Characterizing the Use of Heuristic Optimization Methods for Renewable Energy Systems Design

Chris Sharp[1], Caitlyn Clark[1], Annalise Miller[1], Vincenzo Ferrero[1], Marine Bentivoglio[1], and Mohamadmedhi Ebrahimi[1]
Oregon State University, Corvallis OR, 97331, USA

and

Bryony DuPont[2]
Oregon State University, Corvallis OR, 97331, USA

The objective of this research is to characterize, predict, and improve the performance of complex renewable energy systems using advanced computational design automation methods. While research in the area of optimization – particularly layout optimization – of renewable energy systems has been explored for many years, a synergistic approach that determines *a priori* the ideal optimization methods to apply, and why these methods are well-suited, has yet to be explored. The application of multiple heuristic optimization approaches is tested on two novel renewable energy systems design problems: floating offshore wind energy systems, and wave energy converter systems. These systems are optimized to maximize power development and minimize system costs. Characterizing heuristic optimization algorithms as applied to these test problems implicates ideal methods for breaking down barriers to implementation of renewable energy systems. Additionally, these characterizations lend to a better understanding of how heuristic optimization methods can be subsequently advanced or combined to optimize computational expense.

I. Introduction

Heuristic optimization methods are designed for solving complex systems problems, in that they are able to search large solution spaces effectively and in less time, and can be applied to NP-Hard problems that are unsolvable using traditional, exact optimization methods such as steepest descent [1,2]. Heuristic methods are also capable of optimizing systems represented by nonlinear, non-convex modeling and objective formulations, and can simultaneously optimize discrete and continuous variables, rendering them ideal for approaching accuracy in real-world systems such as power grids [3]. While not guaranteeing global solution-finding as with exact methods, heuristic optimization methods find near-optimal solutions and conduct expansive searches of large solutions spaces [4]. As such, many heuristic optimization methods have been developed and tested in the last twenty years and have been applied to a host of test problems and real-world complex systems.

In this work, we characterize different heuristic optimization methods (including genetic algorithms [5,6], simulated annealing algorithms [7,8], particle swarm optimization [9], extended pattern search [10–13], and evolutionary algorithms) specifically for relative solution optimality, ability to accommodate multidisciplinary (and likely competing) objectives, ability to exploit parallel processing capabilities, and decreased memory use. Moreover, as energy systems require modeling on multiple time scales (for example: oncoming wind is measured on the order of minutes; grid integration and planning on the order of hours or days; device life on the order of years), it is imperative that algorithms are characterized by their ability to accommodate multi-scale models and objectives.

II. Methodology

Heuristic optimization methods are often chosen empirically based on previous experience by the system designer. One recent advancement is the development of collections of heuristic optimization codes made available for use online [14]. A recent study of these frameworks indicated 33 online repositories for such methods, 11 of which were

[1] Graduate Research Assistant, School of Mechanical, Industrial, and Manufacturing Engineering.
[2] Assistant Professor, Mechanical Engineering, AIAA Professional Member.

highlighted for their breadth and applicability [14]. These frameworks facilitate the widespread use of heuristic methods and also try to overcome the effort required to adapt heuristic methods to specific complex systems optimization problems [15]. Two examples of these frameworks, the Opt4J framework [1] and the FOM: A Framework for Metaheuristic Optimization [15], include relevant heuristic optimization methods (e.g. genetic algorithms, simulated annealing, and particle swarm optimization.) and exact methods (e.g. linear programming) in an effort to formalize the applicability of optimization methods with respect to *Exploration vs. Exploitation*. That is, certain methods are ideal for wide search of the solution space (*Explorative, or Diversifying*) while others are designed to perform localized search (*Exploitative, or Intensifying*) [16].

The goal of this work is to characterize popular heuristic optimization methods that can be applied to two specific renewable energy systems: floating offshore wind energy systems and wave energy converter systems. These methods are chosen for their breadth of characteristics, specifically for the characteristics shown in Table 1 [16]. These algorithms include: evolutionary algorithms, genetic algorithms [5,6], simulated annealing algorithms [7,8], particle swarm optimization [9], and extended pattern search [10–13].

Table 1. Characteristics for Algorithm Selection

1	Metaheuristic vs. Analytical
2	Exploration vs. Exploitation
3	Population-based vs. Single Point
4	Dynamic vs. Static Objective
5	One vs. Various Neighborhood Structures
6	Order of the Algorithm

To characterize these algorithms, we explore (1) the ability for the algorithms to converge on a globally optimal solution, (2) the quantity of function evaluations necessary to achieve optimality or convergence, (3) the ability for the algorithm to readily accommodate the multiple, competing objectives of power development and system costs, (4) the ability for the algorithm to accommodate the multiscale temporal modeling considered necessary for realism in power systems optimization (while it is common for generator data to be on the order of minutes or hours, global systems optimization approaches must consider the entire lifetime of the system). Each of these algorithms are described in the following subsections.

A. Evolutionary Algorithm (EA)

Evolutionary Algorithms (EAs) are a class of search algorithms that are often used as function optimizers for static objective functions [17]. EAs are principally a stochastic search and optimization method based on the principles of natural biological evolution. Compared to traditional optimization methods, such as calculus-based and enumerative strategies, EAs are robust, global, and may be applied generally without recourse to domain-specific heuristics [18]. Genetic qualities influence the chance of an individual's survival in that the struggle to live leads to a natural selection or "survival of the fittest", and genetic variants that have proven to be well adapted to the environmental conditions appear preferably in subsequent generations [19].

Two main concepts of evolutionary algorithms are discussed in the literature: genetic algorithms [6] and evolution strategies [20]. There are many similarities between these two methods, but they have some key differences as well. The most important difference comes from the crossover process that occurs in a genetic algorithm. Both methods do include mutation and the mutation plays an important role in the implementation of an EA. The mutation operator randomly alters part of a parent solution to produce an offspring that is mostly like the parent, but with a small amount of variation [21].

EAs operate on a population of potential solutions, applying the principle of survival of the fittest to produce successively better approximations to a solution [17]. Each generation seeks to improve the solutions in the way that eliminates the weak answers according to their level of fitness and keeps the stronger answers for mutation in the next generation. To initialize an Evolutionary Algorithm, the first step is to generate an initial population of parents, p. A row vector, as demonstrated in Figure 1, represents each parent. An initial array of dimension p x (n x m) is generated in which N devices (denoted by cylinders) are inserted in each parent's vector based on randomly generated values ranging from 1 to n x m. At each of these randomly generated locations, the cell is assigned a value of "1," while every

other cell is assigned a value of "0." Hence, the initial parents grid of p x (n x m) cells with each row containing N devices is created, and is defined as the first generation.

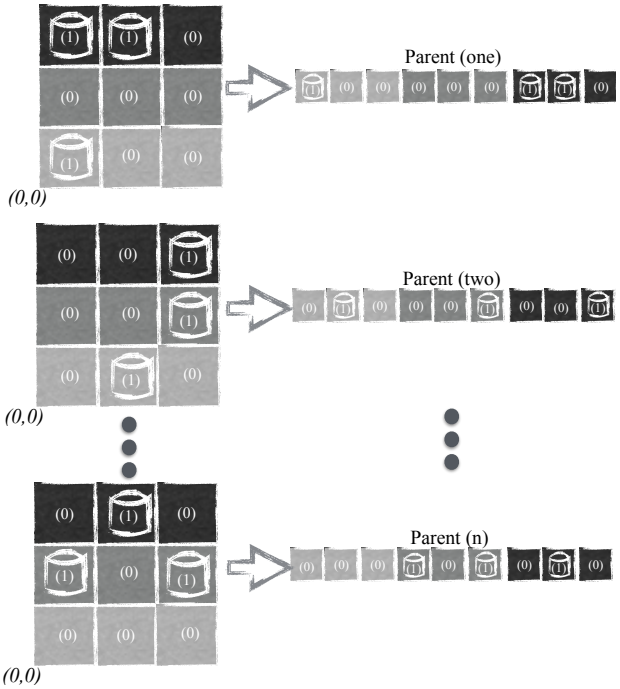

Figure 1. Example of Converting the Discretized Physical Grid Space to Row Vector Format

Once generated, this parent generation is evaluated based on the objective function presented in Eq. 2. Mutation is then applied to the parent array – a defined number of random cells are selected for mutation in each row (parent) of the sorted population. If a "1" cell is subject to mutation, it is swapped with a randomly selected "0" cell in the same vector to maintain the number of devices. Similarly, if a "0" cell is subject to mutation in a vector, it is then swapped with a randomly selected "1" cell in the same vector in order to maintain the number of devices. Once all the parent solutions have been mutated, the newly obtained children array is evaluated with the objective function. After evaluation, the children population is combined with the parent population and this combined pool is then sorted according to the objective function evaluations of each potential solution. The new parent generation is obtained by selecting the p best individuals from the combined solution array. Since the objective function in this project is defined by the ratio of cost to power, the goal is to minimize the objective function, thus minimizing the cost while maximizing the power. Therefore, as long as the objective function of the newly obtained generation is superior to the objective function of the best parents' generation on record, the process continues to search through mutated generations to find an optimum objective function and layout until a set number of generations has elapsed without improvement. On the other hand, whenever a generation obtains a solution with a lower objective function it is recorded as the overall best layout. The process continues as long as better solutions are being found or until a number of generations has elapsed without finding a better solution.

B. Genetic Algorithm (GA)

Related to the EA, the genetic algorithm (GA) is a method based on the evolutionary process in which traits are acquired through the implementation of mutation. However, unlike EAs, GAs also mimic the way that chromosomes are passed down from parent generations to child generations. This process is completed through the use of a crossover process in which pairs of parents are "mated" and pairs of children consequently made using traits from both parents. In this manner, the created children solutions are comprised of information from both parent solutions. GAs work well for array optimization problems due to their ability in handling the many discrete and continuous factors that affect the layout while efficiently converging to an optimal configuration and calculating an overarching objective function.

In the genetic algorithm method, the initial parent population is created in the same manner as the evolutionary algorithm; however, the manner in which the children population is generated has distinct differences. The general flow of the GA used for this work is shown in Figure 2.

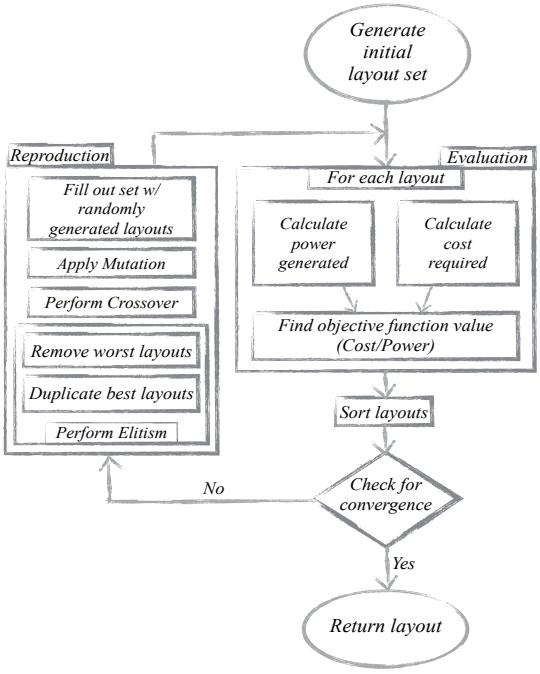

Figure 2. Genetic Algorithm Flowchart

With the initial parents generated, evaluated, and sorted, children are created using elitism, crossover, mutation, and random layout generation. Each of these facets is a tunable parameter that can be adjusted in order to help the GA converge upon a solution. For the GA, elitism involves cloning the best solutions from the parent set into the children set. This means that a set amount of the children set will be identical to the best of the parents. Additionally, the same percentage of parents that are cloned are killed off and that percentage of the children set is populated with random solutions to ensure that the GA can fully explore the solution space and avoid getting stuck in local minima.

After the Elitism operation is complete, crossover is performed on a set upper percentage of the parent population. The parents that were cloned are included in this crossover population to allow the current best solutions more influence on the propagated solutions. The crossover is performed by randomly swapping a set fraction of devices between parents being mated. For example, if the fourth device from parent one is chosen to be moved, then it would be removed from parent one and a device would be placed in parent two at the location from which it was moved in parent one. This new layout, comprised primarily of parent one, but with the variation introduced from parent two, constitutes one of the two children to be created. The second child solution is made by using the same process – by randomly choosing a device from parent two and moving it to parent one. To ensure that the defined number of devices is maintained, devices from both parents is also removed. In short, moving a random device or devices in each parent to a new location or locations that is acquired randomly from the other parent makes the children solutions. Figure 3 demonstrates this process.

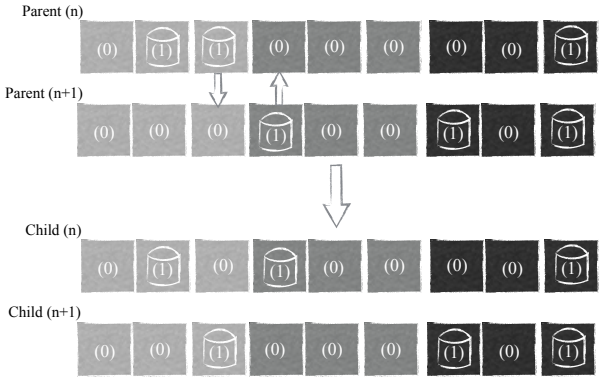

Figure 3. Example of Crossover Process

After the crossover procedure is complete then this portion of the children population is mutated. Mutation allows the method to explore the local solution space around a proposed layout with less randomness than through the introduction of new completely random solutions in elitism. The method involves randomly moving a device in a layout based on a set percentage.

With elitism, crossover, and mutation complete, the objective function of each new layout is found. Then this newly generated children set is sorted and checked for convergence. If attained the algorithm reports the converged solution, but if not, then the children set becomes the next parent set and the process is repeated. Convergence is defined as a prescribed upper percentage of the children set reporting the exact same layouts.

C. Simulated Annealing Algorithm

Simulated Annealing (SA) is a powerful optimization technique, proposed in 1983 by Kirkpatrick et al. [8], because of its ability to converge upon very good solutions for difficult combinatorial optimization problems, while easily dealing with complex nonlinear constraints [22]. Simulated annealing takes advantage of the analogy between the minimization of an optimization problem's cost function and the slow procedure of gradually cooling a metal until it reaches its "freezing" point [8,23,24]. Based on the iterative method proposed by Metropolis et al., this system simulates the performance of atoms in equilibrium at different temperatures while cooling [7].

Less optimal solutions are more likely to be accepted in the early iterations of the algorithm, because of the high temperature; however, by cooling down the temperature, the successors move towards selection of only better solutions. One of the critical parameters used in simulated annealing algorithm is the temperature and the rate at which it decreases.

The final method applied to the WEC array optimization problem is a simulated annealing approach. To implement the SA method, the initialization of the space for WEC placement is very similar to that described in the EA and GA sections prior, except that for the SA the grid is not converted to row vector format.

In the SA method, bad solutions may be accepted at the beginning of the process. The probability of accepting the bad solutions in the algorithm is based on the difference between power generated in the previous and current solutions, and the iteration number. The objective function is found in the same manner as the EA and the GA using Eq. 1.

Implementation of this procedure depends on the temperature and its reduction or cooling rate parameters, which must be tuned based on the nature of the specific problem. The probability of accepting a worse solution is calculated at each epoch using on these numbers.

To implement the SA method, a single randomly generated layout is created by inserting a set number of WECs into discretized locations in a grid. At each iteration, a device is randomly chosen for repositioning to one of the eight potential surrounding cells – assuming that the potential cell is in the allowable space and that no device already exists there. The new location of the chosen device is randomly selected from the potential eight locations. The objective function for the proposed new layout is calculated and compared against the objective function of the current layout. If the proposed position is better than the current position, the proposed position is accepted. However, if the proposed position is not better than the current position, it may still be accepted based on a determined probability.

$$P = e^{-(|\Delta E|/T)} \tag{1}$$

where P is the probability of acceptance, ΔE is the difference between the objective function evaluation of the proposed solution and the overall best solution, and T is the temperature value. The basic algorithm for simulated annealing is the following:

1. Generate a random solution
2. Set initial temperature T = Tl
3. Calculate its cost using a defined cost function
4. Generate a random neighboring solution
5. Calculate the new solution's cost
6. Compare the new solution with the previous solution:
   a. If $c_{new} < c_{old}$: move to the new solution
   b. If $c_{new} > c_{old}$: maybe move to the new solution with the probability of $P = e^{-(\Delta C/T)}$
7. Update the temperature, T
8. Repeat steps 3-7 above until an acceptable solution is found or you reach some maximum number of iterations.

Over the duration of the algorithm the best result is recorded and updated.

D. Particle Swarm Optimization (PSO)

We begin the Particle Swarm Optimization (PSO) with 100 randomly generated layouts in compliance with all constraints. We then identify the swarm's best layout as the initial layout with the lowest objective evaluation. We assign each layout a random initial velocity in the x and y direction between 0 and 2 m. In every iteration, we determine the velocity of individual turbines using Eq. 2.

$$v_{ij,k} = v_{ij,k-1} + C_1 r_1 \left(d_{ij,k-1} - d_{ij,best}\right) + C_2 r_2 \left(d_{ij,k-1} - d_{i,swarmbest}\right) \tag{2}$$

In this function, v is velocity, d is current location, i is the individual turbine number, j is the layout number in the swarm, and k is the iteration. $d_{21,best}$ indicates the location of device 2 in layout 1 that is correlated with the lowest objective evaluation in the history of layout 1, while $d_{2,swarmbest}$ indicates the location of device 2 that is correlated with the lowest objective evaluation in any layout in the swarm's history. $C1$ and $C2$ are weights given to the layout's best and swarm's best performances. In this model, we use $C1 = C2 = 0.001$. While a value of 0.001 is very low for a typical PSO, we discovered that limiting the velocity of turbines resulted in better outcomes that did not violate constraints.

Rather than explicitly enforce spatial constraints on the devices, we used a penalty factor to discourage layouts that violated constraints. The objective evaluation used in this PSO is given in Eq. 3:

$$objective = \frac{Cost_{total}}{Power_{total}} \left(1 + \sum Constraint\_Violations\right) \tag{3}$$

In this function *Constraint Violation* is the sum of constraint violations for every turbine in the layout. For turbines outside the 2000 *m* bounding area, the *Constraint Violation* is the shortest distance from the turbine's location to the bounding area. For devices within 200 *m* of another turbine, the *Constraint Violation* is 200 *m* minus the distance between the turbines in meters.

After each movement, the algorithm updates the best layouts for each layout number and the swarm as a whole. This process continues until 100 iterations have been completed without an improvement in the swarm's best objective function. The best layout with the lowest evaluation in the swarm's history is returned.

E. Extended Pattern Search (EPS)

The Extended Pattern Search Algorithm (EPS) focuses on improvements to a single layout rather than a population. First, we generate a layout without any constraint violations. The algorithm then randomly cycles through each device in a solution. A device is moved first in the -*y* direction one initial step size. If the objective evaluation of the entire layout improves and no constraints are violated, the move is kept. Otherwise the device returns to its original position and attempts moves at the initial step size in the -*x*, +*y*, and +*x* directions until a move is kept. If the device in question is located in the back (downstream) half of the field, the algorithm attempts the search directions in the opposite order.

When no device can accept a move at the initial step size, the EPS begins popping poor performing devices to new locations. The popping algorithm applies sequentially to the 5 worst performing devices. If the evaluation of the entire

field improves and no constraints are violated, the new location is kept and the next device is popped. Each device is allowed 100 popping attempts per iteration. If no pops are accepted, the algorithm continues onto the next device.

After popping the worst performing devices, the EPS begins cycling through individual devices again at one half the original step size. The algorithm terminates when no moves can be found at the minimum step size of 3 m.

III. Case Studies

Two case studies are presented: (1) floating offshore wind systems optimization, and (2) wave energy converter array optimization. In completing these case studies, we employ algorithms that utilize the same cost-per-kilowatt objective function to determine the predicted performance of each energy system. Our overarching objective in completing this work is to identify which algorithms find the most optimal solutions, while doing so at minimal computational expense, measured in computational runtime and number of objective function evaluations:

$$minimize\ F_1: Computational\ Time\ t \qquad \{t \in \mathbb{R} \mid t \geq 0\} \tag{4}$$

$$and/or\ F_1: Number\ of\ Function\ Evaluations\ n \qquad \{n \in \mathbb{Z} \mid n \geq 0\} \tag{5}$$

1) In the floating offshore wind systems optimization work, selected optimization algorithms (Genetic Algorithm Particle Swarm Optimization, and Extended Pattern Search) will select turbine platform locations in a square space with side lengths equaling two kilometers. We perform optimization under unidirectional, constant wind speed conditions with a surface roughness representative open sea conditions. A novel cost model specifically designed to predict the costs of installing and maintaining floating offshore wind farms is used [25]. The decision variables are the (*x,y*) locations of the WECs, and the placement of the devices is constrained to a selected ocean area.

2) Wave energy converter (WEC) array design can be an efficient solution for power maximization and cost reduction, as higher energy may be produced by WECs integrated into arrays than by the same number of isolated WECs. The radiated and scattered waves resulting from the interaction between incident waves and WECs has shown to amplify the interaction factor q, implying that these devices develop more power in array scenarios than the sum of comparable devices operating in isolation. In this case study, we explore the local placement of WECs in arrays using the objective of maximizing power development and minimizing cost; the decision variables are the (*x,y*) locations of the WECs, and the placement of the devices is constrained to a selected ocean area. We compare the use of a Genetic Algorithm, Evolutionary Algorithm, and a Simulated Annealing algorithm.

IV. Results and Discussion

(1) Floating Offshore Wind System Optimization

To date, we have characterized the use of an extended pattern search, a binary genetic algorithm, and a particle swarm optimization method as applied to the floating offshore wind systems optimization problem. In this work, wind turbines are optimized in a square space with side lengths equaling two thousand kilometers. We perform optimization under unidirectional, constant wind speed conditions with a surface roughness of $z0$=0.0005 meters for open sea conditions [6]. We use a power law exponent, αh of 0.11 [2, 7] and assume a farm life of 20-years. Our farm water depth is set at 200 meters, and the distance from shore is set at 32 kilometers. The floating offshore wind farm optimization parameters are summarized in Table 2. Variation in objective function evaluation with respect to the number of turbine platforms located in the space is given in Figure 4.

Table 2: Problem Formulation, Case Study 1

Parameter	Offshore Scenario
Side Length	2000 m
Water Depth	200 m
Life of farm	20 years
Distance from Shore	32 km
Surface Roughness	0.0005 m
Wind Speed	10 m/s
Power Law Exponent	0.11
Number of Popping Attempts	1000
Number of Popped Turbines	5
Turbine Cut-in Speed	3 m/s
Turbine Rated Speed	11.5 m/s

In this work we compare layouts containing 5 to 30 turbines. We run each optimization algorithm five times for each number of turbines. Using the data from all three optimization techniques, we estimate the optimal number of turbines as that which returns the lowest cost-per-kilowatt-power. The best layouts from each individual optimization technique are also presented.

The results of our optimization of floating offshore wind farms composed of 5 to 30 turbines are displayed below. Figure 4 shows the final objective evaluation against the number of turbines optimized and fit with a fifth-degree polynomial through the points collected. The approximate run time of each of these algorithms as applied to this test problem, again with respect to the number of turbine platforms whose locations are being optimized, is given in Figure 5. The minimum objective evaluation by each optimization algorithm is summarized in Table 3. The EPS returned the lowest objective evaluation with the highest number of turbines. This layout is shown in Figure 6, where blue triangles represent turbines and dashed lines represent the boundary of turbine wakes.

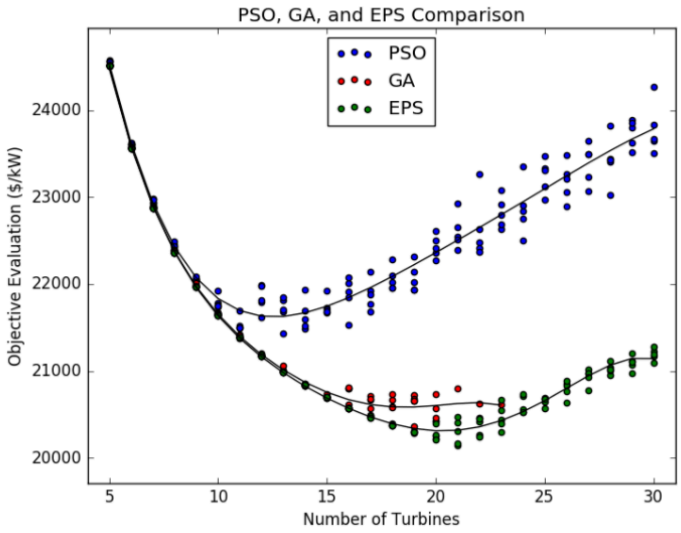

Figure 4: Results of algorithm performance for Case Study 1

Table 3. Algorithm Performance for Case Study 1:Offshore Wind Farm Layout Optimization

Optimization Technique	Number of Turbines	Objective ($/kW)
Genetic Algorithm	19	20366
Particle Swarm Optimization	13	21434
Extended Pattern Search	22	20146

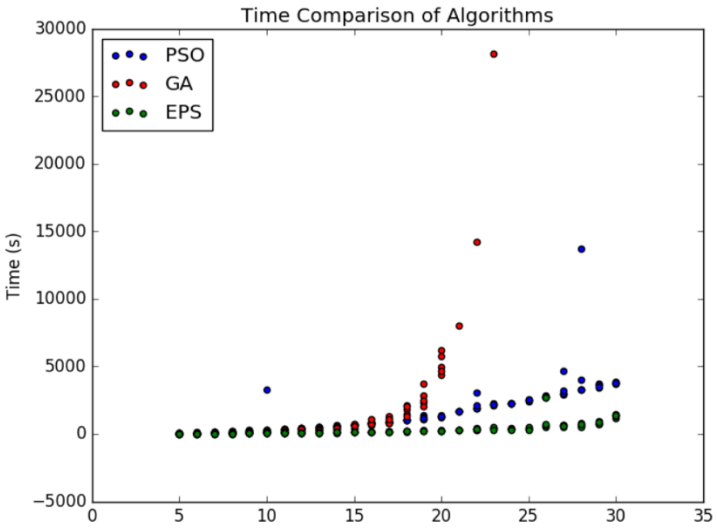

Figure 5. Approximate Algorithm Run time

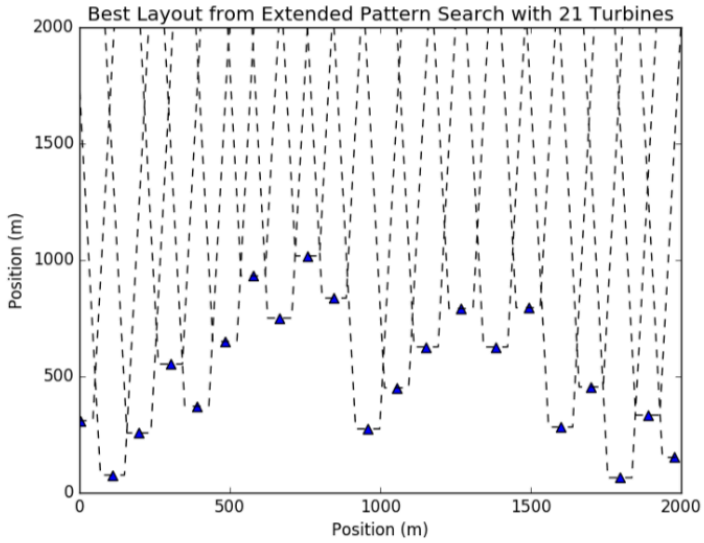

Figure 6: EPS-Derived Optimal Layout, Case Study 1

Figure 4 shows that the PSO resulted in the least optimal layouts with fewer turbines. This is most likely due to inability of the PSO to deal explicitly with the constraints given in the problem. The penalty function used in the PSO applied to the objective of the entire field, and may have been improved by targeting poorly behaving turbines.

Both the GA and the EPS provided similarly optimal layouts, although the EPS performed slightly better. Since both of these techniques preformed well, we shifted the criteria of judgment to the second objective of this work; what is the best method to implement for floating offshore wind farms based on farm performance and computational expense. As shown in Figure 5, the time expenditure of the GA starts to increase exponentially around 20 turbines, while the EPS takes a similar amount of time to run regardless of the number of turbines being optimized. It is possible that the EPS runs more quickly than the GA because it requires less computer memory. Because we require 18 list items for each turbine in a layout and 100 layouts per generation, the GA has an large memory demand. The EPS may run the fastest because it deals with improving a single layout instead of generating 100s of new layouts like the GA. In this sense, it the EPS trades a smaller memory requirement for more iterations.

(2) Wave Energy Converter Array Optimization

A set of studies exploring the application of a genetic algorithm and a simulated annealing approach (compared to an evolutionary algorithm) have been conducted for WEC array optimization. In this study, arrays of 5, 10 and 25 devices are optimized using these three algorithms. Multiple objective functions (including the baseline objective of maximizing power while minimizing cost) have been explored, including maximizing the positive hydrodynamic interactions between devices. WECs are placed in a 60-meter by 60-meter space with a grid resolution of 10 x 10. This resolution equates to 100 different cells in which a WEC can exist. Each algorithm uses a heave-constrained truncated cylinder with a radius of one meter and a draft (distance below the water surface) of one meter. Given the discretized space, the minimum separation distance allowed between WECs must be defined. For the work presented here the minimum separation is deliberately set at six meters based on previous research's defined minimal distance of three times the body's diameter [26]. The wave climate is drawn from a Bretschneider spectrum with a significant wave height of 2-meters and a modal frequency of 0.2 Hertz. The water depth is set at 8-meters with unidirectional waves coming directly from the west. Each algorithm is run a total of ten times and the objective function, interaction factor, and number of function evaluations recorded for the result of each run.

In Table 4, the baseline objective evaluation for 10 and 25 devices are shown; in both cases, the genetic algorithm outperforms the simulated annealing method. However, when exploring the interaction factor as a system objective (representing the positive hydrodynamic interaction between devices), it is clear that the ability of each algorithm to converge on an optimized result (in this case, an interaction factor greater than one) is dependent on the number of devices in the array (Figure 8). All of the layouts generated are unique between the three algorithms. Figure 7 and Figure 9 show the configurations which yield the best objective function between the three algorithms. Again, these layouts were generated by the GA. These interaction factors indicate that the formation of WECs shown in Figures 7 and 8 produce 2.1% more power with 10 devices and 0.46% more power with 25 devices than the combined power that would be produced by equivalent number of devices acting in isolation.

The results from the mean analysis of the objective functions obtained with the three algorithms show that the GA is also more robust at finding a minimized objective function evaluation, while the SA has the worst average result. This further exploration demonstrates that the GA is more efficient when used in an increased search space. In fact, it shows that the GA is globally the best performing algorithm among the three compared, while SA has the worst performance for all numbers of devices considered. Figure 9 portrays the interaction factor results of the different algorithms.

Table 4. Comparison of Optimized 10- and 25-WEC Arrays

	10 Devices - Objective Function		
	Mean	**Algorithm Best**	**Overall Best**
Evolutionary Algorithm	*3.0321*	*3.0287*	*3.0180*
Genetic Algorithm	*3.0222*	*3.0180*	
Simulated Annealing Algorithm	*3.0330*	*3.0237*	
	25 Devices - Objective Function		
	Mean	**Algorithm Best**	**Overall Best**
Evolutionary Algorithm	*2.3162*	*2.3042*	*2.2741*
Genetic Algorithm	*2.2794*	*2.2741*	
Simulated Annealing Algorithm	*2.3388*	*2.3315*	

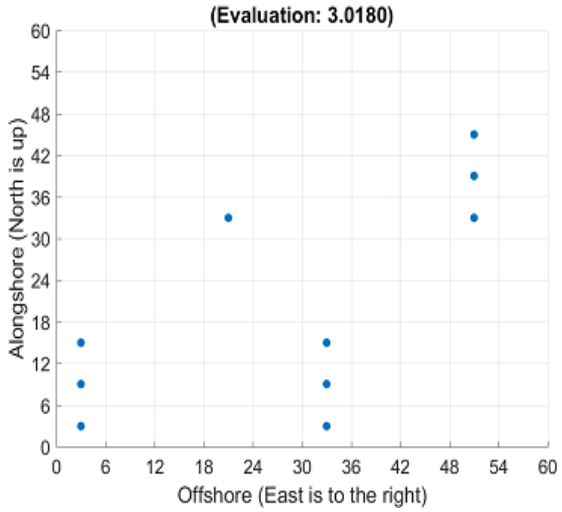

Figure 7: Best Result, Case Study 2, 10 WECs

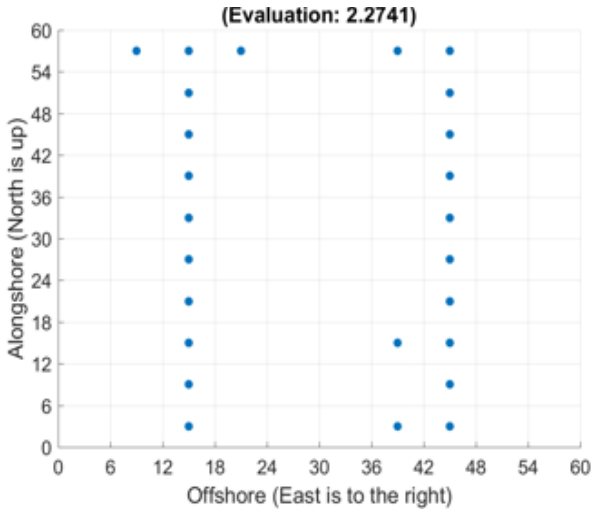

Figure 9: Best Result, Case Study 2, 25 WECs

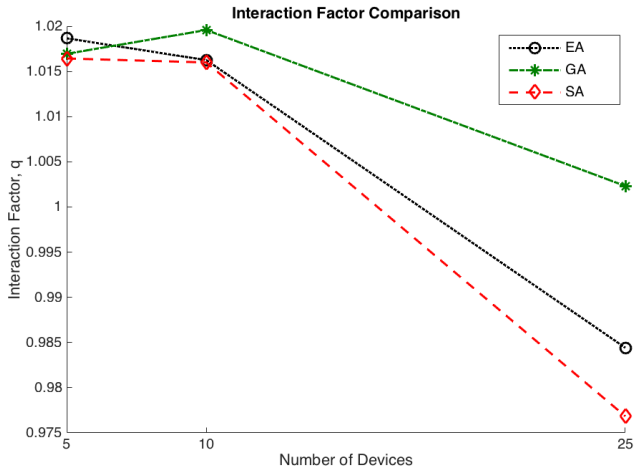

Figure 8. Interaction Factor of Optimized WEC arrays with respect to the number of WECs in the array

V. Conclusion

In this study, we characterized the performance of multiple heuristic optimization methods as applied to two renewable energy system case studies: floating offshore wind farm optimization, and wave energy converter array optimization. Algorithms were selected for variation in six characteristics: (1) all algorithms were (meta) heuristic in nature, (2) exploration vs. exploitation, (3) population-based vs. single point, (4) dynamic vs. static objective, (5) one vs. various neighborhood structures, and (6) the order of the algorithm. The heuristic algorithms we included are evolutionary algorithms, genetic algorithms, simulated annealing algorithms, particle swarm optimization, and extended pattern search algorithms.

In both case studies, the design variables are the locations of the devices, and the primary objective function is the simultaneous reduction of system cost and the maximization of system power. In the first case study (floating offshore wind farm optimization), a genetic algorithm, an extended pattern algorithm, and a particle swarm optimization are studied. The extended pattern search algorithm was most capable of designing an optimal floating offshore wind farm,

resulting in a layout with 22 turbines and the lowest run time compared to the other tested algorithms. In the second case study (wave energy converter array optimization), the genetic algorithm provided superior layouts as compared to an evolutionary algorithm and a simuated annealing algorithm, even while testing both a cost/power objective and an interaction factor objective. Future work will include further classification of these algorithms, and wider application to renewable energy systems.

References

[1] Lukasiewycz, M., Glaß, M., Reimann, F., and Teich, J., "Opt4J: A Modular Framework for Meta-heuristic Optimization," *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, 2011, pp. 1723–1730.

[2] Stephenson, M., Amarasinghe, S., Martin, M., and O'Reilly, U.-M., "Meta Optimization: Improving Compiler Heuristics with Machine Learning," *ACM Conference on Programming Language Design and Implementation*, 2003, pp. 77–90.

[3] Fesanghary, M., and Ardehali, M. M., "A novel meta-heuristic optimization methodology for solving various types of economic dispatch problem," *Energy*, vol. 34, 2009, pp. 757–766.

[4] Cooper, L., "Heuristic Methods for Location-Allocation Problems," *SIAM Review*, vol. 6, 1964, pp. 37–53.

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

[6] Goldberg, D. E., *Genetic Algorithms in Search Optimization and Machine Learning*, Addison Wesley, 1989.

[7] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E., "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, 1953, pp. 1087–1092.

[8] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by simulated annealing," *Science*, vol. 220, 1983, p. 671.

[9] Kennedy, J., and Eberhart, R., "Particle swarm optimization," *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, 1995, pp. 1942–1948 vol.4.

[10] Torczon, V., "On the convergence of pattern search algorithms," *SIAM Journal on Optimization*, vol. 7, 1997, pp. 1–25.

[11] Torczon, V., William, C., and Trosset, M. W., "From Evolutionary Operation to Parallel Direct Search : Pattern Search Algorithms for Numerical Optimization," *Computing Science and Statistics*, 1998, pp. 396–401.

[12] Kolda, T. G., Lewis, R. M., and Torczon, V., "Optimization by Direct Search : New Perspectives on Some Classical and Modern Methods ∗," *Search*, vol. 45, 2003, pp. 385–482.

[13] Hooke, R., and Jeeves, T. a., ""Direct Search'' Solution of Numerical and Statistical Problems," *Journal of the ACM*, vol. 8, 1961, pp. 212–229.

[14] Parejo, J. A., Ruiz-Cortés, A., Lozano, S., and Fernandez, P., "Metaheuristic optimization frameworks: a survey and benchmarking," *Soft Computing*, vol. 16, 2011, pp. 527–561.

[15] Parejo, J., Racero, J., and Guerrero, F., "FOM: A framework for metaheuristic optimization," *ICCS*, 2003, pp. 886–895.

[16] Blum, C., and Roli, A., "Metaheuristics in combinatorial optimization: Overview and Conceptual Comparison," *ACM Computing Surveys*, vol. 35, 2003, pp. 268–308.

[17] Droste, S., Jansen, T., and Wegener, I., "On the analysis of the (1+1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, 2002, pp. 51–81.

[18] Zhang, G., Gheorghe, M., and Wu, C., "A Quantum-Inspired Evolutionary Algorithm Based on P systems for a Class of Combinatorial Optimization."

[19] Baumert, T., Brixner, T., Seyfried, V., Strehle, M., and Gerber, G., "Rapid communication Femtosecond Pulse Shaping by an Evolutionary Algorithm with Feedback," *Applied Physics B: Lasers and Optics*, vol. 782, 1997, pp. 779–782.

[20] Schwefel, H.-P. P., *Evolution and Optimum Seeking: The Sixth Generation*, New York, NY, USA: John Wiley & Sons, Inc., 1993.

[21] Zhang, Q., Sun, J., and Tsang, E., "An Evolutionary Algorithm With Guided Mutation for the Maximum Clique Problem," *IEEE Transactions on Evolutionary Computation*, vol. 9, 2005, pp. 192–200.

[22] Simopoulos, D. N., Kavatza, S. D., and Vournas, C. D., "Reliability Constrained Unit Commitment Using Simulated Annealing," *Power Systems, IEEE Transactions on*, vol. 21, 2006, pp. 1699–1706.

[23] Laarhoven, P. J. M., and Aarts, E. H. L., *Simulated Annealing: Theory and Applications*, Norwell, MA, USA: Kluwer Academic Publishers, 19887.
[24] Simopoulos, D. N., Kavatza, S. D., and Vournas, C. D., "Unit Commitment by an Enhanced Simulated Annealing Algorithm," *IEEE Transactions on Power Systems*, vol. 21, 2006, pp. 68–76.
[25] Miller, A., Forinash, C., and DuPont, B., "An Extended Pattern Search Approach for Optimizing Floating Offshore Wind Farms," *Journal of Wind Engineering and Industrial Aerodynamics*, 2017, p. Under Review.
[26] Child, B. F. M., Cruz, J., and Livingstone, M., "The development of a tool for optimising arrays of wave energy converters," *Proceedings of the Ninth European Wave and Tidal Energy Conference, EWTEC.*, 2011, pp. 1–11.