UNCLASSIFIED: Distribution Statement A. Approved for Public Release
2014 NDIA GROUND VEHICLE SYSTEMS ENGINEERING & TECHNOLOGY SYMPOSIUM
Modeling & Simulation, Testing and Validation (MSTV) Technical Session
August 12-14, 2014 - Novi, Michigan

Simulation of Microgrid and Mobile Power Transfer System interaction using Distributed Multiobjective Evolutionary Algorithms

Andrew G. Dunn
Jeremy B. Mange, PhD
Annette G. Skowronska
David J. Gorsich, PhD
US Army TARDEC
Warren, MI

Vijitashwa Pandey, PhD
Zissimos P. Mourelatos, PhD
Mechanical Engineering Department
Oakland University
Rochester, MI

Abstract

Optimization of a microgrid interacting with mobile power transfer systems is a multiobjective problem that grows to be computationally expensive as components and fidelity are added to the simulation. In previous work [17] we proposed an optimization strategy relying on evolutionary computing. With an evolutionary computing approach, seeking a well-distributed set of solutions on the entire optimal frontier necessitates a large population and frequent evaluation of the aforementioned simulation. With these challenges, and inspiration from Roy et al. [14] distributed pool architecture, we propose an architecture for distributed pool evolutionary computing that differs from the Roy et al. design. We use this strategy with a microgrid and mobile power transfer system simulation to optimize for cost and relaibility. We find that the distributed approach achieves increased performance in raw system execution time, and in some cases converges faster than a non distributed version of the same evolutionary strategy.

Introduction

Ground systems are achieving greater levels of electrification. Conventional systems such as propulsion, protection, and communications will likely continue to be primary stakeholders in the increased capability. However, integration with external systems that use this capability is increasing. How ground systems will support external electrical systems will be an important consideration for design and configured deployment. When ground systems link with other systems to share power sources and loads a microgrid is formed. Microgrids by nature can/will operate without a connection to utility. In some environments, specifically military, the logistics burden of less than efficient power allocation becomes significantly large.

In an earlier publication [17], we presented how ground systems may be thought of as mobile power transfer systems that can link together and form major components of a microgrid. This prior work demonstrated that the theory of repairable systems could be applied, along with a minimal set of metrics, to a framework that managed the interaction of a microgrid and mobile power transfer systems. We will use the mobile power transfer systems (MPTS) terminology hereon to generally describe ground systems with source and load capabilities. The MPTS were dynamically treated as power loads or sources at different times, and the overall management of those systems together became a complex optimization problem trying to minimize cost and maximize reliability. Simulating a microgrid and MPTS interaction becomes computationally expensive as you add complexity and granularity to the model.

Our model, discussed in greater detail within, is multiobjective, and as such there is a need to generate an optimal frontier of solutions. Evolutionary computation and, more specifically genetic algorithms, provide for a mechanism to explore a population of solutions that, after many generations can represent the optimal frontier. In our prior publication [17], we chose to use the Nondominated Sorting Genetic Algorithm (NSGA-II) introduced by Deb and Pratap in [7]. The optimization was implemented in a single threaded fashion and it became quickly apparent that increased complexity, or fidelity, of the model would take a great deal of run-time. As multiprocessing systems are now ubiquitously available, we searched for distributed strategies for evolutionary computing. Several models have been proposed, one in particular by Roy et al. [14] introduced the concept of distributed evolutionary computations interacting with a single population pool. The strategy is described as focused on network deployment, such as a supercomuputing environment, with measures for fault tolerance and heterogenity between the instances of the evolutionary computation(s).

We have implemented an intepretation of the Roy et al. [14] design. The NSGA-II strategy that was used in our prior work [17] along with reliability metrics from [12] are all implemented in a microgrid and mobile power transfer system simulation. We will:

- Describe prerequisite material on Evolutionary Computation, Multiobjective optimization, and NSGA-II as to provide context for our pool framework design choices.
- Describe in detail the ground system with microgrid simualtion that serves as an evaluation function for our optimization.
- Describe the proposed distributed pool framework and draw comparisons between our implementation and the inspiring work of Deb et al. [8] and Roy et al. [14].
- Describe results comparing a non-pool NSGA-II implementation to the distributed pool, as well as different pool parameters and effects of scaling

We do not claim any fundamental breakthroughs in enhancing the efficiency of a particular evolutionary algorithm via reducing the fundamental complexity. However, through the pairing of existing, efficient evolutionary algorithms with a distributed pool framework, we can realize a reduced need for simulation evaluations and speedup in computational cost (through distribution). This approach is most specifically warranted when the simulation is computationally costly and one does not have exclusive access to a distributed computing environment. The speedup, in turn, allows for a more detailed and realistic model to bring the results of the simulations closer to real-world phenomena.

Evolutionary Computation

The origins of evolutionary computation can be traced as far back as the 1950s ([3], [2]), with a increasing interest through the 1970s ([10], Rechenberg [13]). At first there were limitations in both methodology and available computing power, however as these limitations waned, interest in evolutionary computing increased. Evolutionary computation mimics the process of natural evolution, the driving process for the emergence of complex and well-adapted organic structures [1]. Put succinctly and with strong simplification, evolution is the result of interplay between creation of new genetic information and its evaluation and selection [1]. The better an individual performs under evaluation and selection, the greater the chance for the individual to live longer and generate offspring. Over the course of evolution, this leads to a penetration of the population with genetic information of individuals of above-average evaluation [1]. Evolutionary computing can be thought of more as a problem solving strategy than a one-size-fits-all tool [1], [11]. In keeping with the biological analog, and to support processes such as survival and reproduction, we need to define the concept of the *individual*.

An *individual* is a representation of a *candidate solution*, which in turn is a set of properties that are representations of the solution domain. These properties are selected in a problem-specific manner and are typically called the *genotype* of the *individual*. The *genotype* properties, in keeping with the biological analog, can be mutated or altered by evolutionary operators. As two or more *individuals* need to be compared to determine which is “better”, it is necessary to also have some measure of “goodness”, which in evolutionary computation literature is known as the *fitness* of an individual [1]. The *fitness* is a measure of how well one *candidate solution* solves a problem. The *fitness function* maps the *individuals genotype* (solution space) into a corresponding *phenotype* (fitness space). In evolutionary computing the *fitness function* is the representation of the desired problem to be solved, in this context it is a simulation of mobile power transfer systems and microgrid interaction (discussed in the next section).

Once the representation is chosen, the evolutionary operators must be specified. These operators define the mechanisms of *selection*, *variation*, and *replacement* [1]. *Selection* determines how *candidate solutions* are chosen to participate in the creation of new *candidate solutions*

[9]. *Variation* determines how new *candidate solutions* are created from existing *candidate solutions*, and how existing *candidate solutions* are modified to explore new areas of the solution space [9]. *Replacement* determines how *candidate solutions* are chosen to remain viable for *selection* [9].

It is helpful to view the execution of a genetic algorithm as a two stage process. It starts with the *current population*. *Selection* is applied to the *current population* to create an *intermediate population*. Then variation and replacement are applied to the *intermediate population* to create the *next population*. The process of going from the *current population* to the *next population* constitutes one generation in the execution of an evolutionary computation [19]. Below we use the aforementioned terminology to defined a general evolutionary computation, where P_t represents the population at time t.

Algorithm 1 Evolutionary Computation

```
1: $t = 0$
2: Initialize($P_t$)
3: Evaluate($P_t$)
4: while termination condition not met do
5:     parents = Selection($P_t$)
6:     offspring = Variation($P_t$)
7:     Evaluate(offspring)
8:     $P_{t+1}$ = Replacement($P_t \cup$ offspring )
9:     t += 1
10: end while
```

An important concept to grasp is that the number of function evaluations is critical when comparing two evolutionary computation algorithms [9]. A function evaluation is simply one mapping of *individual genotype* to *fitness/phenotype*. The measure of fitness evaluations is the typically processing unit in evolutionary computing literature when discussing the efficiencies of a particular algorithm, for example [7].

In our terminology above we worked to define concepts of *population*, *individual*, *fitness* and *evolutionary operators* without discussion of termination criteria. In our example algorithm the generational process is repeated until a termination condition has been reached. Termination can be defined in a problem specific manner, however there are many common termination conditions such as; discovery of a *candidate solution* that satisfies minimum criteria, number of function evaluations, and number of generations.

Bäck et al. [1] states the majority of current evolutionary computation implementations come from three different, but strongly related, approaches: *genetic algorithms*, *evolutionary programming*, and *evolutionary strategies*. We will focus on *genetic algorithms* with NSGA-II, however each of the mentioned approaches can be implemented with variations of representation and the evolutionary operators generally introduced above. We will use these conceptual building blocks to describe the used evolutionary algorithm and distributed pool framework.

With *genetic algorithms* in general it is desired to have the fitness function evaluated relatively fast. As genetic algorithms work with a population of *candidate solutions*, there is an incurred cost for evaluating each member of the population. Furthermore, the *population* is replaced (all or in part) on a generational basis, meaning you incur this cost at a potentially high frequency [19]. The speed in which our simulation can be evaluated, or the nature of how we distribute the evaluation becomes very important for bounding what we're able to explore in terms of simulation or *candidate solution* granularity.

Simulation of MPTS with Microgrid

The modeling and simulation portion of this work involves a modeled microgrid and multiple mobile power transfer systems, with simulated operation over a specified timeframe. Four main categories of components are included: generators (power sources), loads (power draws, both static and dynamic), contactors (control switches for the power circuits for generators and loads), and vehicles, which can dynamically act as either power sources or power loads, depending on their state and the state of the microgrid as a whole. The reliability of each instance of each type of these components is characterized by a mean time between failure (MTBF) value, and simulations involve Monte Carlo runs over the specified time period with dynamic control logic and stochastic failures introduced based on the MTBF values estimated from a previous physical implementation of the system being modeled (discussed in [17]).

The control of the microgrid, and particularly of the addition or removal of vehicle systems as either power sources or loads, is modeled in terms of four control variables:

- **Source shed** - the point above which the relative power must rise before a power source is removed
- **Source replace** - the point below which the relative power must drop before an available power source is added
- **Load shed** - the point above which the relative power must rise before a power load will be removed

- **Load replace** - the point below which the relative power must drop before an available power load will be added

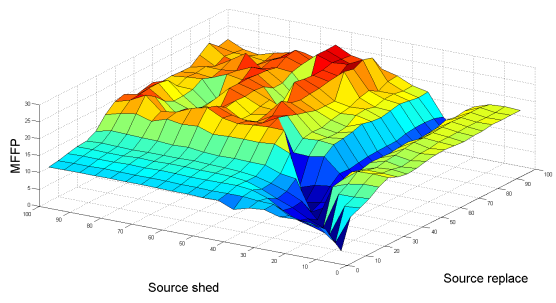

Figure 1: MFFP output for a limited range of two input values

These simulations track three main parameters to characterize the resultant behavior over the simulated time period: **cost**, **number of failures**, and **minimum failure-free period (MFFP)** [12]. In this context, a failure is defined as an instance in which a power load did not receive the amount of power it required, either because it was not available or because the dynamic control strategy did not provide it. The dynamic costs associated with the operation of different generators at different supplied power loads are also estimated based on the previous work with this system ([17]). Since these simulations are Monte Carlo simulations, for a particular scenario, multiple simulations are run, and the resulting values are averaged for a more accurate set of characteristic outputs.

Within the simulation, the interactions between the dynamic control strategy parameters and the output values are very complex. For instance, the figure 1 on page 4 shows the output values for MFFP for a limited set of input values for just two of the dynamic control parameter variables (with the other two held constant). Even this very limited slice of the domain search space reveals a complex interaction. This complexity is what necessitates an optimization strategy to successfully find optimal dynamic control strategy parameters for a given model.

These inputs and outputs, then, define the optimization task for the microgrid simulation: find optimal dynamic control parameters that minimize cost and number of failures and maximize MFFP. Because there are multiple outputs, this is a multi-objective optimization task, the ultimate result of which is a Pareto frontier with the number of dimensions equal to the number of objectives; in this case, three.

Multiple Objectives & NSGA-II

For our simulation, as an optimization problem, there does not exist a single solution (genotype) that simultaneously optimizes each objective (phenotype). In this case the objective functions (simulation) are said to be conflicting and there exists a set of Pareto optimal solutions. A solution is called non-dominated or Pareto optimal if none of the objective functions can be improved in value without degrading some of the other objective values.

The first application of evolutionary computation to multiobjective optimization was Schaffer's Vector Evaluated Genetic Algorithm (VEGA) [16], [4], [9]. Since then, evolutionary multiobjective optimization has been an incredibly active area of research [4]. This is due to the natural union of evolutionary computing and multiobjective optimization, which stems from the fact that evolutionary computing algorithms are generally very good optimizers and they work simultaneously on a set of candidate solutions [4], [9]. As multiobjective optimization problems require a set of solutions (Pareto Optimal), rather than a single solution, the final population from an evolutionary computation algorithm provides just such a set [9].

There are novel ways to implement evolutionary operators on multiobjective *Individuals*. With single objective optimization the concept of "better" can be expressed simply as an inequality. Where as multiobjective evolutionary operators have, by necessity, more complex strategies for defining "better". Our prior optimization [17] was using an implementation of the NSGA-II algorithm introduced by Deb and Pratap in [7]. The novel advancement by Deb and Patap was to use a newly proposed *fast non-dominated sort*, *fast crowd distance estimation*, and *simple crowd comparison* which resulted in a complexity improvment from $O(MN^3)$ to $O(MN^2)$ and the elimination of a tuning parameter. Using the terminology introduced in the Evolution Computing section above we have the context to understand the novel aspects of NSGA-II.

In the Algorithm on page 3 we described Evolutionary Computation in general. In literature this is typically referred to as the canonical *simple genetic algorithm* [9], [18]. The NSGA-II algorithm can primarily be though of as an implementation of a replacement strategy that is called on line 9. In the following algorithm we will make sure of a static function notation that operates on data within the context, below there is a description corresponding to the notation used in [7].

Where Sort corresponds to *fast non-dominated sort*,

Algorithm 2 NSGA-II as EC Replacement

```
Require: $P_t$, offspring
1: combined = $P_t \cup$ offspring
2: fronts = Sort(combined)
3: for front in fronts do
4:     Distance(front)
5:     survivors = survivors + front
6: end for
7: Comparison(survivors)
8: return survivors(0 to Length($P_t$))
```

Distance corresponds to *fast crowd distance estimation*, and Comparison corresponds to*simple crowd comparison*, we describe simplified implementation details from [7]:

- **Sort** - (Line 2) Bins the *Individuals* into "fronts" based on how the number of other *Individuals* are dominated by the solution.
- **Distance** - (Line 4) assigns a distance metric to each *Individual* based on a composite "distance" for each objective fitness
- **Comparison** - (Line 7) Provides a mechanism for comparing *Individuals* using both rank and distance to achieve a more uniformly spread Pareto frontier

We mention the details of this strategy within the context of the Replacement notion mentioned earlier due to its use in the distributed pool architecture. It is important to realize that this replacement strategy is given the parents and offspring, and returns a set of *Individuals* to match the population size of the parents. As the prior text and algorithm is a significant simplification of the publication [7], we would encourage independent investigation (specifically the diagrams within [7]).

Distributed Evolutionary Algorithms

The act of optimization requires many evaluations of *candidate solutions* and comparisons of their assigned *fitness* on a potentially grand scale. Deb et al. [8], [6] dicuss how the number of objectives significantly influences the dimension of the true Pareto optimial front. We consider that was we introduce objectives to our simulation, there will be increased burdon to explore the Pareto frontier.

Furthermore Deb et al. goes on to state that when the task is to find a well-distributed set of solutions on the entire Pareto optimal front, the population size of the evolutionary computation is necessarily large [8]. These challenges led to proposed models for distributed evolutionary computation. In [8] there is a review of three models of distributed computing introduced by [5], along with new suggested methods for distributing multiobjective evolutionary computation.

One of the three models is interesting in our case. The master-slave model is a very simple form of adding distribution to evolutionary computing by having a single master evolutionary algorithm with many slave processors that are used exclusively for evaluation. This became a baseline objective for our study, to distribute the execution of the evaluation function across multiple processes or networked nodes. During implementation we searched for other strategies that may provide a benefit greater than the simple master-slave model.

Roy et al. in [14] and [15] proposes a novel approach to distribution where the population pool is shared amongst several autonomous evolutionary computations. In the pool architecture there are multiple workers, each running a copy of the evolutionary computation. For Roy's implementation the protocol was designed to support heterogeneity between worker capabilities and potential worker failure modes. As this work served to be significant inspiration for our approach we will re-convey the design in detail to later draw comparisons. For consistency we will use the term *worker* instead of the original *processor*.

> There are $p \geq 1$ participating workers. Each participating worker runs a sequential evolutionary computation with a population of size u. There is a common pool $\mathcal{P}$ of individuals of size $n > u$. Each individual in the pool is stored in a shared data structure, which can be accessed concurrently by multiple processors. In more detail, $\mathcal{P}$ is partitioned into $\mathcal{P}_1, \ldots, \mathcal{P}_p$. Each partition $\mathcal{P}_k (1 \leq k \leq p)$ is a collection of single-writer (written by processor k) shared variables where each shared variable holds an individual of the evolutionary computation.
>
> There are two basic operations performed on $\mathcal{P}$ by any participating processor: *ReadIn* and $WriteOut$. The *ReadIn* operations performed on $\mathcal{P}$ by processor k picks u individuals uniformly at random from $\mathcal{P}$ and copies them into k's local data structure P_k. The $WriteOut$ operation performed on $\mathcal{P}$ by processor k writes back the individuals in P_k to the portion of $\mathcal{P}$ that is allotted to k. Here, in order to ensure convergence of the evolutionary computation, an element of elitism is applied, i.e. the individual i in P_k replaces an individual j in $\mathcal{P}_k$ only if i is fitter than j.

In [14] the *worker* is only running a single generation in between *ReadIn* and *WriteOut*. Our next section will outline our proposed framework in detail and enumerate differences in design choice.

Distributed Pool Architecure

Using the work of Deb et al. [7], [8] and Roy et al. [14] as inspiration we constructed a network distributed implementation of the NSGA-II evolutionary algorithm. We defined our framework in terms of *Pool* and *Workers*, where the is a single *Pool* and n participating *Workers*.

Although we use the same basic *ReadIn* procedure described in Roy et al. [14], there are a few distinctions between their approach and ours. First, our pool $\mathcal{P}$ is not partitioned into the $\mathcal{P}, ..., \mathcal{P}_p$ sub-structures. Rather, when the u individuals are uniformly randomly chosen during the *ReadIn* (or, in our terminology, *Pick*) procedure, the indices in $\mathcal{P}$ corresponding to these chosen individuals are recorded as $k_1, k_2, ..., k_u$. Then, during the *WriteOut* (or, in our terminology, *Replace*) operation, the new individuals produced by the genetic operations run by processor k are re-written to these same $k_1, k_2, ..., k_u$ locations within $\mathcal{P}$, utilizing the NSGA-II replacement strategy.

We implemented a "credit" based protocol between the *Pool* and *Workers* where, upon request, a *Worker* is granted "credit" to perform a task. *Worker* nodes connect to the pool through a request/reply messaging scheme. The request/reply communication is blocking for both the *Worker* and the *Pool*. This ensures that the *Worker* process will block upon placing a request into the *Pool*, waiting for a reply. The protocol shared between *Worker* and *Pool* is based on the following messages:

- **init** - The initial connection of a *Worker* to the *Pool*. At this time the *Worker* will have no task payload to deliver. At this time the *Pool* will perform a *Pick* and hand back to the worker a set of *Individuals* as well as their respective *population* indices to be operated upon.
- **task** - The *Worker* delivers back a task payload to the *Pool*, this is comprised of the evolved population with the respective indicies. From this the *Pool* executes a *Replace*, then a subsequent *Pick* to hand the *Worke* another task.
- **termination** - At each time a *Worker* submits a task to the *Pool*, the internal termination criteria is checked. If met, the *Pool* returns a termination message to the *Worker* (instead of a task payload), then waits for a period of time for potential *Workers* to deliver the termination news (however eventually terminating itself, stranding workers who have been away for too long)

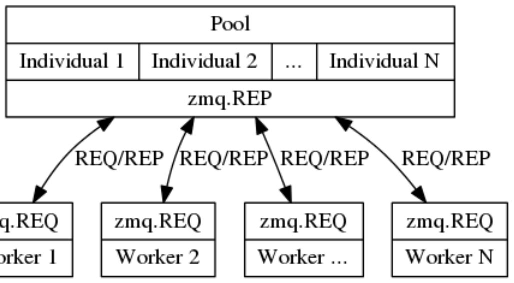

Figure 2: Messaging Topology (Utilizing ØMQ)

The two functions of the *Pool* are:

- *Pick*(size) - Selects a parametrized number of *Individuals* from the *Pool* population. This selection is done in a uniformly random fashion. The response message is both the instance of the *Individuals* selected, and their respective indices.
- *Replace*(indices, offspring) - Builds up the current instances of the parent population from the supplied indices, then does an NSGA-II style Replacement between the parents (existing *Pool Individuals*) and the offspring.

Above we mention only one implementation of *Pick* and *Replace*. The Roy implementation chose to use a uniformly random selection for *Pick*, which we also implement. There are many potential implementations for *Replace* that will affect the *Pool* population. For example:

- *simple worker domination strategy* - when the textit-Worker submits its evolved population to the textit-Pool, the evolved population always dominates the picked indices (No comparison is made).
- *partitioned worker domination strategy* - mimics the design by [14] by partitioning the *Pool* up initially by the expected amount of *Workers*. When the *Worker* submits its evolved population to the pool it is copied over the partition allocated for the respective *Worker*.
- *NSGA-II replacement strategy* - uses an NSGA-II style of replacement using the indices of the *Pool* population as parents and the *Worker* evolved population as offspring.

For our implementation we chose to use an *NSGA-II replacement strategy* to facilitate dynamic allocation of participating *Workers* with the *Pool* instance. With a *partitioned worker domination strategy* the *Pool* would need a more complex protocol for interaction, or have a fixed set of initial participating *Worker*s.

The termination criteria for the *Pool* currently are limited to number of evaluations or number of generations. The number of generations is a fractional calculation due to the *Worker* being limited through the *Pick* parameter to a subset of the overall population.

Just as in the work of Roy et al. [14], we are able to be fault tolerant to *Worker* failure. The *Pool* has no expectations that a *Worker* will return from being assigned a task. Failure of a *Worker* in this sense is seamlessly tolerated as the *Pool* only increments and checks the termination criteria when a *Worker* returns a completed task. This would allows for the *Pool* to embrace heterogeneous (speed/physical architecture) *Workers*, without need to manage synchronization of any kind. Furthermore, the "credit" based protocol provides flexibility for changing the number of *Workers* during the simulation. This would be particularly useful in a time-shared environment where computational resources may become more or less plentiful during the entirety of the simulation.

In a time-shared supercomputing environment one would typically rely on message passing through the MPI standard, we choose to use ØMQ (ZeroMQ) asynchronous messaging library due to its adoption in modern Linux distributions and extensive use in elastic hardware environments. Through ØMQ and the Python language bindings we are able to serialize python objects directly, which is how we're passing all messages within the system. In our implementation we currently have the evolutionary algorithm declared within the *Worker* object, however, pending the complexity of its serialization we ideally would like to implement the evolutionary algorithm at the *Pool* level, which reduces the *Worker* implementation complexity significantly. This technique would also facilitate tuning of the individual evolutionary algorithm parameters dynamically during a simulation.

Finally, we want to introduce the available configuration parameters for both the *Pool* and *Worker* as they are used to tune performance in the following section:

Pool	single instance
uri	network location
population	*Pool* population size
pick	*Worker Pick* parameter
termination	# of evaluations or generations
Worker	n instances
uri	network location of *Pool*
termination	# of evaluations or generations

Approach & Results

We implemented the distributed pool architecture along with the microgrid and MPTS simulation, seeking to optimize for cost, number of failures, and minimum failure-free period (MFFP). We used an implementation of NSGA-II that, aside from the replacement strategy outlined above, used a tournament style selection along with mutation and crossover. We established a reference implementation of the Non Pool NSGA-II that was single threaded and, optionally, multi-threaded.

Overall, the distributed optimization algorithm showed vast improvement over the previous algorithm in terms of simulation speed, and some additional improvement in terms of optimal parameter convergence, both of which are explored in detail in the following sections. These improvements are of significant importance for future research in this area, both for the actual calculation of dynamic power control parameters for systems of this type, and for detailed modeling and simulation of more complex microgrid systems, including the types of multi-context vehicle-grid interactions modeled by the current simulation.

Execution Speed

The most immediate improvement of the distributed algorithm described in this paper is the vastly lower execution speed for equivalent simulation work. This improvement holds when comparing the numbers of simulations run or the time required to reach a specific value for any of the output objectives. Figure 3 shows a typical example of this phenomenon, with the time required to reach various minimum levels of cost shown, comparing the non-distributed algorithm to the distributed pool algorithm. The distributed pool speedup is shown with only 8 processors, however, the flexibility in the design and network protocol allows for an arbitrary number of processors and thus even greater speedups.

This improvement is perhaps the most significant for the overall objectives of this project, because the vast speed improvement allows for more complex and granular simulations of both the microgrid environment and the mobile power transfer systems of interest. These more advanced models and simulations were some of the main motivating goals of this work.

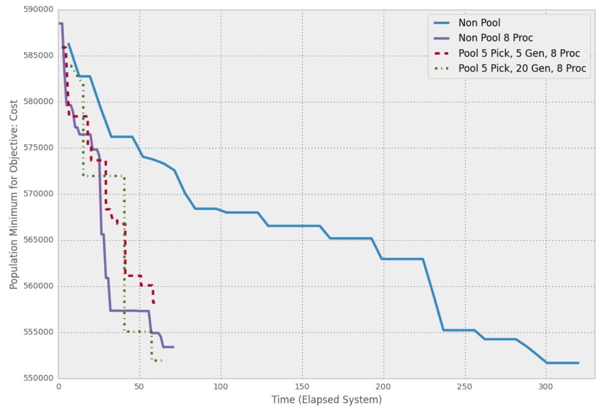

Figure 3: 5000 Evaluations, System time with Population minimum cost

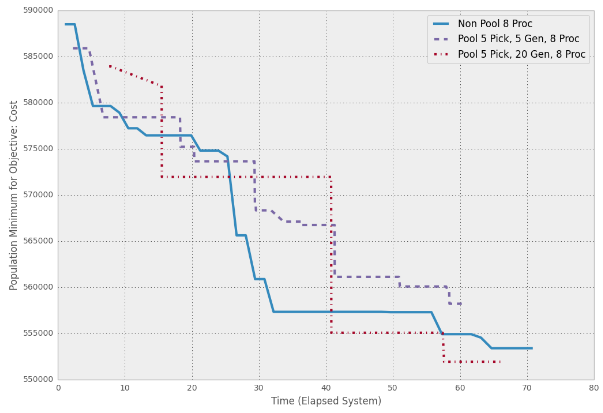

Figure 4: 5000 Evaluations, all multiprocessor implementations

We have fixed the population size to be 100 and the termination criteria to be 5000 evaluations for both the distributed pool algorithm and the non distributed algorithm. Both *Pool*(s) are configured with a *Pick* size of 5, where in one we use a have the *Worker* process 5 generations, and the other we have the *Worker* process 20 generations. Below we show the same graph, however we've removed the non multi-threaded results to display more clearly the evaluation time.

The initial assignment of values for the population *candidate selection* is random, however we have seeded the pseudo random number generator with the same key for each of the four compared simulations. We were excited to realize that the execution time of the pool, on a very low latency network, was less than the multiprocessing counterpart. This excitement is tempered with knowledge that any meta-heuristic type search algorithm that has random components in its underlying operation needs statistics based performance comparison before difinitive claims can be made.

Convergence

The rate of optimization algorithm convergence is typically measured by the number of objective function evaluations necessary to reach a particular value for one or many of the outputs (objectives). As mentioned previously, for this microgrid control optimization task, an evaluation of the objective function involves specifying dynamic control strategy parameters, performing many Monte Carlo simulation runs over a specified time period, and averaging the results of these runs for final values of cost, number of failures, and minimum free failure period (MFFP).

Although the results in this area are somewhat complex because of the many possible parameter sets for the distributed pool itself, in most cases the new distributed algorithm showed increased convergence speed for all of the outputs with some set or sets of parameters. The figures below show the results of a comparison between the original NSGA-II algorithm and the new distributed pool algorithm (with various parameter sets) with respect to each of the three outputs.

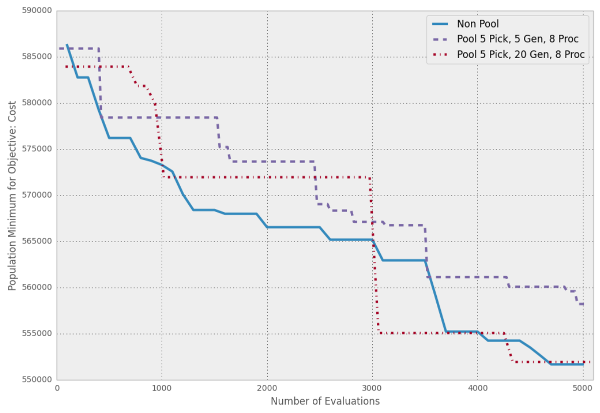

Figure 5: Comparison of algorithms with respect to cost

Figure 5 displays these results with respect to cost, showing the current-best value for each algorithm as the number of objective function evaluations increases.

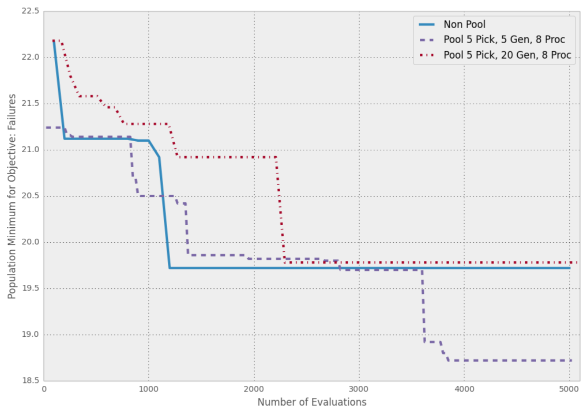

Figure 6: Comparison of algorithms with respect to number of failures

Figure 6 displays these results with respect to number of failures, again showing the current-best value for each algorithm as the number of objective function evaluations increases.

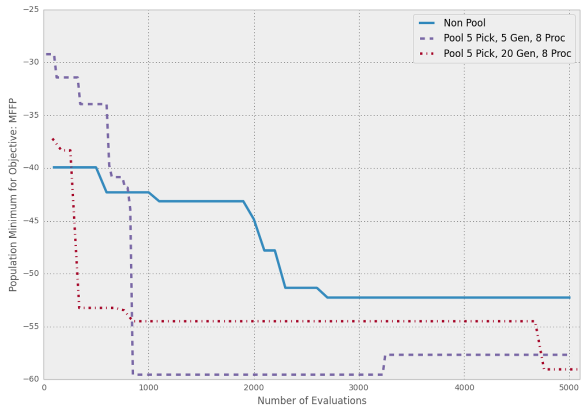

Figure 7: Comparison of algorithms with respect to MFFP

Figure 7 displays these results with respect to MFFP, once again showing the current-best value for each algorithm as the number of objective function evaluations increases. Note that because MFFP is a maximization parameter, for the sake of consistency, the inverse of MFFP is minimized. In this parameter the most significant convergence advantage can be seen, as both of the displayed distributed pool algorithms calculated better MFFP values for almost all numbers of evaluations. This, to reemphasize, is in addition to the vast speed gains already discussed.

Pareto Frontier

Figure 8 shows an output Pareto frontier for two of the objectives, number of failures and cost. Two of the points, p_1 and p_2, are highlighted on the figure to illustrate the trade-off inherent in the Pareto frontier, and to clarify how the optimization algorithm and results thereof are tied back to the microgrid simulation.

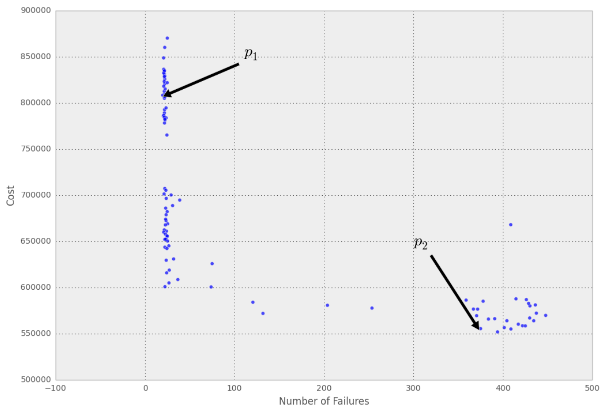

Figure 8: Pareto Frontier

p_1 has a total cost of 807602 with 20.2419 associated failures, whereas p_2 has a total cost of 554386 with 375.161 associated failures. The tradeoff between these is straightforward, and Pareto frontiers are generally used by decision-makers who asses the relative risks and values of each objective. In this case, if deciding between these two specific options, a decision-maker might choose whether cost or number failures was a more significant objective, and choose p_2 or p_1, respectively. Once this selection was made on the basis of the objectives involved, the dynamic control parameters that produced those objective values could be used for the control of the actual microgrid.

Note that there are actually three output objective parameters involved in this simulation. However, for the sake of simplicity of demonstration, only two are shown in the figure. The actual tradeoff analysis for the real-life problem being modeled would involve all of these objective aspects.

Conclusions and Future Work

As discussed in the Results section, the selection of parameters within the distributed pool itself in this distributed architecture has a significant impact on the results of the optimization. Although a complete examination of ideal parameter selection is outside the scope of this work, it is a future area of research that could yield even more valuable results.

A key motivation of this research effort was the desire for the ability to simulate more complex microgrid and mobile power transfer system interactions, as well as to increase the fidelity and simulated time length. For this to be possible, a significant increase was necessary in the speed of the optimization routine. This work has provided that increase, and paved the way forward for complex, detailed modeling and simulation of these intricate power systems and control strategies for managing them.

We plan to implement the distributed pool strategy on a time-shares supercomputing environment where it will be possible to investigate a much larger populations size, hundreds of workers, and collections of multiprocessing simulations for each worker. Our reported results are currently based on a limited number of simulations run. We plan to pursue a much more comprehensive study investigating the *Pool* parameters and performing a statistically described benchmark of the results.

Acknowledgement

The work of Aaron Lee Garrett [9] (ECsPy and inspyred) and the inspyred community was used for implementation of the evolutionary algorithm. The ØMQ (ZeroMQ) messaging library was used for implementation of the distributed pool, along with pyzmq python bindings. Significant inspiration for the distributed pool design has come from Roy et al. [14].

Disclaimer

References

[1] T Back, Ulrich Hammel, and HP Schwefel. Evolutionary computation: Comments on the history and current state. *Evol. Comput. ...*, (April):3–17, 1997.

[2] George EP Box. Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, pages 81–101, 1957.

[3] Hans J Bremermann. Optimization through evolution and recombination. *Self-organizing systems*, pages 93–106, 1962.

[4] Carlos A Coello Coello. Recent trends in evolutionary multiobjective optimization. In *Evolutionary Multiobjective Optimization*, pages 7–32. Springer, 2005.

[5] Carlos A Coello Coello, David A Van Veldhuizen, and Gary B Lamont. *Evolutionary algorithms for solving multi-objective problems*, volume 242. Springer, 2002.

[6] Kalyanmoy Deb et al. *Multi-objective optimization using evolutionary algorithms*, volume 2012. John Wiley & Sons Chichester, 2001.

[7] Kalyanmoy Deb and Amrit Pratap. A fast and elitist multiobjective genetic algorithm: NSGA-II. *..., IEEE Trans.*, 6(2):182–197, 2002.

[8] Kalyanmoy Deb, Pawan Zope, and Abhishek Jain. Distributed computing of pareto-optimal solutions with evolutionary algorithms. *Evol. Multi-criterion Optim.*, pages 534–549, 2003.

[9] Aaron Garrett. *Neural Enhancement for Multiobjective Optimization.* PhD thesis, Auburn University, 2008.

[10] John H Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)*, 9(3):297–314, 1962.

[11] Zbigniew Michalewicz and David B Fogel. *How to solve it: modern heuristics.* Springer, 2004.

[12] Vijitashwa Pandey, Annette G Skowronska, Zissimos P Mourelatos, David Gorsich, and Matthew Castanier. Reliability and functionality of repairable systems using a minimal set of metrics: Design and maintenance of a smart charging microgrid. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information*

in Engineering Conference. American Society of Mechanical Engineers, 2013.

[13] Ingo Rechenberg. Cybernetic solution path of an experimental problem. 1965.

[14] Gautam Roy, Hyunyoung Lee, Jennifer L Welch, Yuan Zhao, Vijitashwa Pandey, and Deborah Thurston. A distributed pool architecture for genetic algorithms. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1177–1184. IEEE, 2009.

[15] Gautam Samarendra N. Roy. A Distributed Pool Architecture for Genetic Algorithms. Master's thesis, Texas A&M, 2009.

[16] J David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st international Conference on Genetic Algorithms*, pages 93–100. L. Erlbaum Associates Inc., 1985.

[17] Annette Skowronska, David Gorsich, Jeremy Mange, Andrew G. Dunn, Vijitashwa Pandey, and Zissimos Mourelatos. Global Strategies for Optimizing the Reliability and Performance of U.S. Army Mobile Power Transfer Systems. In *NDIA Gr. Veh. Syst. Eng. Technol. Symp.*, 2013.

[18] Michael D Vose. *The simple genetic algorithm: foundations and theory*, volume 12. MIT press, 1999.

[19] Darrell Whitley. A genetic algorithm tutorial. *Stat. Comput.*, 1994.