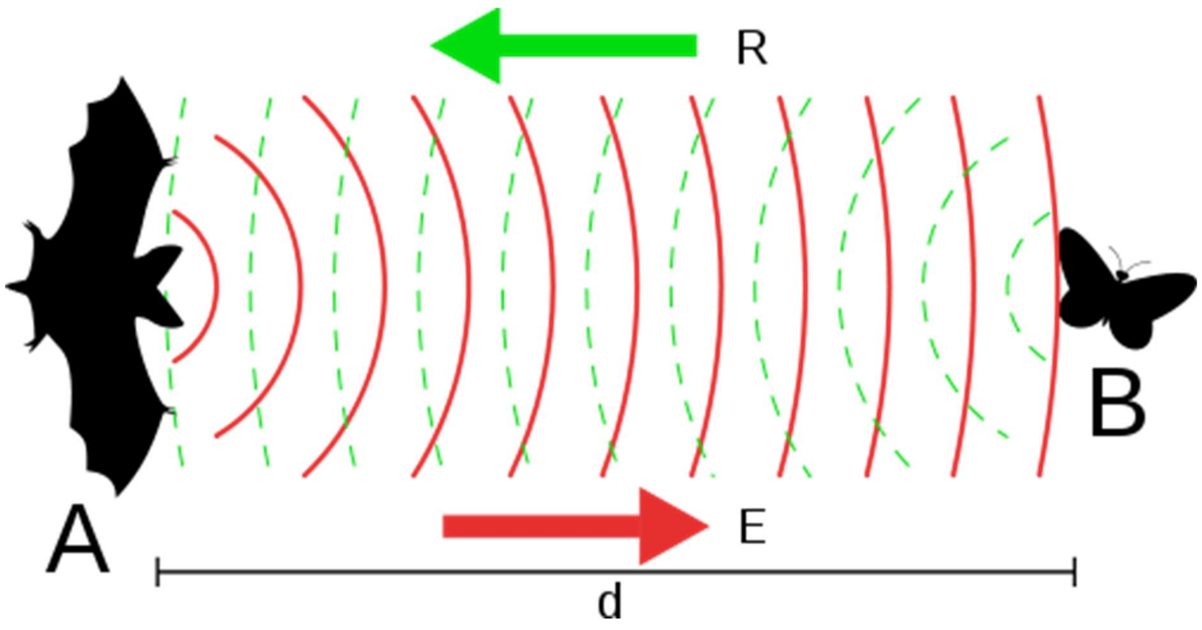

Figure 6: Bat Echolocation: (A) Bat, (B) Prey, (d) Distance, (E) Emitted Wave of Bat, (R) Reflected Wave of Prey

As is the case with real bats, the virtual bats in the search space decrease their loudness when getting close to their prey to focus on their target, and increase the rate of pulse emissions for higher tracking fidelity. In the algorithm, after initializing the bats with their positions, flying velocities, frequency bounds, pulse rate, and loudness, their fitness is calculated and a new solution is generated. New solutions are generated via Equation (3.7), Equation (3.8), and Equation (3.9):

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{3.7}$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x_*)f_i \tag{3.8}$$

$$x_i^t = x_i^{t-1} + v_i^t \tag{3.9}$$

where $\beta \in [0,1]$ is a random vector drawn from a uniform distribution and x_* is the current best global location. If the new position is better than the old one, it is recorded, and if the number of iterations is not maxed out, a new solution is generated again. If it is not a better solution and a random number is generated between zero and one and compared to the current pulse emission rate. If the random number is less than the pulse rate, the iterations are checked and a new solution is generated. If the random number is greater than the pulse rated, the best solution is selected and a local best solution is generated around that point. If the bat's loudness is greater than the random number and the newly generated point is better than the previous, the new solution is accepted, the emission rate is increased, and the loudness is decreased before checking the iterations and starting the whole cycle again. As time goes on, the pulse rate and loudness of the bats follow Equation (3.10) and Equation (3.11):

$$r_i^{t+1} = r_i^0[1 - \exp(-\gamma t)] \tag{3.10}$$

$$A_i^{t+1} = \alpha A_i^t \tag{3.11}$$

where α and γ are user-defined tuning constants. Usually, these are defined in the ranges 0<α<1 and γ>0, such that both loudness decays and pulse rate progress exponentially according to Equation (3.12):

$$A_i^t \to 0, \qquad r_i^t \to r_i^0, \qquad as\ t \to \infty \tag{3.12}$$

The flow diagram for the BA is in Figure 7.

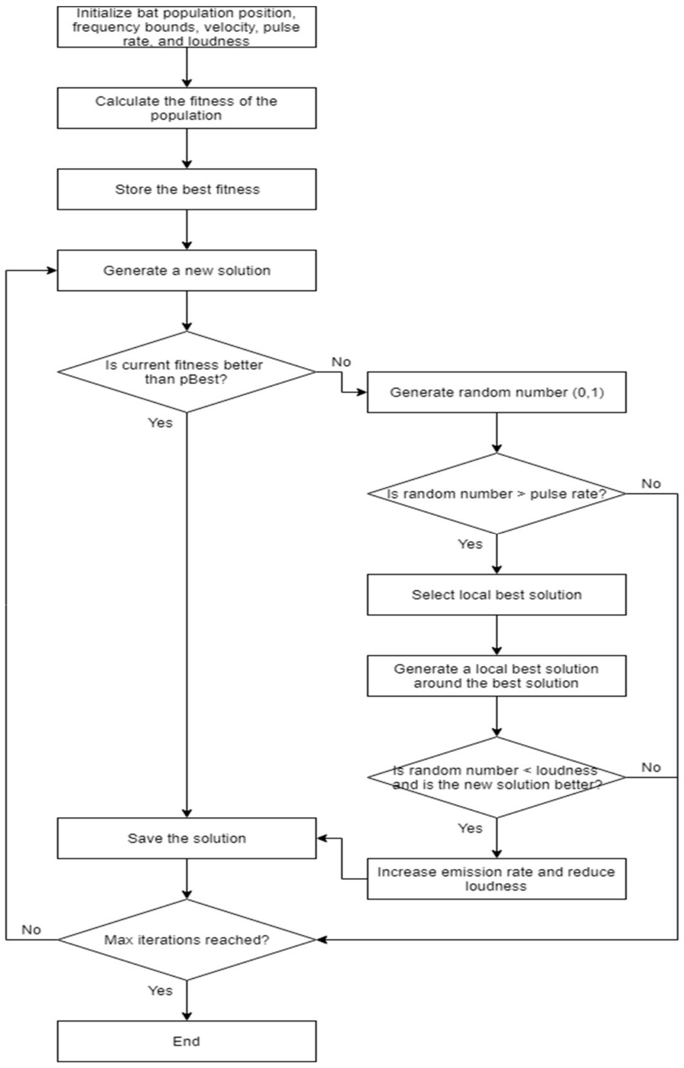
Initialize bat population position, frequency bounds, velocity, pulse rate, and loudness
Calculate the fitness of the population
Store the best fitness
Generate a new solution
Is current fitness better than pBest?
No
Yes
Generate random number (0,1)
Is random number > pulse rate?
No
Yes
Select local best solution
Generate a local best solution around the best solution
Is random number < loudness and is the new solution better?
No
Yes
Save the solution
Increase emission rate and reduce loudness
Max iterations reached?
No
Yes
End

Figure 7: Bat Algorithm Flow Diagram

The BA has many advantages. Like many other metaheuristics, it can relatively quickly solve problems that are nonlinear, discontinuous, and nonconvex in a simple, flexible, and easy-to-implement way. The innovative feature of the BA is its ability to both search globally, but also automatically zoom in on a search space region that looks promising by naturally adjusting loudness and emission rate to switch its function from exploration to exploitation as time goes on. As a result, BA typically has a very quick start. However, it can tend to converge quickly at first, then slowdown in later stages. In addition, because it is so new and not well tested yet, it is not clear what generic tuning parameters should be used for best results.

3.1.4 The First Fit Decreasing Bin Packing Algorithm

While the previous sections have described metaheuristics, this section expounds on an application specific heuristic. The military diesel generator loading application has stark similarities to the bin packing problem. In the bin packing problem, the objective is to pack boxes of a certain weight/size into the least number of bins possible without exceeding the capacity of each bin. In the case of this application, the electrical loads (boxes) are loaded (packed) into the fewest generators (bins) possible without exceeding the output power capacity (size) of each generator (bin). Since diesel generator fuel efficiencies typically increase with the electrical loading, using the minimum number of generators assures the maximum net fuel economy of the system, assuming each generator has the same efficiency-versus-loading curve. Note that while the electrical loads are treated as one-dimensional, the electric loads in this scenario are actually two-dimensional due to the use of load profiles over a given time. When attempting to fill a generator with two or more loads, their 2D load profiles are added on top of each other, with the overall peak power consumption during the time period being treated as the 1D size of the load box to be fit into the 1D generator bin.

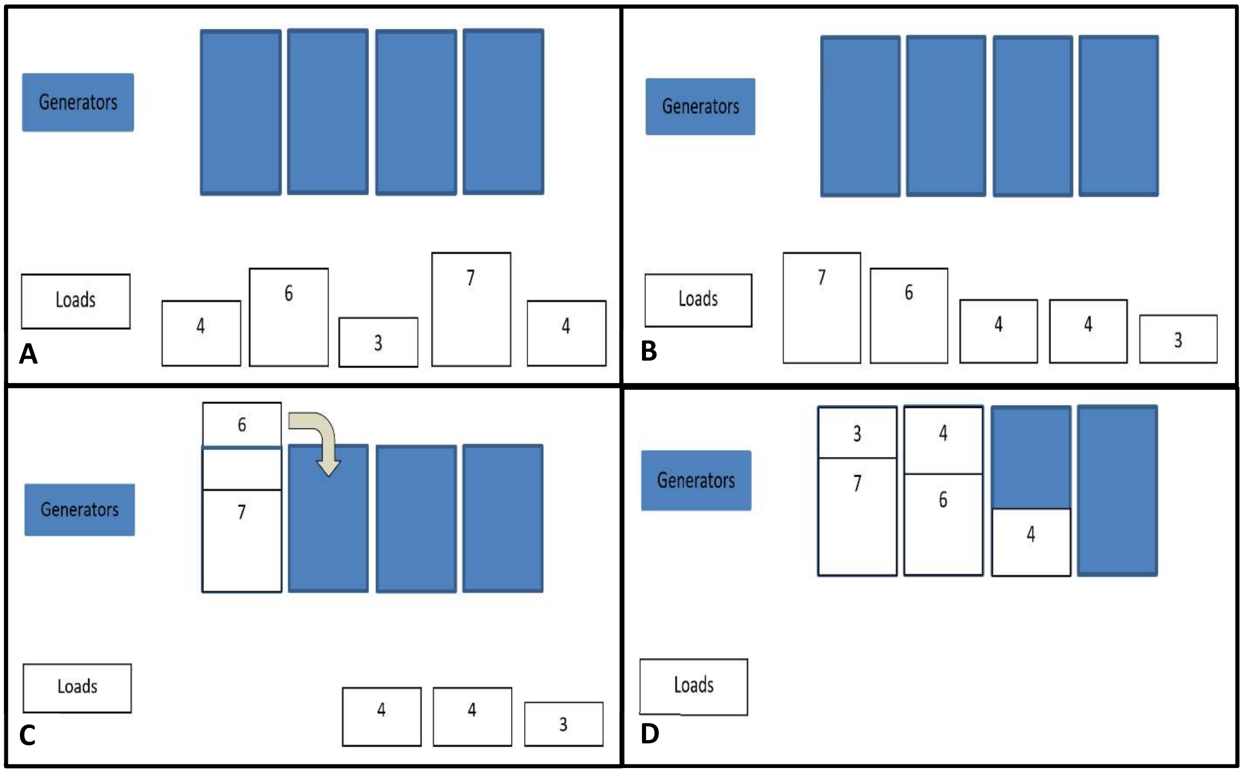

Figure 8: First Fit Decreasing Flow Diagram: (A) Randomly Assorted Loads, (B) Sorted Loads from Biggest to Smallest, (C) Second Load Is Too Big to Fit in First Generator So It Is Placed in the Second, (D) Final Load Assignment

The FFD bin packing algorithm [54] is a greedy algorithm which arranges the electrical loads in decreasing peak power size, and then attempts to fit them into generators from the highest to lowest peak efficiency, as seen in Figure 8A and Figure 8B. If the first generator cannot pack the load without exceeding its maximum threshold, as in Figure 8C, the algorithm tries to pack the load into the next generator on the list until all the electrical loads are packed into generators, as shown in Figure 8D. If all loads fit within the designated generators, a resulting fitness is calculated. If they do not fit, or if there are no available generators remaining, the result is a nonsolution. These steps can be seen conceptually in Figure 9.

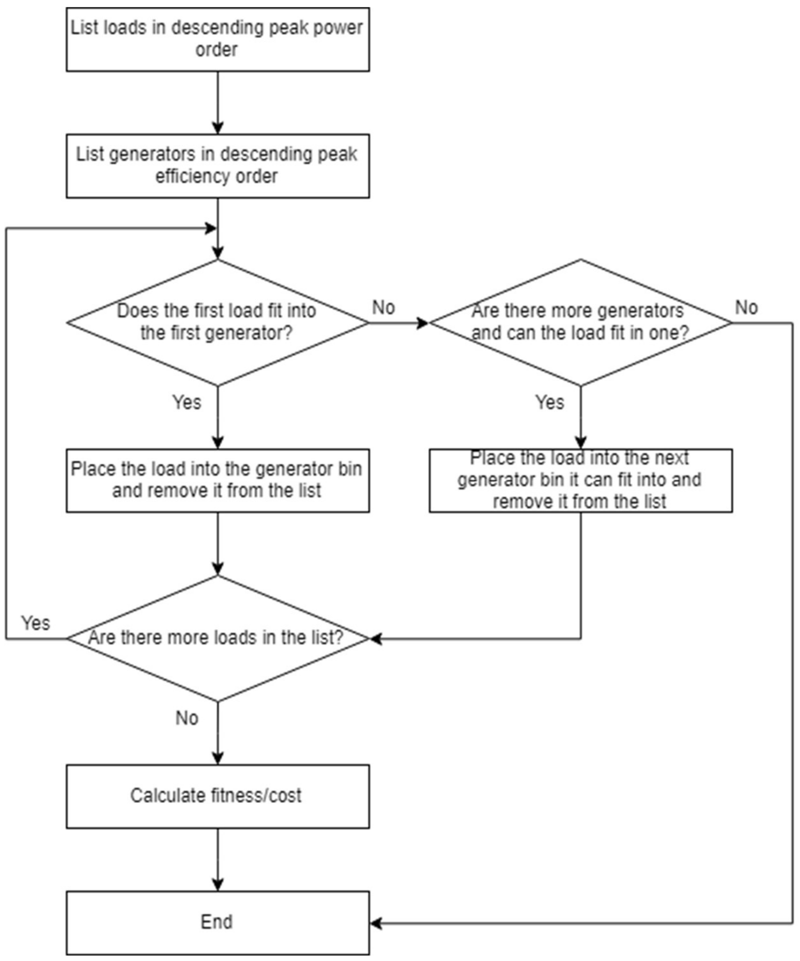
List loads in descending peak power order
List generators in descending peak efficiency order
Does the first load fit into the first generator?
No
Are there more generators and can the load fit in one?
No
Yes
Yes
Place the load into the generator bin and remove it from the list
Place the load into the next generator bin it can fit into and remove it from the list
Yes
Are there more loads in the list?
No
Calculate fitness/cost
End

Figure 9: First Fit Decreasing Bin Packing Algorithm Flow Diagram

While this algorithm is simpler than the other metaheuristics, its additional assumptions/abstractions suggest it may produce a comparable optimum cost in a small fraction of the time since it does not require the computation of a large number of fitness functions every iteration cycle like the previous algorithms. While there are many other bin packing algorithms like max rest (MR), first fit (FF), next fit (NF), next fit decreasing (NFD), and best fit (BF), FFD largely seems the most optimal according to many sources like [53] where FFD had the most optimal solution among six bin packing algorithms across 11 different benchmarking data sets.

3.1.5 The Exhaustive Search Algorithm

The ES algorithm is a rudimentary algorithm that simply iterates through every possible combination of electrical loads in available discrete generators. If success criteria are met, meaning the electrical demand is met without overloading generators, the fitness of that function is calculated. If it is better than the previous solution, its cost and configuration are saved. This procedure is diagramed in Figure 10.

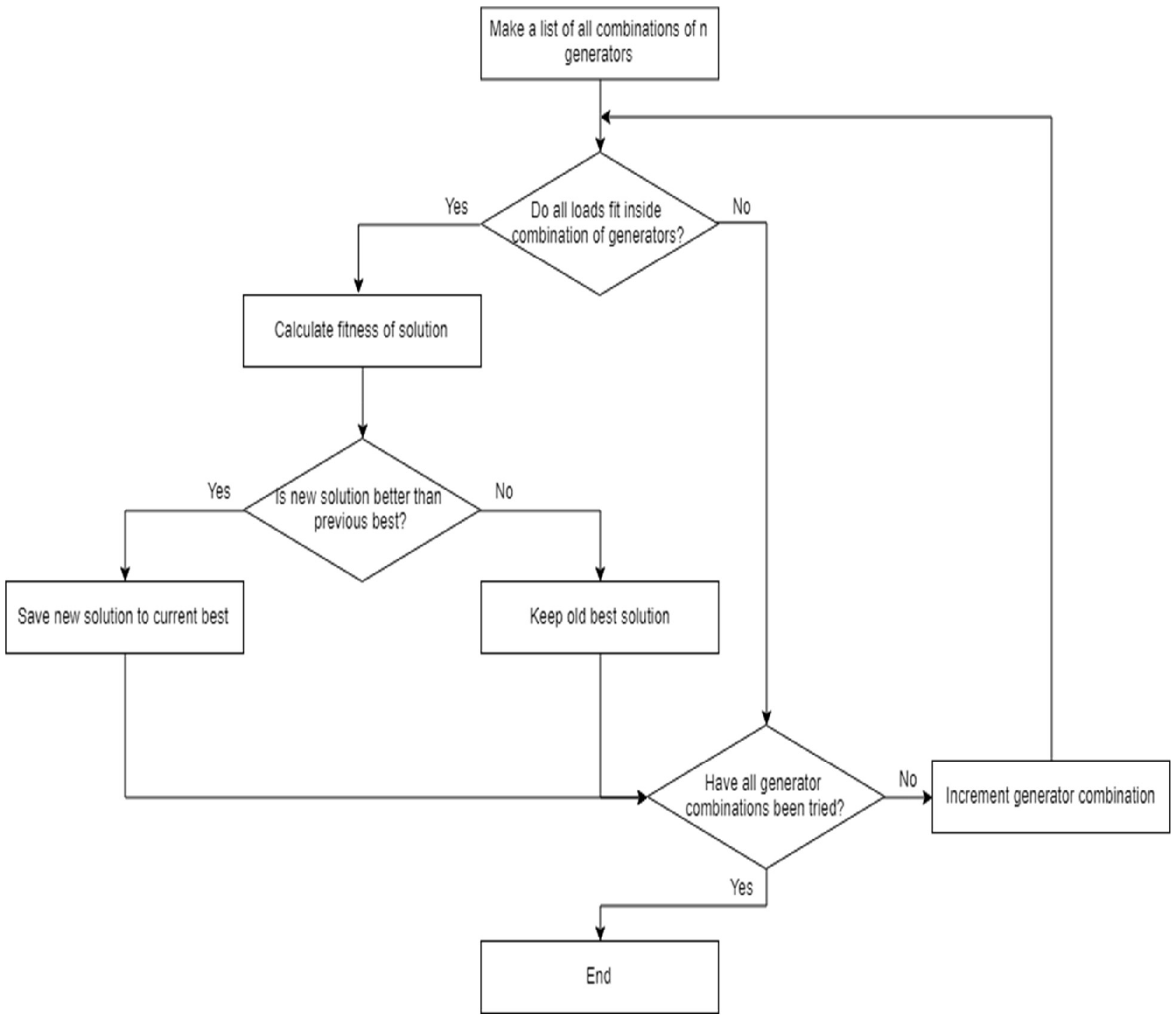
Make a list of all combinations of n generators
Do all loads fit inside combination of generators?
Yes
No
Calculate fitness of solution
Is new solution better than previous best?
Yes
No
Save new solution to current best
Keep old best solution
Have all generator combinations been tried?
No
Increment generator combination
Yes
End

Figure 10: ES Algorithm Flow Diagram

Since this method runs through every possible solution, the algorithm is comparatively slow and runtime increases factorially with the scale of the problem, but it guarantees finding the global optimum. Depending on the size of the forward operating base and the number of required loads and available generators, the longer run time may be permissible in exchange for the optimum solution at a small enough scale.

3.2 Procedure

3.2.1 Setup

The five algorithms (PSO, CS, BA, FFD, and ES) were coded using the Python programming language in the Jupyter Notebook integrated development environment (IDE). All the algorithms relied on a load power profile and a generator profile, while the metaheuristic algorithms and ES additionally relied on a load configuration file.

Load power profile data was generated to simulate a variety of different loads one might find: constant resistive loads, inductive cycling loads with different cycle periods, and a few small random loads to simulate small miscellaneous loads throughout the day. Power data was broken down into peak power segments for each discrete time step. While any data sample frequency can be input into the algorithms, four samples per hour corresponding to 15-minute interval data was used for a given 24-hour period. Breaking down the power consumption by time allows for the realistic pairing of complementary loads in a generator to average out, like exterior lighting for evening hours, and computers for work hours. The smaller the time interval, the more accurate and useful the exercise becomes. Fifteen-minute data was chosen to balance real world applicability with a short enough runtime, so as to run a statistically interesting number of trials. To test all load configurations, load sizes were selected for the capability to operate all loads in any one generator.

Generator profiles were created based on experimental field data collected from military diesel generators. The most common military generator line in use is the Tactical Quiet Generator (TQG). Fuel consumption curves were created for multiple capacity models by electrically loading the generators at 25%, 50%, 75%, and 100% capacity while measuring fuel consumption. For this simulation, 30 kW and 60 kW models and their respective consumption equations were used due to their predominant use. The 30 kW and 60 kW equations were respectively modeled as Equation (3.13) and Equation (3.14):

$$y = 3.594\mathrm{x}^{\wedge}3 - 5.6974\mathrm{x}^{\wedge}2 + 4.5771\mathrm{x} + 0 \tag{3.13}$$

$$y = 3.4458\mathrm{x}^{\wedge}3 - 6.4626\mathrm{x}^{\wedge}2 + 7.5029\mathrm{x} + 0 \tag{3.14}$$

respectively, where y is fuel consumption and x is percent loading of the generator. Their fuel consumption graphs are in Figure 11 and Figure 12. The cost of diesel fuel was set at a common forward operation base rate of $100 per gallon.

Configuration profiles were generated to list every possible combination of all loads in all generators, so long as all loads are used. These combination numbers were used as the “x-axis” for ES, PSO, BA, and CS as discussed in Section 3.2.2.

The load, generator, and configuration profile data were preprocessed and fed into the algorithms. A screening step was taken to assure that no impossible combinations were considered where the aggregate load exceeded the capacity of the generator. Each algorithm was graded on optimization of solution, i.e. lowest system cost, and shortness of runtime. Runtime counters were started after the variables and profile data were loaded into the programs and screened. Each algorithm was executed 100 times and average values were taken of their results.

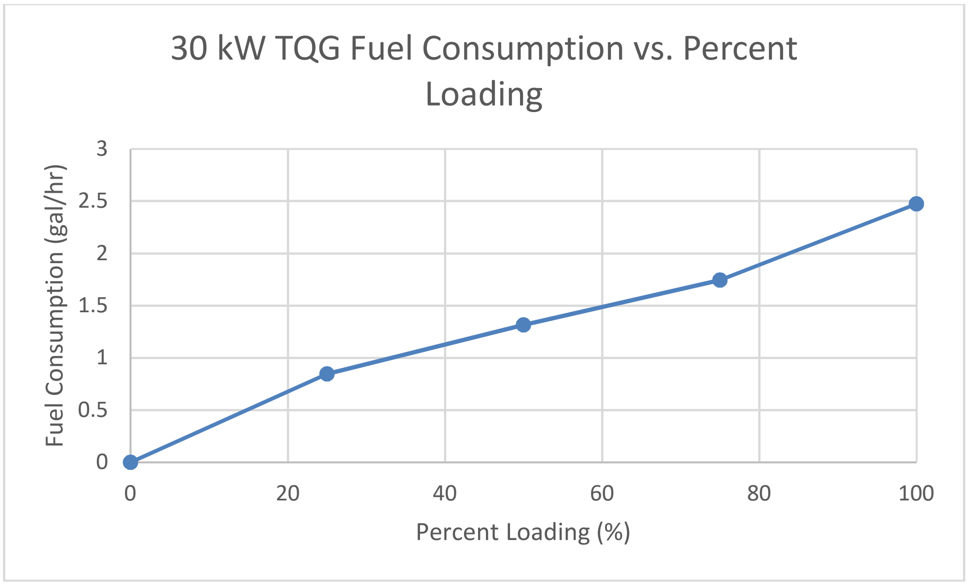

Figure 11: 30 kW TQG Fuel Consumption vs. Percent Loading

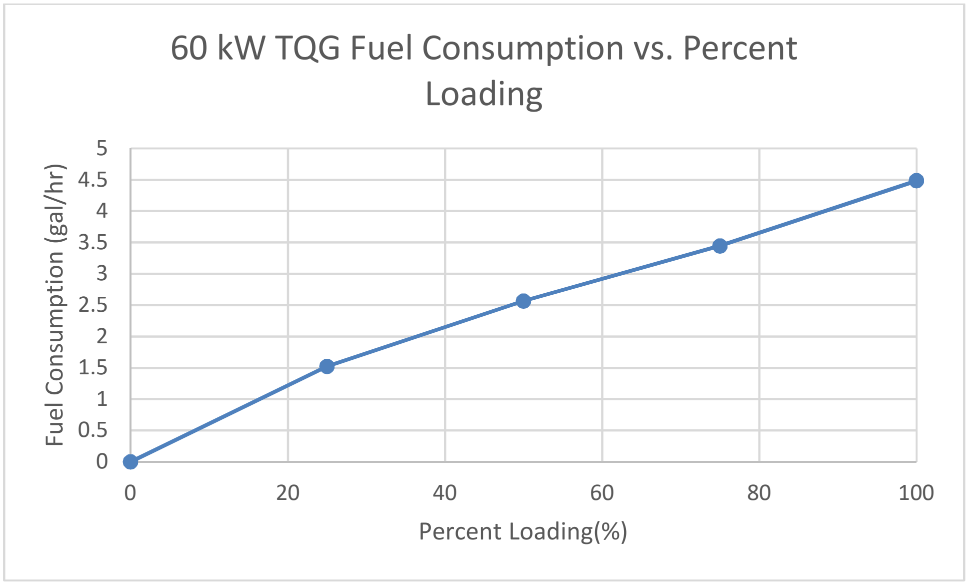

Figure 12: 60 kW TQG Fuel Consumption vs. Percent Loading

3.2.2 Dimension Reduction

Normally, the metaheuristics setup would consist of establishing one "fitness" dimension plus either n generator dimensions or m load dimensions, whichever is lower. However, this becomes very difficult to compare and work with when utilizing the ES method, which requires a single dimension. In addition, while the metaheuristic algorithms are capable at handling multi-dimensional problems, multidimensionality still tends to increase complexity and runtime. As a result, a technique was developed to reduce the metaheuristic search space from $n + 1$ or $m + 1$ dimensions down to only two.

In the initial setup of the computer code, all combinations and permutations of loads in generators are created, ordered, and addressed in a matrix such that $x = 1$ corresponds to a particular setup of loads in generators, and so forth. This generates the number of possible solutions proportional to Equation (3.15):

$$a \approx n^m \tag{3.15}$$

where a is the number of possible solutions, n is the number of available generators, and m is the number of required loads. While the number of solutions increases exponentially, the significant number of impossible combinations where a generator would be overloaded also increases at approximately the same rate. Consequently, these impossible combinations are removed from the matrix before the metaheuristics search them, thereby drastically decreasing the search space and saving the time of evaluating and subsequently discarding impossible solutions. The second "y-dimension" is simply the fitness, or fuel cost, of that specific combination in accordance to the defined fitness function.

3.2.3 Experimental Trials

Three different trial sizes were designed to test the algorithms on relatively small, medium, and large search space scales. The small trial simulated the scenario of two generators and five loads (2x5). For comparison to the ES method at a small scale, the metaheuristic algorithms were only allowed two particles and three iterations each. The medium trial scenario comprised two generators and ten loads (2x10). A good rule of thumb for small dimensional, n, uses of metaheuristic algorithms, where $n < 10$, is to assign particle numbers equal to $10n$. As a result, 10 particles were used, and metaheuristic algorithm iterations were set to 15. The large trial involved three generators and ten loads (3x10). Due to the vastly larger search space in the large trial, the particle number was set to 20 and iterations were set to 20.

The process of using optimization methods to tune and optimize optimization algorithms, called meta-optimization, is its own entire field of research. It is the last step in this nested optimization scheme illustrated in Figure 13. There are even many instances of using

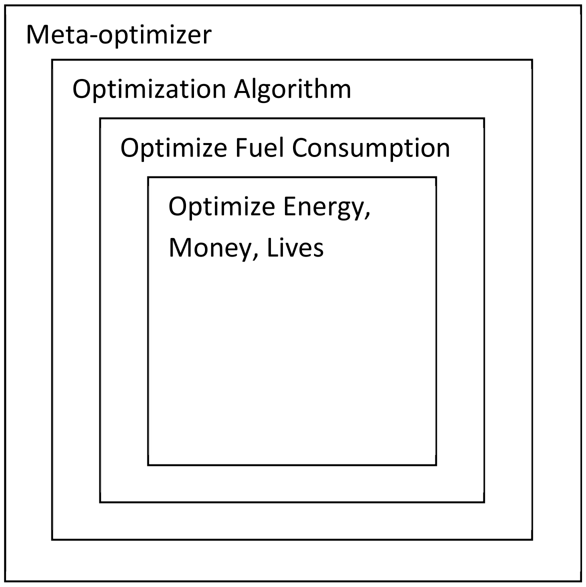

Figure 13: Nested Optimization Scheme

metaheuristic algorithms to tune other metaheuristic algorithms. In this experiment, an iterative method was used to test various tuning parameters for each metaheuristic algorithm. The tuning parameters for PSO include the inertial (w), individual (c_1), and social (c_2) weighting components. Common values for each of these coefficients were tested (w=0.5, c_1=c_2=1) as well as both a higher (w=0.75, c_1=c_2=1.5) and lower value (w=0.25, c_1=c_2=0.5), for a total of 27 unique combinations. The BA has tuning parameters including level of impulse emission (r_0), volume of sound (A_0), minimum wave frequency (f_{min}), maximum wave frequency (f_{max}), change of volume constant (α), and change of impulse emission constant (γ). Through analysis, the three constants found to affect the solution outcome most include the maximum wave frequency, the change of volume constant, and the change of impulse emission constant. Common values for these coefficients were tested (f_{max}=0.02, α=0.6, γ=0.6) in addition to higher (f_{max}=0.03, α=0.9, γ=0.9) and lower values (f_{max}=0.01, α=0.3, γ=0.3), for a total of 27 unique combinations while the other values were set to r_0=0.9, A_0=0.5, f_{min}=0. The CS algorithm was easiest to tune since it only has one parameter, p_a, for the probability that a cuckoo's egg would be detected. Due to a single tuning parameter, 11 tuning scenarios were tested, where p_a was between zero and one, with an incremental step size of 0.1.

Each trial first consists of a metaheuristic algorithm tuning step. To save runtime costs without harming tuning accuracy, the load profile was reduced from 96 samples, pertaining to 15-minute data for a day, down to 10. Each combination of tuning parameters listed above is tested 100 times and averaged for comparison. Since the number of iterations remains constant, runtime is fairly consistent; therefore, solution error as compared to the ES method's solution is used to compare the tuning parameters. The optimal tuning parameters for each algorithm are then run 100 times using the full 96-sample load profile and compared.

3.3 Results

3.3.1 Trial 1: Two Generators, Five Loads

The first, small trial consisted of the setup of just two generators and five loads. There are only about 31 unique combinations in the search space, as seen in Figure 14. It should be noted that combinations to the left of the graph have more loads consolidated to as few generators possible, whereas the loads distribute more evenly among the generators toward the right. As was theorized, the graph shows that highly loaded generator configurations were more fuel-efficient, and therefore more cost-efficient, than evenly distributed load configurations. Due to the nature of combinatorics, there are more distributed load configurations than consolidated load configurations, and accordingly there are more poor solutions than good solutions in the entirety of the search space, as seen in the Figure 15 histogram.

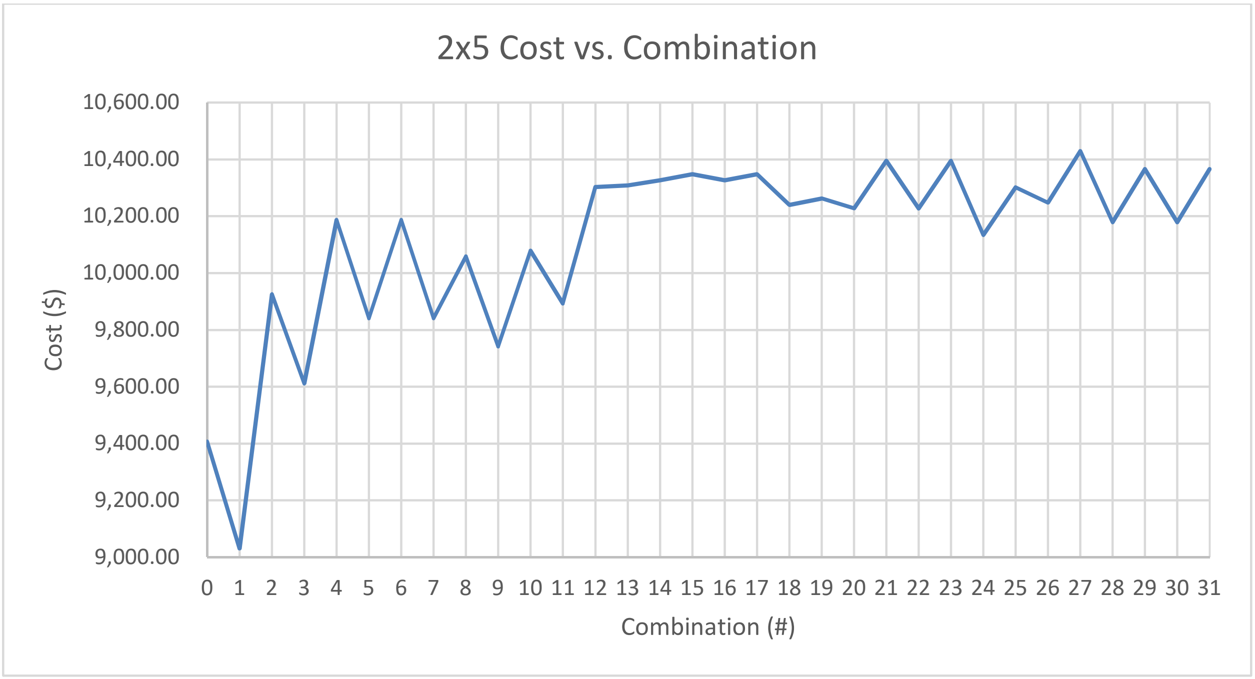

Figure 14: 2x5 Cost vs. Combination

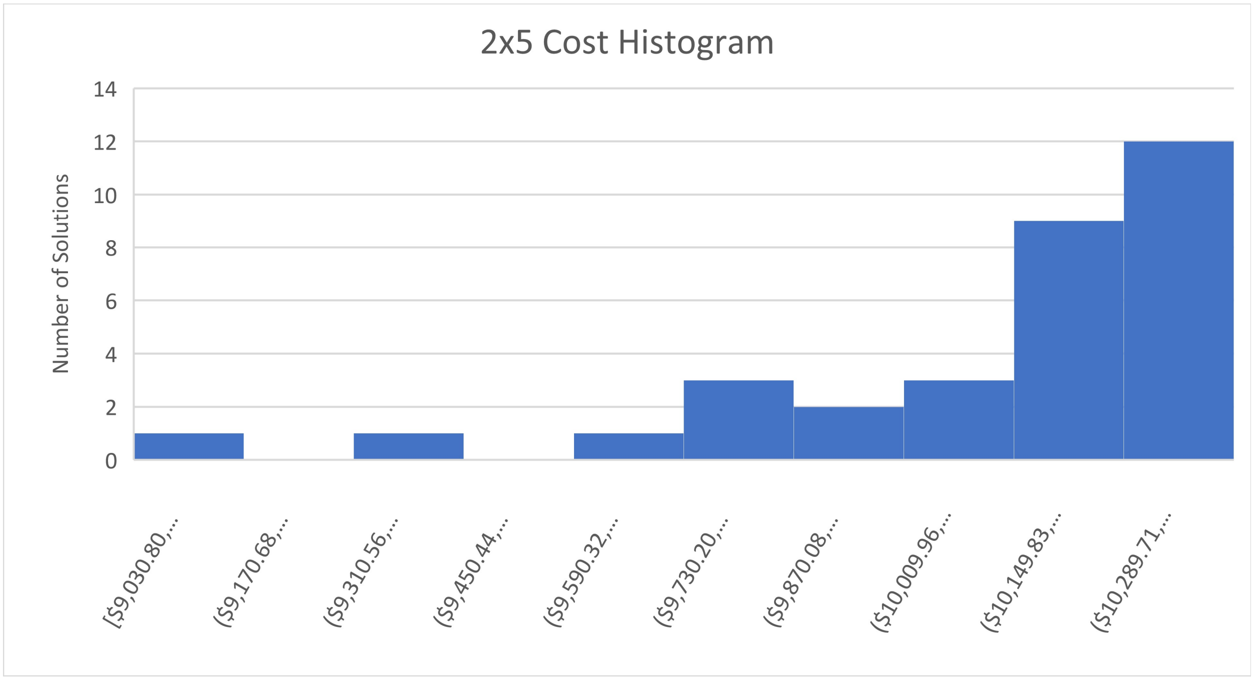

Figure 15: 2x5 Cost Histogram

As previously discussed, an iterative meta-optimization method was used to search for optimal tuning parameters for PSO, BA, and CS, shown in Table 2, Table 3, and Table 4 respectively. The most optimal tuning parameter for each algorithm to be used in the final results has been highlighted. Table 5 compares the winner of each algorithm's calibration results, and Table 6 shows the final results scaled up to a full day's worth of 15-minute interval data.

Table 2: 2x5 PSO Calibration Table

Trial	Solution					Runtime				
	Average	Range			SD	Average	Range			SD
w: 0.25 , c1: 0.5 , c2: 0.5	987.71	909.63	-	1032.92	34.06	0.03018	0.01562	-	0.05546	0.0067
w: 0.25 , c1: 0.5 , c2: 1.0	982.88	909.63	-	1030.44	32.88	0.03098	0.01562	-	0.05297	0.0055
w: 0.25 , c1: 0.5 , c2: 1.5	974.11	909.63	-	1034.40	41.03	0.02923	0.01561	-	0.04697	0.0055
w: 0.25 , c1: 1.0 , c2: 0.5	986.54	909.63	-	1032.31	32.91	0.02964	0.01561	-	0.04752	0.0058
w: 0.25 , c1: 1.0 , c2: 1.0	984.92	909.63	-	1030.44	33.95	0.03115	0.01562	-	0.05747	0.0074
w: 0.25 , c1: 1.0 , c2: 1.5	978.48	909.63	-	1038.48	36.60	0.03013	0.01518	-	0.04652	0.0042
w: 0.25 , c1: 1.5 , c2: 0.5	990.51	909.63	-	1032.92	32.71	0.02934	0.01562	-	0.04385	0.0044
w: 0.25 , c1: 1.5 , c2: 1.0	989.24	909.63	-	1032.92	32.73	0.02958	0.01562	-	0.04616	0.0040
w: 0.25 , c1: 1.5 , c2: 1.5	978.52	909.63	-	1032.31	38.70	0.02923	0.01511	-	0.04687	0.0056
w: 0.5 , c1: 0.5 , c2: 0.5	989.90	909.63	-	1032.92	29.95	0.03213	0.01562	-	0.04697	0.0052
w: 0.5 , c1: 0.5 , c2: 1.0	990.22	909.63	-	1032.92	34.77	0.02921	0.01417	-	0.04702	0.0048
w: 0.5 , c1: 0.5 , c2: 1.5	977.60	909.63	-	1032.92	38.73	0.03000	0.01561	-	0.06251	0.0068
w: 0.5 , c1: 1.0 , c2: 0.5	989.24	909.63	-	1038.48	30.60	0.03061	0.01562	-	0.06251	0.0074
w: 0.5 , c1: 1.0 , c2: 1.0	984.42	909.63	-	1038.48	38.10	0.03322	0.01561	-	0.07813	0.0107
w: 0.5 , c1: 1.0 , c2: 1.5	976.15	909.63	-	1031.99	38.40	0.03165	0.01505	-	0.06659	0.0082
w: 0.5 , c1: 1.5 , c2: 0.5	988.99	909.63	-	1034.40	36.69	0.02890	0.01561	-	0.04687	0.0062
w: 0.5 , c1: 1.5 , c2: 1.0	984.71	909.63	-	1032.92	35.12	0.02997	0.01561	-	0.06703	0.0072
w: 0.5 , c1: 1.5 , c2: 1.5	980.57	909.63	-	1032.31	36.69	0.02950	0.01562	-	0.04337	0.0032
w: 0.75 , c1: 0.5 , c2: 0.5	988.93	909.63	-	1032.92	33.48	0.02889	0.01562	-	0.04576	0.0036
w: 0.75 , c1: 0.5 , c2: 1.0	976.68	909.63	-	1031.99	37.46	0.02939	0.01316	-	0.06250	0.0056
w: 0.75 , c1: 0.5 , c2: 1.5	976.14	909.63	-	1032.92	37.63	0.02974	0.01315	-	0.04411	0.0040
w: 0.75 , c1: 1.0 , c2: 0.5	990.79	909.63	-	1032.92	29.27	0.02949	0.01562	-	0.03641	0.0021
w: 0.75 , c1: 1.0 , c2: 1.0	984.62	909.63	-	1032.92	36.57	0.02956	0.01561	-	0.05081	0.0062
w: 0.75 , c1: 1.0 , c2: 1.5	976.04	909.63	-	1034.40	37.82	0.03131	0.01561	-	0.07813	0.0090
w: 0.75 , c1: 1.5 , c2: 0.5	994.50	909.63	-	1032.92	25.30	0.02883	0.01561	-	0.04687	0.0060
w: 0.75 , c1: 1.5 , c2: 1.0	981.90	909.63	-	1030.44	36.15	0.03116	0.01561	-	0.04689	0.0067
w: 0.75 , c1: 1.5 , c2: 1.5	979.26	909.63	-	1031.99	37.22	0.02927	0.01561	-	0.04689	0.0057

Table 3: 2x5 BA Calibration Table

Trial	Solution					Runtime				
	Average	Range			SD	Average	Range			SD
fmax: 0.01 , alpha: 0.3 , csi: 0.3	989.53	909.63	-	1031.99	32.96	0.04233	0.02568	-	0.07844	0.0085
fmax: 0.01 , alpha: 0.3 , csi: 0.6	989.73	909.63	-	1038.48	34.91	0.04175	0.02613	-	0.07820	0.0084
fmax: 0.01 , alpha: 0.3 , csi: 0.9	994.50	909.63	-	1034.40	28.73	0.04177	0.02780	-	0.05706	0.0053
fmax: 0.01 , alpha: 0.6 , csi: 0.3	996.95	909.63	-	1034.40	32.75	0.04084	0.02616	-	0.05364	0.0050
fmax: 0.01 , alpha: 0.6 , csi: 0.6	992.59	909.63	-	1032.92	31.83	0.04137	0.02714	-	0.07226	0.0087
fmax: 0.01 , alpha: 0.6 , csi: 0.9	986.27	909.63	-	1034.40	35.22	0.04044	0.02834	-	0.04689	0.0077
fmax: 0.01 , alpha: 0.9 , csi: 0.3	993.41	909.63	-	1034.40	33.51	0.04166	0.03081	-	0.06251	0.0075
fmax: 0.01 , alpha: 0.9 , csi: 0.6	988.87	909.63	-	1032.92	34.19	0.04234	0.02690	-	0.09374	0.0116
fmax: 0.01 , alpha: 0.9 , csi: 0.9	989.13	909.63	-	1032.92	36.26	0.04113	0.02712	-	0.06937	0.0070
fmax: 0.02 , alpha: 0.3 , csi: 0.3	993.57	909.63	-	1038.48	34.18	0.04168	0.02887	-	0.05514	0.0059
fmax: 0.02 , alpha: 0.3 , csi: 0.6	987.92	909.63	-	1034.40	35.22	0.04108	0.02618	-	0.05670	0.0071
fmax: 0.02 , alpha: 0.3 , csi: 0.9	990.41	909.63	-	1034.40	35.47	0.04058	0.02572	-	0.05869	0.0069
fmax: 0.02 , alpha: 0.6 , csi: 0.3	988.64	909.63	-	1032.92	33.16	0.04265	0.03698	-	0.05796	0.0037
fmax: 0.02 , alpha: 0.6 , csi: 0.6	988.50	909.63	-	1032.92	32.69	0.04110	0.02382	-	0.07390	0.0070
fmax: 0.02 , alpha: 0.6 , csi: 0.9	994.87	909.63	-	1034.40	31.96	0.04116	0.03081	-	0.06249	0.0080
fmax: 0.02 , alpha: 0.9 , csi: 0.3	988.81	909.63	-	1034.40	33.82	0.04214	0.02786	-	0.08927	0.0086
fmax: 0.02 , alpha: 0.9 , csi: 0.6	989.53	909.63	-	1032.31	35.17	0.04106	0.03123	-	0.06405	0.0075
fmax: 0.02 , alpha: 0.9 , csi: 0.9	991.75	909.63	-	1032.92	27.97	0.04183	0.02575	-	0.08122	0.0082
fmax: 0.03 , alpha: 0.3 , csi: 0.3	990.08	909.63	-	1032.92	33.83	0.04173	0.02512	-	0.05996	0.0059
fmax: 0.03 , alpha: 0.3 , csi: 0.6	990.58	909.63	-	1032.92	32.98	0.04162	0.02618	-	0.05586	0.0056
fmax: 0.03 , alpha: 0.3 , csi: 0.9	992.48	909.63	-	1034.40	29.33	0.04113	0.02512	-	0.05168	0.0060
fmax: 0.03 , alpha: 0.6 , csi: 0.3	980.42	909.63	-	1032.92	37.46	0.04112	0.02527	-	0.05303	0.0058
fmax: 0.03 , alpha: 0.6 , csi: 0.6	990.95	909.63	-	1034.40	35.37	0.04263	0.03123	-	0.05089	0.0038
fmax: 0.03 , alpha: 0.6 , csi: 0.9	993.26	909.63	-	1032.92	34.58	0.04136	0.03123	-	0.07269	0.0075
fmax: 0.03 , alpha: 0.9 , csi: 0.3	988.05	909.63	-	1034.40	35.16	0.04134	0.03123	-	0.06250	0.0081
fmax: 0.03 , alpha: 0.9 , csi: 0.6	987.85	909.63	-	1034.40	32.87	0.04410	0.02616	-	0.07879	0.0076
fmax: 0.03 , alpha: 0.9 , csi: 0.9	995.19	909.63	-	1032.92	33.50	0.04150	0.03069	-	0.06354	0.0064

Table 4: 2x5 CS Calibration Table

Trial	Solution					Runtime				
	Average	Range			SD	Average	Range			SD
pa: 0	965.88	909.63	-	1031.99	36.82	0.05689	0.04503	-	0.08876	0.0075
pa: 0.1	969.70	909.63	-	1029.38	35.19	0.05956	0.04227	-	0.07196	0.0052
pa: 0.2	966.39	909.63	-	1024.30	36.95	0.06040	0.04142	-	0.09777	0.0100
pa: 0.3	969.54	909.63	-	1024.30	35.76	0.06017	0.04686	-	0.11636	0.0097
pa: 0.4	972.02	909.63	-	1024.30	34.37	0.05952	0.04687	-	0.07495	0.0049
pa: 0.5	967.85	909.63	-	1029.38	34.74	0.05854	0.04686	-	0.09150	0.0080
pa: 0.6	965.97	909.63	-	1019.44	35.38	0.05935	0.04376	-	0.09374	0.0074
pa: 0.7	964.75	909.63	-	1014.35	37.23	0.05752	0.04352	-	0.07814	0.0086
pa: 0.8	967.71	909.63	-	1027.57	36.79	0.05729	0.04686	-	0.07660	0.0077
pa: 0.9	967.45	909.63	-	1027.57	36.26	0.06010	0.04344	-	0.08962	0.0081
pa: 1.0	971.94	909.63	-	1029.38	31.90	0.05917	0.03977	-	0.12021	0.0094

Table 5: 2x5 Calibration Results

2x5 Calibration Results												
Algorithm	Solution						Runtime					
	Average ($)	% Error	Range ($)			SD	Average (s)	% Diff	Range (s)			SD
ES	909.63	0.0%	909.63	-	909.63	0.00	0.0428	0.0%	0.0261	-	0.1052	0.0107
FFD	909.63	0.0%	909.63	-	909.63	0.00	0.0066	-146.8%	0.0042	-	0.0080	0.0006
PSO	974.11	6.8%	909.63	-	1,034.40	41.03	0.0292	-37.7%	0.0156	-	0.0470	0.0055
BA	980.42	7.5%	909.63	-	1,032.92	37.46	0.0411	-4.1%	0.0253	-	0.0530	0.0058
CS	964.75	5.9%	909.63	-	1,014.35	37.23	0.0575	29.3%	0.0435	-	0.0781	0.0086

Table 6: 2x5 Final Results

2x5 Final Results												
Algorithm	Solution						Runtime					
	Average ($)	% Error	Range ($)			SD	Average (s)	% Diff	Range (s)			SD
ES	9,030.80	0.0%	9,030.80	-	9,030.80	0.00	0.352	0.0%	0.344	-	0.391	0.0097
FFD	9,030.80	0.0%	9,030.80	-	9,030.80	0.00	0.014	-184.9%	0.008	-	0.023	0.0018
PSO	9,819.37	8.4%	9,030.80	-	10,429.59	401.62	0.214	-48.5%	0.187	-	0.250	0.0137
BA	9,865.65	8.8%	9,030.80	-	10,365.36	404.75	0.299	-16.0%	0.281	-	0.320	0.0115
CS	9,715.45	7.3%	9,030.80	-	10,239.40	353.26	0.420	17.7%	0.359	-	0.484	0.0201

The Trial 1 results demonstrate that, for similar runtimes compared to the ES method at the small scale, metaheuristic algorithms can provide solution errors around seven to eight percent in this application. However, the first fit decreasing algorithm was shown to provide the perfect solution with a runtime an order of magnitude smaller than the ES method. Nevertheless, metaheuristic algorithms are expected to perform better on bigger data sets.

3.3.2 Trial 2: Two Generators, Ten Loads

The second, medium trial consisted of the setup of two generators and ten loads. There are about 1000 unique combinations in the search space, as seen in Figure 16. Like before, more distributed load configurations can be seen on the right of the graph, occurring in a step-like pattern as configurations change from 10 loads in Generator 1 and 0 loads in Generator 2, all the way to 5 loads in Generator 1 and 5 loads in Generator 2. Again, Figure 17 illustrates the increasing gap between a few good solutions and an avalanche of poor solutions.

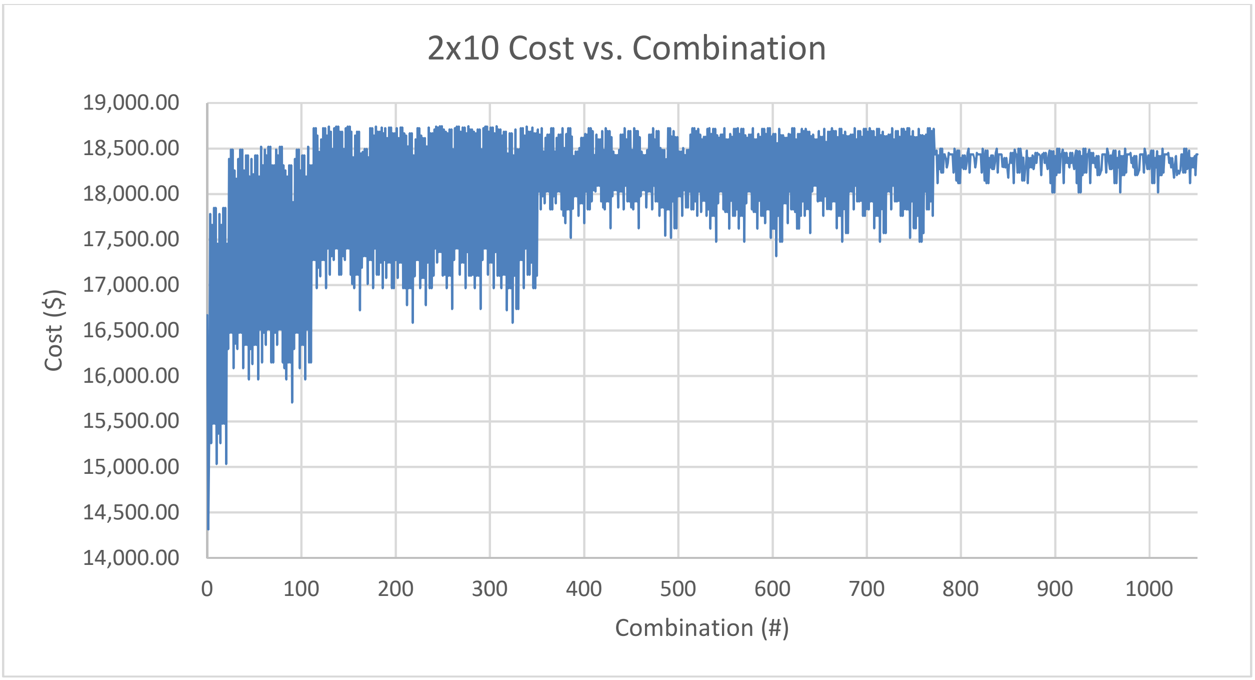

Figure 16: 2x10 Cost vs. Combination Graph

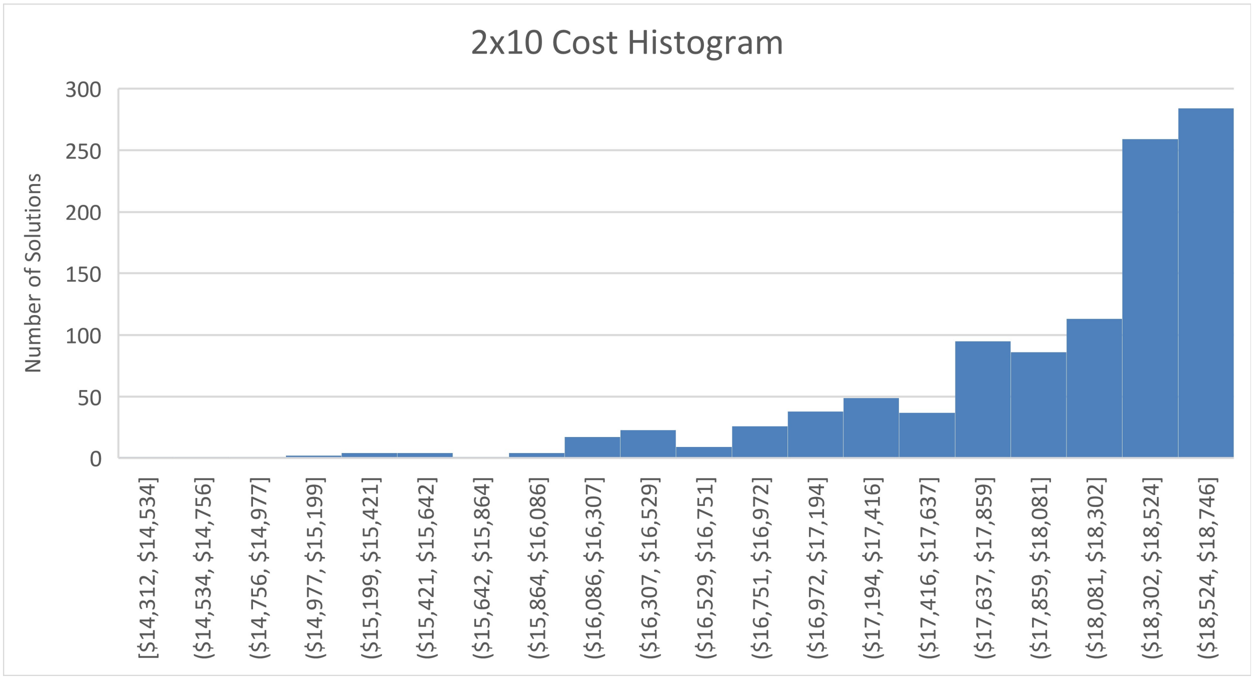

Figure 17: 2x10 Cost Histogram

Trial 2 meta-optimization was performed for PSO, BA, and CS, shown in Table 7, Table 8, and Table 9 respectively. Table 10 compares the winner of each algorithm's calibration results, and Table 11 shows the final results scaled up to a full day's worth of 15-minute interval data.

Table 7: 2x10 PSO Calibration Table

Trial	Solution					Runtime				
	Average	Range			SD	Average	Range			SD
w: 0.25 , c1: 0.5 , c2: 0.5	1611.44	1438.14	-	1753.31	60.49	0.9197	0.9025	-	1.0104	0.012
w: 0.25 , c1: 0.5 , c2: 1.0	1596.74	1438.14	-	1754.20	96.77	0.9291	0.8745	-	1.2393	0.042
w: 0.25 , c1: 0.5 , c2: 1.5	1524.73	1438.14	-	1753.31	92.36	0.9098	0.8675	-	0.9914	0.022
w: 0.25 , c1: 1.0 , c2: 0.5	1613.85	1438.14	-	1736.55	56.49	0.9439	0.9065	-	1.4522	0.075
w: 0.25 , c1: 1.0 , c2: 1.0	1579.79	1438.14	-	1743.79	89.68	0.9246	0.8885	-	1.0504	0.022
w: 0.25 , c1: 1.0 , c2: 1.5	1514.40	1438.14	-	1753.31	90.01	0.9101	0.8655	-	0.9744	0.019
w: 0.25 , c1: 1.5 , c2: 0.5	1628.82	1438.14	-	1743.79	61.61	0.9222	0.9025	-	1.0194	0.015
w: 0.25 , c1: 1.5 , c2: 1.0	1580.54	1438.14	-	1753.31	95.31	0.9190	0.8755	-	0.9554	0.011
w: 0.25 , c1: 1.5 , c2: 1.5	1499.57	1438.14	-	1753.31	86.28	0.9114	0.8655	-	1.0514	0.029
w: 0.5 , c1: 0.5 , c2: 0.5	1593.46	1438.14	-	1743.79	81.93	0.9226	0.9005	-	1.0524	0.016
w: 0.5 , c1: 0.5 , c2: 1.0	1517.26	1438.14	-	1754.20	97.60	0.9105	0.8635	-	1.0054	0.018
w: 0.5 , c1: 0.5 , c2: 1.5	1495.30	1438.14	-	1753.31	84.27	0.9029	0.8615	-	1.0404	0.026
w: 0.5 , c1: 1.0 , c2: 0.5	1595.90	1438.14	-	1743.79	82.78	0.9250	0.9015	-	1.0474	0.021
w: 0.5 , c1: 1.0 , c2: 1.0	1525.39	1438.14	-	1753.31	96.81	0.9142	0.8705	-	0.9904	0.017
w: 0.5 , c1: 1.0 , c2: 1.5	1490.14	1438.14	-	1743.79	81.02	0.9040	0.8665	-	1.0224	0.021
w: 0.5 , c1: 1.5 , c2: 0.5	1599.06	1438.14	-	1753.31	84.56	0.9256	0.8995	-	1.0474	0.022
w: 0.5 , c1: 1.5 , c2: 1.0	1527.51	1438.14	-	1753.31	98.59	0.9117	0.8715	-	1.0404	0.020
w: 0.5 , c1: 1.5 , c2: 1.5	1498.16	1438.14	-	1743.79	97.41	0.9035	0.8705	-	1.0454	0.022
w: 0.75 , c1: 0.5 , c2: 0.5	1505.16	1438.14	-	1736.55	88.08	0.9140	0.8825	-	1.0744	0.020
w: 0.75 , c1: 0.5 , c2: 1.0	1484.53	1438.14	-	1743.79	83.58	0.8980	0.8665	-	0.9464	0.016
w: 0.75 , c1: 0.5 , c2: 1.5	1500.86	1438.14	-	1753.31	86.74	0.8946	0.8525	-	0.9564	0.017
w: 0.75 , c1: 1.0 , c2: 0.5	1521.48	1438.14	-	1753.31	81.31	0.9154	0.8855	-	1.0034	0.016
w: 0.75 , c1: 1.0 , c2: 1.0	1491.71	1438.14	-	1753.31	78.04	0.9053	0.8705	-	0.9924	0.016
w: 0.75 , c1: 1.0 , c2: 1.5	1489.62	1438.14	-	1743.79	79.32	0.8968	0.8545	-	0.9994	0.021
w: 0.75 , c1: 1.5 , c2: 0.5	1527.96	1438.14	-	1754.20	81.87	0.9159	0.8905	-	1.0334	0.018
w: 0.75 , c1: 1.5 , c2: 1.0	1482.63	1438.14	-	1736.55	67.79	0.9083	0.8635	-	1.0114	0.021
w: 0.75 , c1: 1.5 , c2: 1.5	1478.91	1438.14	-	1669.34	58.16	0.8986	0.8575	-	0.9355	0.015

Table 8: 2x10 BA Calibration Table

Trial	Solution				Runtime			
	Average	Range		SD	Average	Range		SD
fmax: 0.01 , alpha: 0.3 , csi: 0.3	1550.30	1438.14 -	1753.31	91.84	1.4736	1.4202 -	1.6560	0.033
fmax: 0.01 , alpha: 0.3 , csi: 0.6	1540.10	1438.14 -	1754.20	89.78	1.4698	1.4242 -	1.5761	0.021
fmax: 0.01 , alpha: 0.3 , csi: 0.9	1547.23	1438.14 -	1730.63	81.07	1.4761	1.4122 -	1.6531	0.036
fmax: 0.01 , alpha: 0.6 , csi: 0.3	1554.77	1438.14 -	1743.79	89.78	1.4738	1.4252 -	1.5961	0.022
fmax: 0.01 , alpha: 0.6 , csi: 0.6	1553.92	1438.14 -	1753.31	86.07	1.4740	1.4352 -	1.5731	0.017
fmax: 0.01 , alpha: 0.6 , csi: 0.9	1556.21	1438.14 -	1743.79	87.08	1.4745	1.4062 -	1.6171	0.027
fmax: 0.01 , alpha: 0.9 , csi: 0.3	1545.97	1438.14 -	1743.79	86.80	1.4698	1.4232 -	1.5641	0.019
fmax: 0.01 , alpha: 0.9 , csi: 0.6	1546.41	1438.14 -	1743.79	83.57	1.4746	1.4212 -	1.6241	0.022
fmax: 0.01 , alpha: 0.9 , csi: 0.9	1540.24	1438.14 -	1730.63	81.25	1.4693	1.4342 -	1.5011	0.013
fmax: 0.02 , alpha: 0.3 , csi: 0.3	1529.03	1438.14 -	1694.46	75.99	1.4773	1.4422 -	1.6211	0.026
fmax: 0.02 , alpha: 0.3 , csi: 0.6	1531.44	1438.14 -	1736.55	83.07	1.4729	1.4132 -	1.5901	0.024
fmax: 0.02 , alpha: 0.3 , csi: 0.9	1530.61	1438.14 -	1753.31	77.91	1.4747	1.4242 -	1.6560	0.029
fmax: 0.02 , alpha: 0.6 , csi: 0.3	1524.75	1438.14 -	1753.31	76.12	1.4675	1.4212 -	1.5051	0.016
fmax: 0.02 , alpha: 0.6 , csi: 0.6	1540.97	1438.14 -	1730.63	75.02	1.4759	1.4292 -	1.6291	0.031
fmax: 0.02 , alpha: 0.6 , csi: 0.9	1539.51	1438.14 -	1743.79	83.62	1.4695	1.4232 -	1.6900	0.027
fmax: 0.02 , alpha: 0.9 , csi: 0.3	1540.52	1438.14 -	1736.55	77.39	1.4742	1.4452 -	1.7660	0.032
fmax: 0.02 , alpha: 0.9 , csi: 0.6	1536.37	1438.14 -	1753.31	75.58	1.4732	1.4282 -	1.5231	0.014
fmax: 0.02 , alpha: 0.9 , csi: 0.9	1531.36	1438.14 -	1753.31	77.68	1.4738	1.4412 -	1.6400	0.025
fmax: 0.03 , alpha: 0.3 , csi: 0.3	1520.95	1438.14 -	1754.20	74.34	1.4689	1.4262 -	1.6121	0.023
fmax: 0.03 , alpha: 0.3 , csi: 0.6	1530.28	1438.14 -	1753.31	70.68	1.4727	1.4212 -	1.5941	0.029
fmax: 0.03 , alpha: 0.3 , csi: 0.9	1518.72	1438.14 -	1754.20	80.57	1.4690	1.4132 -	1.6021	0.026
fmax: 0.03 , alpha: 0.6 , csi: 0.3	1522.12	1438.14 -	1753.31	78.18	1.4752	1.4382 -	1.5951	0.025
fmax: 0.03 , alpha: 0.6 , csi: 0.6	1534.29	1438.14 -	1673.14	70.53	1.4673	1.4182 -	1.5041	0.014
fmax: 0.03 , alpha: 0.6 , csi: 0.9	1541.13	1438.14 -	1736.55	77.80	1.4722	1.4332 -	1.6640	0.023
fmax: 0.03 , alpha: 0.9 , csi: 0.3	1533.77	1438.14 -	1743.79	76.91	1.4660	1.4082 -	1.4991	0.014
fmax: 0.03 , alpha: 0.9 , csi: 0.6	1529.15	1438.14 -	1711.10	68.74	1.4715	1.4242 -	1.6580	0.023
fmax: 0.03 , alpha: 0.9 , csi: 0.9	1529.92	1438.14 -	1753.31	83.96	1.4692	1.4162 -	1.5141	0.013

Table 9: 2x10 CS Calibration Table

Trial	Solution					Runtime				
	Average	Range			SD	Average	Range			SD
pa: 0	1560.49	1438.14	-	1743.79	84.44	1.8316	1.7490	-	2.0458	0.033
pa: 0.1	1564.48	1438.14	-	1781.41	83.33	1.8300	1.7560	-	1.9499	0.026
pa: 0.2	1561.80	1438.14	-	1743.79	83.61	1.8332	1.7540	-	2.0388	0.039
pa: 0.3	1553.87	1438.14	-	1754.20	95.92	1.8280	1.7400	-	1.9998	0.035
pa: 0.4	1538.57	1438.14	-	1711.10	78.98	1.8238	1.7360	-	1.8709	0.021
pa: 0.5	1552.18	1438.14	-	1739.60	76.34	1.8312	1.7540	-	1.9699	0.023
pa: 0.6	1541.36	1438.14	-	1721.06	76.33	1.8263	1.7460	-	2.0958	0.034
pa: 0.7	1546.37	1438.14	-	1736.55	79.72	1.8260	1.7750	-	1.9629	0.027
pa: 0.8	1531.71	1438.14	-	1705.45	69.73	1.8291	1.7820	-	1.9429	0.021
pa: 0.9	1546.27	1438.14	-	1711.10	78.11	1.8242	1.7360	-	1.8669	0.019
pa: 1.0	1539.39	1438.14	-	1736.55	73.17	1.8301	1.7830	-	1.9399	0.024

Table 10: 2x10 Calibration Results

2x10 Calibration Results												
Algorithm	Solution						Runtime					
	Average ($)	% Error	Range ($)			SD	Average (s)	% Diff	Range (s)			SD
ES	1,438.14	0.0%	1,438.14	-	1,438.14	0.00	2.1057	0.0%	2.0568	-	2.6086	0.063
FFD	1,438.14	0.0%	1,438.14	-	1,438.14	0.00	0.0072	-198.6%	0.0060	-	0.0090	0.001
PSO	1,478.91	2.8%	1,438.14	-	1,669.34	58.16	0.8986	-80.4%	0.8575	-	0.9355	0.015
BA	1,518.72	5.5%	1,438.14	-	1,754.20	80.57	1.4690	-35.6%	1.4132	-	1.6021	0.026
CS	1,531.71	6.3%	1,438.14	-	1,705.45	69.73	1.8291	-14.1%	1.7820	-	1.9429	0.021

Table 11: 2x10 Final Results

2x10 Final Results												
Algorithm	Solution						Runtime					
	Average ($)	% Error	Range ($)			SD	Average (s)	% Diff	Range (s)			SD
ES	14,312.39	0.0%	14,312.39	-	14,312.39	0.00	16.820	0.0%	16.702	-	17.041	0.068
FFD	14,312.39	0.0%	14,312.39	-	14,312.39	0.00	0.017	-199.6%	0.016	-	0.033	0.003
PSO	14,822.25	3.5%	14,312.39	-	17,473.65	728.42	7.147	-80.7%	6.559	-	7.497	0.140
BA	15,228.21	6.2%	14,312.39	-	17,473.65	867.32	11.771	-35.3%	10.249	-	12.296	0.275
CS	15,287.88	6.6%	14,312.39	-	16,778.14	770.78	14.598	-14.1%	11.624	-	14.952	0.343

The Trial 2 results demonstrate even higher metaheuristic algorithm solution accuracy than Trial 1, with even faster runtimes compared to the ES method. PSO provides 3.5% higher average solution with less than half the runtime of ES, while demonstrating a smaller standard deviation than the other metaheuristic algorithms. The FFD algorithm again comes out the clear winner in accuracy and runtime, performing the same task as ES three orders of magnitude faster.

3.3.3 Trial 3: Three Generators, Ten Loads

The third, large trial consisted of the setup of three generators and ten loads. There are about 70,000 unique combinations in the search space, as seen in Figure 18. This graph demonstrates characteristics similar to those of the other trials with an even higher disparity in the number of good versus poor solutions in Figure 19.

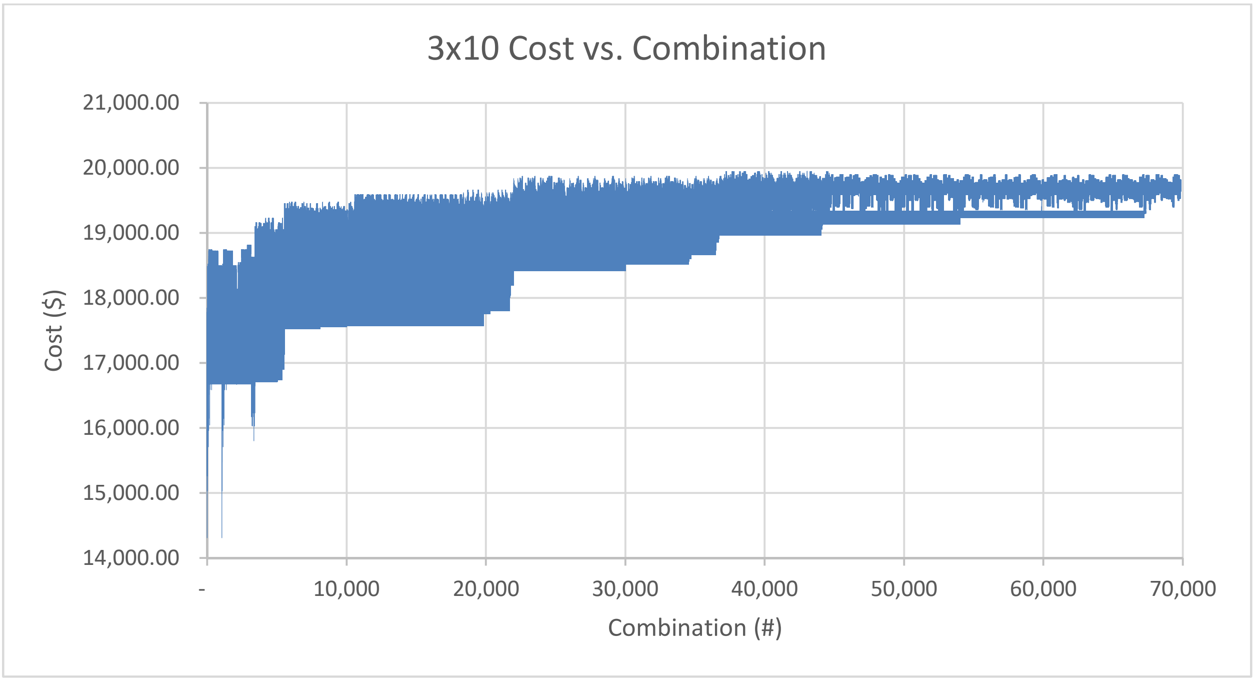

Figure 18: 3x10 Cost vs. Combination Graph

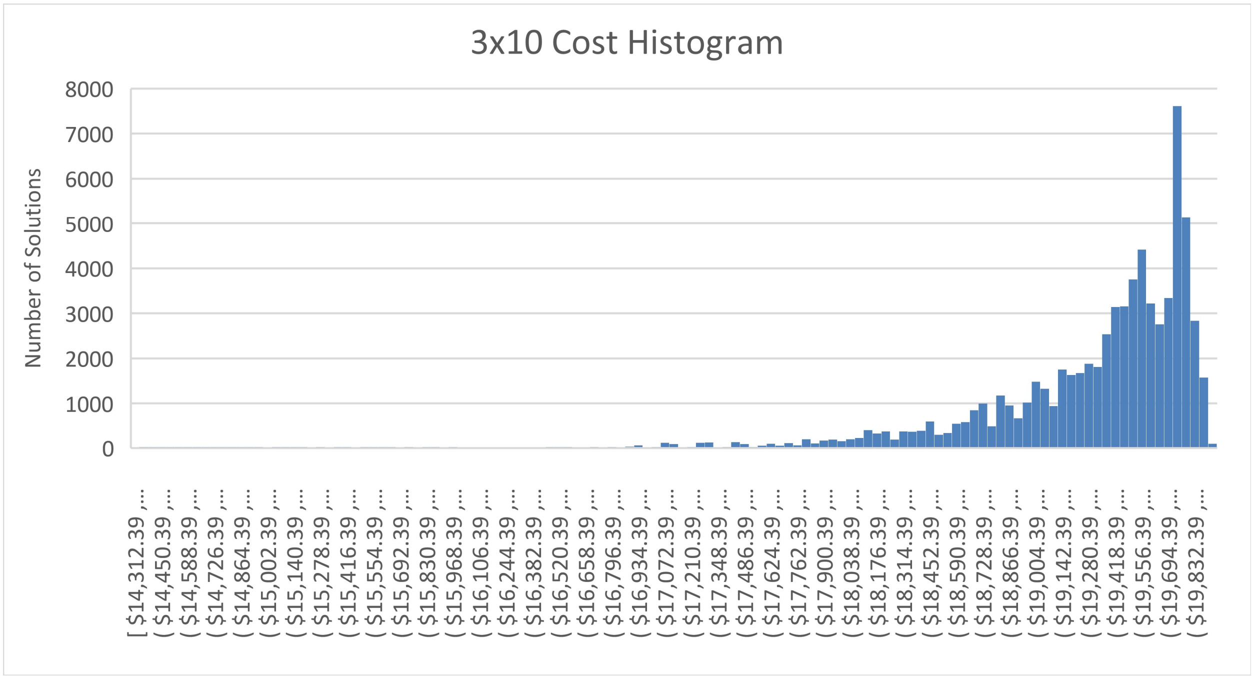

Figure 19: 3x10 Cost Histogram

Trial 3 meta-optimization was performed for PSO, BA, and CS, shown in Table 12, Table 13, and Table 14 respectively. Table 15 compares the winner of each algorithm's calibration results, and Table 16 shows the final results scaled up to a full day's worth of 15-minute interval data.

Table 12: 3x10 PSO Calibration Table

Trial	Solution					Runtime				
	Average	Range			SD	Average	Range			SD
w: 0.25 , c1: 0.5 , c2: 0.5	1688.59	1438.14	-	1787.08	87.62	2.4712	2.0244	-	2.9360	0.374
w: 0.25 , c1: 0.5 , c2: 1.0	1552.78	1438.14	-	1789.24	122.97	2.7266	2.5623	-	3.0154	0.113
w: 0.25 , c1: 0.5 , c2: 1.5	1588.01	1438.14	-	1774.74	114.73	2.6213	2.4998	-	2.9529	0.112
w: 0.25 , c1: 1.0 , c2: 0.5	1704.95	1500.44	-	1820.05	74.73	2.8879	2.8123	-	3.0320	0.026
w: 0.25 , c1: 1.0 , c2: 1.0	1547.91	1438.14	-	1844.68	117.94	2.7532	2.5658	-	2.9918	0.104
w: 0.25 , c1: 1.0 , c2: 1.5	1579.70	1438.14	-	1773.08	111.92	2.6146	2.4998	-	3.1256	0.110
w: 0.25 , c1: 1.5 , c2: 0.5	1690.90	1438.14	-	1809.04	77.51	2.8984	2.8279	-	3.0154	0.038
w: 0.25 , c1: 1.5 , c2: 1.0	1555.96	1438.14	-	1787.08	117.39	2.7421	2.5467	-	2.9217	0.094
w: 0.25 , c1: 1.5 , c2: 1.5	1577.32	1438.14	-	1782.09	110.41	2.6012	2.5024	-	2.8904	0.087
w: 0.5 , c1: 0.5 , c2: 0.5	1585.57	1438.14	-	1789.24	119.75	2.7968	2.6248	-	3.0467	0.081
w: 0.5 , c1: 0.5 , c2: 1.0	1561.67	1438.14	-	1770.12	109.43	2.6558	2.5310	-	2.9841	0.108
w: 0.5 , c1: 0.5 , c2: 1.5	1601.18	1438.14	-	1846.61	102.97	2.5571	2.4998	-	2.9216	0.061
w: 0.5 , c1: 1.0 , c2: 0.5	1612.98	1438.14	-	1842.52	119.29	2.8268	2.6873	-	3.0310	0.064
w: 0.5 , c1: 1.0 , c2: 1.0	1557.11	1438.14	-	1852.89	112.90	2.6574	2.5467	-	2.9495	0.088
w: 0.5 , c1: 1.0 , c2: 1.5	1584.51	1438.14	-	1669.34	103.91	2.5734	2.4998	-	2.7249	0.058
w: 0.5 , c1: 1.5 , c2: 0.5	1595.62	1438.14	-	1791.47	111.31	2.8253	2.6717	-	2.9842	0.058
w: 0.5 , c1: 1.5 , c2: 1.0	1543.92	1438.14	-	1770.12	102.62	2.6575	2.5154	-	3.0935	0.094
w: 0.5 , c1: 1.5 , c2: 1.5	1587.39	1438.14	-	1669.34	101.06	2.5666	2.4998	-	2.7810	0.053
w: 0.75 , c1: 0.5 , c2: 0.5	1574.77	1438.14	-	1774.74	73.80	2.6671	2.5623	-	2.9373	0.068
w: 0.75 , c1: 0.5 , c2: 1.0	1607.34	1438.14	-	1669.34	84.31	2.5728	2.5154	-	2.7810	0.047
w: 0.75 , c1: 0.5 , c2: 1.5	1617.67	1438.14	-	1669.34	80.83	2.5491	2.4998	-	2.7810	0.049
w: 0.75 , c1: 1.0 , c2: 0.5	1562.79	1438.14	-	1774.74	78.36	2.7147	2.5779	-	2.9217	0.066
w: 0.75 , c1: 1.0 , c2: 1.0	1596.59	1438.14	-	1842.52	87.96	2.5866	2.5121	-	2.8748	0.055
w: 0.75 , c1: 1.0 , c2: 1.5	1625.79	1438.14	-	1669.34	75.86	2.5353	2.4842	-	2.6881	0.033
w: 0.75 , c1: 1.5 , c2: 0.5	1577.01	1438.14	-	1782.09	77.06	2.7219	2.5779	-	3.1560	0.076
w: 0.75 , c1: 1.5 , c2: 1.0	1578.51	1438.14	-	1669.34	78.92	2.6057	2.5311	-	2.8255	0.048
w: 0.75 , c1: 1.5 , c2: 1.5	1620.68	1438.14	-	1669.34	77.83	2.5456	2.4998	-	2.7810	0.042

Table 13: 3x10 BA Calibration Table

Trial	Solution					Runtime				
	Average	Range			SD	Average	Range			SD
fmax: 0.01 , alpha: 0.3 , csi: 0.3	1590.49	1438.14	-	1774.74	72.87	4.5930	4.4372	-	4.8065	0.084
fmax: 0.01 , alpha: 0.3 , csi: 0.6	1610.55	1438.14	-	1789.24	76.44	4.5952	4.4372	-	4.8278	0.077
fmax: 0.01 , alpha: 0.3 , csi: 0.9	1597.39	1438.14	-	1837.21	78.14	4.5957	4.4372	-	4.8590	0.083
fmax: 0.01 , alpha: 0.6 , csi: 0.3	1601.51	1438.14	-	1782.09	76.62	4.5977	4.4059	-	4.7456	0.074
fmax: 0.01 , alpha: 0.6 , csi: 0.6	1600.54	1438.14	-	1789.24	84.11	4.6057	4.4059	-	4.7421	0.083
fmax: 0.01 , alpha: 0.6 , csi: 0.9	1591.61	1438.14	-	1852.89	76.74	4.6158	4.4777	-	4.8903	0.075
fmax: 0.01 , alpha: 0.9 , csi: 0.3	1603.36	1438.14	-	1838.77	82.16	4.6276	4.4684	-	4.8434	0.074
fmax: 0.01 , alpha: 0.9 , csi: 0.6	1599.02	1438.14	-	1851.93	87.98	4.5867	4.3747	-	4.7259	0.077
fmax: 0.01 , alpha: 0.9 , csi: 0.9	1610.64	1438.14	-	1797.88	87.03	4.6019	4.4059	-	4.8746	0.082
fmax: 0.02 , alpha: 0.3 , csi: 0.3	1597.30	1438.14	-	1669.34	57.80	4.5699	4.3909	-	4.7229	0.073
fmax: 0.02 , alpha: 0.3 , csi: 0.6	1601.35	1438.14	-	1782.09	60.58	4.5673	4.3903	-	4.7398	0.073
fmax: 0.02 , alpha: 0.3 , csi: 0.9	1589.40	1438.14	-	1791.47	65.11	4.5754	4.3747	-	4.7965	0.086
fmax: 0.02 , alpha: 0.6 , csi: 0.3	1600.85	1438.14	-	1762.59	66.81	4.5776	4.3747	-	4.7809	0.078
fmax: 0.02 , alpha: 0.6 , csi: 0.6	1601.33	1438.14	-	1792.74	70.54	4.5685	4.4215	-	4.7808	0.083
fmax: 0.02 , alpha: 0.6 , csi: 0.9	1591.69	1438.14	-	1774.74	73.51	4.5604	4.4036	-	4.7809	0.074
fmax: 0.02 , alpha: 0.9 , csi: 0.3	1600.07	1438.14	-	1795.81	67.72	4.5734	4.4059	-	4.7965	0.078
fmax: 0.02 , alpha: 0.9 , csi: 0.6	1587.67	1438.14	-	1755.06	70.22	4.5598	4.4059	-	4.8121	0.087
fmax: 0.02 , alpha: 0.9 , csi: 0.9	1589.28	1438.14	-	1712.42	67.41	4.5577	4.4059	-	4.7653	0.071
fmax: 0.03 , alpha: 0.3 , csi: 0.3	1616.11	1438.14	-	1669.34	49.53	4.5430	4.3903	-	4.7965	0.074
fmax: 0.03 , alpha: 0.3 , csi: 0.6	1606.60	1438.14	-	1669.34	58.56	4.5421	4.3747	-	4.7795	0.077
fmax: 0.03 , alpha: 0.3 , csi: 0.9	1599.03	1438.14	-	1669.34	58.68	4.5378	4.3903	-	4.7653	0.074
fmax: 0.03 , alpha: 0.6 , csi: 0.3	1593.60	1438.14	-	1669.34	65.33	4.5390	4.4059	-	4.7965	0.073
fmax: 0.03 , alpha: 0.6 , csi: 0.6	1607.75	1438.14	-	1774.74	68.57	4.5439	4.3903	-	4.9942	0.094
fmax: 0.03 , alpha: 0.6 , csi: 0.9	1592.39	1438.14	-	1669.34	61.60	4.5431	4.4059	-	4.8434	0.079
fmax: 0.03 , alpha: 0.9 , csi: 0.3	1598.20	1438.14	-	1669.34	62.37	4.0134	3.1716	-	#####	1.733
fmax: 0.03 , alpha: 0.9 , csi: 0.6	1599.70	1438.14	-	1669.34	59.07	3.2994	3.2029	-	3.5355	0.057
fmax: 0.03 , alpha: 0.9 , csi: 0.9	1592.06	1438.14	-	1669.34	71.98	3.3094	3.1716	-	3.6716	0.080

Table 14: 3x10 CS Calibration Table

Trial	Solution					Runtime				
	Average	Range			SD	Average	Range			SD
pa: 0	1604.04	1438.14	-	1798.85	91.75	4.0358	3.8122	-	4.4142	0.120
pa: 0.1	1600.84	1438.14	-	1771.45	81.95	4.1144	3.8435	-	5.0704	0.197
pa: 0.2	1599.27	1438.14	-	1807.38	79.19	4.0823	3.8591	-	5.3071	0.186
pa: 0.3	1593.29	1438.14	-	1785.86	85.53	4.5798	3.8747	-	5.7405	0.670
pa: 0.4	1592.69	1438.14	-	1785.03	81.95	4.9496	3.8903	-	5.9058	0.689
pa: 0.5	1603.42	1438.14	-	1781.41	72.78	5.1911	3.9060	-	6.0039	0.628
pa: 0.6	1586.49	1438.14	-	1770.12	72.30	5.2439	3.8900	-	6.2859	0.651
pa: 0.7	1601.94	1438.14	-	1770.87	80.40	5.1360	3.8435	-	6.2736	0.675
pa: 0.8	1597.59	1438.14	-	1787.08	78.73	5.2822	3.9060	-	5.8934	0.573
pa: 0.9	1610.32	1438.14	-	1770.87	60.11	5.4789	3.9841	-	5.8433	0.415
pa: 1.0	1600.92	1438.14	-	1774.74	65.42	5.4591	4.0123	-	5.7652	0.425

Table 15: 3x10 Calibration Results

3x10 Calibration Results												
Algorithm	Solution						Runtime					
	Average ($)	% Error	Range ($)			SD	Average (s)	% Diff	Range (s)			SD
ES	1,438.14	0.0%	1,438.14	-	1,438.14	0.00	158.2720	0.0%	158.0022	-	158.5077	0.172
FFD	1,438.14	0.0%	1,438.14	-	1,438.14	0.00	0.0073	-200.0%	0.0052	-	0.0090	0.001
PSO	1,543.92	7.1%	1,438.14	-	1,770.12	102.62	2.6575	-193.4%	2.5154	-	3.0935	0.094
BA	1,587.67	9.9%	1,438.14	-	1,755.06	70.22	4.5598	-188.8%	4.4059	-	4.8121	0.087
CS	1,586.49	9.8%	1,438.14	-	1,770.12	72.30	5.2439	-187.2%	3.8900	-	6.2859	0.651

Table 16: 3x10 Final Results

3x10 Final Results												
Algorithm	Solution						Runtime					
	Average ($)	% Error	Range ($)			SD	Average (s)	% Diff	Range (s)			SD
ES	14,312.39	0.0%	14,312.39	-	14,312.39	0.00	1304.170	0.0%	1262.385	-	1338.186	23.28
FFD	14,312.39	0.0%	14,312.39	-	14,312.39	0.00	0.018	-200.0%	0.010	-	0.031	0.01
PSO	15,250.06	6.3%	14,312.39	-	17,759.70	1047.87	20.810	-193.7%	18.830	-	23.131	0.64
BA	15,912.75	10.6%	14,312.39	-	16,667.92	652.75	34.703	-189.6%	29.326	-	37.892	2.20
CS	15,846.99	10.2%	14,312.39	-	17,759.70	854.58	42.154	-187.5%	35.605	-	46.101	2.45

Trial 3 shows results similar to those of the other trials, with a larger emphasis on runtime superiority. FFD again has a perfect solution, this time with a runtime five orders of magnitude faster than ES, and three orders of magnitude faster than the PSO, BA, and CS. Metaheuristic algorithm runtimes are two orders of magnitude lower than those of the ES method with average solution errors at or below about 10 percent. PSO again provides a better solution, on average, compared to BA and CS. Since the algorithms defined a set number of iterations for each trial, solution errors can further be reduced by allowing further iterations, and since there is such a wide difference between metaheuristic algorithm and ES runtimes, this would likely be very beneficial. This same relationship holds for the number of particles.

4. Conclusion

This thesis explored the use of heuristic and metaheuristic optimization algorithms to solve the ELD problem with isolated generators. Overall, FFD is the clear winner of all these algorithms in this version of the ELD problem, with significantly faster runtimes with low range and standard deviation, and flawless solutions. In this scenario, load sizes were designed so that all loads can fit into a single generator, thereby allowing the algorithms to evaluate the full range of potential combinations. In other scenarios where all loads cannot perfectly fit into a single generator, FFD may not execute flawless solutions. Nonetheless, it is estimated that solution error would be relatively small, and runtime would be just as fast. Indeed, FFD also demonstrates its ability to scale its runtime better for bigger configuration compared to the other algorithms. In addition, since FFD does not require a configuration profile like the other algorithms, it is even faster and easier to implement the preprocessing steps.

While FFD performed very well, the metaheuristic algorithms also performed well compared to ES, especially at larger configurations. PSO was the clear winner in this application, with better solutions and runtimes about twice as fast as BA and CS. Further research is suggested for tuning PSO and evaluating other PSO hybrid algorithms for even better results. Since few sources could be found comparing BA and CS in the literature search, their comparison was particularly interesting. From the data, little difference in their outcomes was identified. CS was certainly easier to tune with only a single tuning parameter, and while significant time was devoted to meta-optimizing BA's tuning parameters, further research can be made in the future to evaluate all tuning parameters for BA in comparison to CS. All these algorithms demonstrated that generators with higher loadings typically resulted in lower costs.

The feasibility of dimension reduction for this application was demonstrated. There are real world advantages to this strategy in interconnected systems, especially when system configuration and available generators do not change, but load levels do. Further research is suggested to directly compare this low dimension approach with a high dimension approach.

These metaheuristic algorithms show promise in solving grid-level ELD problems where time-varying loads require generator utilization algorithms to be recalculated rapidly for cost savings. In addition, these algorithms can be further augmented by adding other generation equipment like renewable energy sources. As engineers try to incorporate more data and energy sources into the mix of our electricity supply, these algorithms become more and more significant. With proper utilization engineers can save energy, money, and lives.

References

[1] C. Wald and T. Captain, "Energy security: America's best defense," Deloitte Development LLC, 2009.

[2] W. T. Elsayed, Y. G. Hegazy, F. M. Bendary, and M. S. El-bages, "Modified social spider algorithm for solving the economic dispatch problem," *Eng. Sci. Technol, an Int. J.*, vol. 19, no. 4, pp. 1672–1681, 2016.

[3] J. Lin and F. H. Magnago, "Power system economic dispatch," in *Electricity Markets: Theories and Applications*, 1st ed., Hoboken, New Jersey: John Wiley & Sons, Inc., 2017, pp. 119–146.

[4] M. Sudhakaran, P. A.-D.-V. Raj, and T. G. Palanivelu, "Application of particle swarm optimization for economic load dispatch problems," *2007 Int. Conf. Intell. Syst. Appl. to Power Syst.*, pp. 1–7, 2007.

[5] S. K. Nayak, K. R. Krishnanand, B. K. Panigrahi, and P. K. Rout, "Application of artificial bee colony to economic load dispatch problem with ramp rate limits and prohibited operating zones," in *2009 World Congress on Nature and Biologically Inspired Computing, NABIC 2009 - Proceedings*, 2009.

[6] M. N. Nwohu and O. O. Paul, "Evaluation of economic load dispatch problem in power generating stations by the use of ant colony search algorithms," *Int. J. Res. Stud. Electr. Electron. Eng.*, vol. 3, no. 1, pp. 2454–9436, 2017.

[7] S. Khandualo, A. K. Barisal, P. K. Pradhan, and P. K. Patro, "A gravitational search algorithm for solving economic load dispatch problem," *Int. J. Eng. Res. Technol.*, vol. 3, no. 1, pp. 293–297, 2014.

[8] M. Udgir, H. M. Dubey, and M. Pandit, "Gravitational search algorithm: A novel optimization approach for economic load dispatch," *Int. Conf. Microelectron. Commun. Renew. Energy*, pp. 1–6, 2013.

[9] M. Bhoye, S. N. Purohit, I. N. Trivedi, M. H. Pandya, P. Jangir, and N. Jangir, "Energy management of renewable energy sources in a microgrid using cuckoo search algorithm," *IEEE Students' Conf. Electr. Electron. Comput. Sci.*, 2016.

[10] R. Podmore, "Economic power dispatch with line security limits," *IEEE Trans. Power Appar. Syst.*, vol. PAS-93, no. 1, pp. 289–295, 1974.

[11] F. N. Lee and A. M. Breipohl, "Reserve constrained economic dispatch with prohibited operating zones," *IEEE Trans. Power Syst.*, vol. 8, no. 1, pp. 246–254, 1993.

[12] I. T. Kolkata, “Medium scale multi-constraint economic load dispatch using hybrid metaheuristics,” *2017 3rd Int. Conf. Res. Comput. Intell. an Commun. Networks*, pp. 169–173, 2017.

[13] N. Kumar, U. Nangia, and K. B. Sahay, “Economic load dispatch using improved particle swarm optimization algorithms,” *Proc. 6th IEEE Power India Int. Conf. PIICON 2014*, pp. 1–6, 2014.

[14] F. M. Shahir, M. Farsadi, H. Zafari, and A. Sadighmanesh, “Solving economic emission load dispatch problems using particle swarm optimization with smart inertia factor,” *ELECO 2015 - 9th Int. Conf. Electr. Electron. Eng.*, pp. 500–504, 2016.

[15] A. Sharma and S. Vadhera, “Comparative analysis of economic load dispatch using evolutionary and nature based algorithms,” *2017 Int. Conf. Power Embed. Drive Control*, pp. 296–300, 2017.

[16] N. J. Singh, J. S. Dhillon, and D. P. Kothari, “Integrated particle swarm optimization variants for economic load dispatch problem,” *2016 7th India Int. Conf. Power Electron.*, pp. 1–5, 2016.

[17] Y. Wu and J. Guo, “Particle swarm optimization using Lévy distribution for economic load dispatch problem,” *2017 Int. Conf. Ind. Informatics - Comput. Technol. Intell. Technol. Ind. Inf. Integr.*, no. 4, pp. 262–265, 2017.

[18] A. Zaraki and M. F. Bin Othman, “Implementing particle swarm optimization to solve economic load dispatch problem,” *2009 Int. Conf. Soft Comput. Pattern Recognit.*, pp. 60–65, 2009.

[19] S. Gautham and J. Rajamohan, “Economic load dispatch using novel bat algorithm,” *1st IEEE Int. Conf. Power Electron. Intell. Control Energy Syst.*, pp. 1–4, 2016.

[20] G. Pradhan and P. D. Dewangan, “Solving optimal load dispatch problem using enhanced bat optimization algorithm,” *Int. Conf. Innov. Power Adv. Comput. Technol. [i-PACT2017]*, pp. 1–6.

[21] H. Tehzeeb ul Hassan, M. Zunair Zamir, M. Usman Asghar, and H. M. Aamir Faiz, “Economic load dispatch using novel bat algorithm with quantum and mechanical behaviour,” *2017 Int. Symp. Wirel. Syst. Networks*, pp. 1–6, 2017.

[22] J. Tholath Jose, “Economic load dispatch including wind power using bat algorithm,” *2014 Int. Conf. Adv. Electr. Eng.*, pp. 1–4, 2014.

[23] Y. Tonce, K. Priyanto, M. F. Maulana, and A. Giyantara, “Dynamic economic dispatch using chaotic bat algorithm on 150 kV Mahakam power system,” *2017 Int. Semin. Intell. Technol. Its Appl. Dyn.*

[24] G. Yamina Ahlem, B. Hamid, L. Fatiha, and G. Fatima Zohra, “New approach for solving economic load dispatch problem,” *2014 Int. Conf. Electr. Sci. Technol. Maghreb*, pp. 1–5, 2014.

[25] M. K. Duttal A, Das S, Tudu B, “A novel improved algorithm using cuckoo search for economic load dispatch,” *1st IEEE Int. Conf. Power Electron. Intell. Control Energy Syst.* , pp. 1–6, 2016.

[26] S. Sahoo, K. Mahesh Dash, R. C. Prusty, and A. K. Barisal, “Comparative analysis of optimal load dispatch through evolutionary algorithms,” *Ain Shams Eng. J.*, vol. 6, pp. 107–120, 2015.

[27] M. H. Sulaiman and M. R. Mohamed, “Solving economic dispatch problems utilizing cuckoo search algorithm,” in *Proceedings of the 2014 IEEE 8th International Power Engineering and Optimization Conference, PEOCO 2014*, 2014, pp. 89–93.

[28] H. T. Jadhav, S. Raj, and R. Roy, “Solution to economic emission load dispatch problem using modified artificial bee colony algorithm,” in *2013 3rd International Conference on Electric Power and Energy Conversion Systems*, 2013, pp. 1–6.

[29] R. Kaur and G. K. Gill, “Solution to economic load dispatch problem using cuckoo search algorithm,” *Int. J. Electr. Electron. Res.*, vol. 3, no. 2, pp. 362–369, 2015.

[30] T. Trung Nguyen, D. Ngoc Vo, N. Vu Quynh, and L. Van Dai, “Modified cuckoo search algorithm: A novel method to minimize the fuel cost,” *Multidiscip. Digit. Publ. Inst.*, vol. 11, no. 6, pp. 1–27, 2018.

[31] S. Kumar, S. Mehta, and Y. S. Brar, “Solution of economic load dispatch problem using gravitational search algorithm with valve point loading,” *Int. J. Eng. Res. Technol.*, vol. 3, no. 6, pp. 2007–2012, 2014.

[32] M. N. Abdullah, A. H. A. Bakar, N. A. Rahim, H. Mokhlis, and C. Tan, “Implementation of hybrid particle swarm optimization for combined economic-emission load dispatch problem,” *2014 IEEE 8th Int. Power Eng. Optim. Conf.*, vol. 8, no. March, pp. 402–407, 2014.

[33] M. A. Ansari and N. Kardam, “Implementation of particle swarm optimization for dynamic economic load dispatch problem,” *2013 Int. Conf. Energy Effic. Technol. Sustain.*, pp. 1273–1278, 2013.

[34] Z.-L. Gaing, “Constrained dynamic economic dispatch solution using particle swarm optimization,” *Power Eng. Soc. Gen. Meet. 2004. IEEE*, pp. 153–158, 2004.

[35] S. G. Gaurav Kumar Gupta, “Particle swarm intelligence based dynamic economic dispatch with daily load patterns including valve point effect,” *2017 3rd Int. Conf. Cond. Assess. Tech. Electr. Syst.*, pp. 83–87, 2017.

[36] V. Hosseinnezhad and E. Babaei, “Economic load dispatch using PSO and TLBO,” *Int. J. Electr. Power Energy Syst.*, vol. 49, no. 1, pp. 212–219, 2013.

[37] V. K. Jadoun, K. R. Niazi, A. Swarnkar, and N. Gupta, “Variable acceleration coefficient-based particle swarm optimization for non-convex economic load dispatch problem,” *2011 IEEE PES Int. Conf. Innov. Smart Grid Technol. ISGT India 2011*, pp. 126–130, 2011.

[38] B. Ramesh, V. Chandra Jagan Mohan, and V. C. Veera Reddy, “Application of bat algorithm for combimned economic load and emission dispatch,” *Int J Electr Electron Eng Telecommun*, pp. 1–9, 2013.

[39] S. Bhattacharya, B. Ranjan, A. Routray, and A. Dash, “Implementation of bat algorithm in economic dispatch for units with multiple fuels and valve- point effect,” *2017 Int. Conf. Electr. Instrum. Commun. Eng.*, pp. 1–7, 2017.

[40] S. K. Lingala R, Bethina A, Rao P, “Economic load dispatch using heuristic algorithms,” *2015 IEEE Int. WIE Conf. Electr. Comput. Eng.*, pp. 519–522, 2015.

[41] S. Sahoo, K. M. Dash, and A. Kumar Barisal, “Solution of economic load dispatch by evolutionary optimization algorithms- A comparative study,” *2014 Int. Conf. Control. Instrumentation, Energy Commun.*, pp. 259–263, 2014.

[42] D. Sen and P. Acharjee, “Hybridization of cuckoo search algorithm and chemical reaction optimization for economic load dispatch problem,” *2016 Int. Conf. Expo. Electr. Power Eng. (EPE 2016)*, pp. 798–804, 2016.

[43] K. C. Sudhakar A, “Bio inspired algorithms in power system operation: A review,” *2017 Int. Conf. Recent Trends Electr. Electron. Comput. Technol.*, pp. 113–119, 2017.

[44] R. Chellappan and D. Kavitha, “Economic and emission load dispatch using cuckoo search algorithm,” *Int. Conf. Innov. Power Adv. Comput. Technol. [i-PACT2017]*, pp. 1–7, 2017.

[45] T. Sukmadi, A. Dwi Wardhana, and M. Agus Riyadi, “Optimization of gas turbine power plant economic dispatch using cuckoo search algorithm method,” *4th Int. Conf. Inf. Tech., Comput. Electr. Eng.*, pp. 131–135, 2017.

[46] Q. X. Hua P, Zuo-fang L, “Economic dispatch of power systems including electric vehicle and wind farm,” *2017 IEEE Conf. Energy Internet Energy Syst. Integr.*, pp. 1–5, 2017.

[47] Y. A. Gherbi, H. Bouzeboudja, and F. Lakdja, “Hybrid metaheuristic for the combined economic-emission dispatch problem,” in *12th International Symposium on Programming and Systems, ISPS 2015*, 2015, pp. 1–7.

[48] H. Zhang and Q. Hui, “Cooperative bat searching algorithm: A combined perspective from multiagent coordination and swarm intelligence,” in *13th IEEE Conference on Automation Science and Engineering (CASE)*, 2017, pp. 1362–1367.

[49] C. C. Meher K, Swain R, “An analysis of dynamic economic dispatch using search space reduction based gravitational search algorithm,” *Int. J. Energy Power Eng.*, vol. 10, no. 3, pp. 452–459, 2016.

[50] I. A. Farhat and M. E. El-Hawary, “Mult-objective economic-emission optimal load dispatch using bacterial foraging algorithm,” *25th IEEE Can. Conf. Electr. Comput. Eng.*, pp. 1–5, 2012.

[51] A. Y. Saber and G. K. Venayagamoorthy, “Economic load dispatch using bacterial foraging technique with particle swarm optimization biased evolution,” *IEEE Swarm Intell. Symp. St. Louis MO USA*, pp. 1–8, 2008.

[52] A. Ranjan, P. Khargonekar, and S. Sahni, “Offline first fit scheduling in smart grids,” in *Proceedings - IEEE Symposium on Computers and Communications (ISCC)*, 2016, pp. 758–763.

[53] B. Rieck, “Basic analysis of bin-packing heuristics,” *Publ. por Interdiscip. Cent. Sci. Comput.*, pp. 1–9, 2010.

[54] D. S. Johnson, “Fast algorithms for bin packing,” *J. Comput. Syst. Sci.*, vol. 8, pp. 272–314, 1974.

[55] N. Ntene and J. H. van Vuuren, “A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem,” *Discret. Optim.*, vol. 6, pp. 174–188, 2009.

[56] J. Kennedy and R. Eberhart, “Particle swarm optimization,” *Neural Networks, 1995. Proceedings., IEEE Int. Conf.*, vol. 4, pp. 1942–1948, 1995.

[57] X.-S. Yang and S. Deb, “Cuckoo search via Lévy flights,” *2009 World Congr. Nat. Biol. Inspired Comput.*, pp. 210–214, 2009.

[58] X. S. Yang, “A new metaheuristic bat-inspired algorithm,” in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, 2010, pp. 65–74.