SANDIA REPORT
SAND2015-8067
Unlimited Release
Printed September, 2015

Classifier-Guided Sampling for Complex Energy System Optimization

Peter B. Backlund, John P. Eddy

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online order:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

Classifier-Guided Sampling for Complex Energy System Optimization

Peter B. Backlund and John P. Eddy
System Readiness and Sustainment Technologies
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS1188

Abstract

This report documents the results of a Laboratory Directed Research and Development (LDRD) effort entitled "Classifier-Guided Sampling for Complex Energy System Optimization" that was conducted during FY 2014 and FY 2015. The goal of this project was to develop, implement, and test major improvements to the classifier-guided sampling (CGS) algorithm. CGS is type of evolutionary algorithm for performing search and optimization over a set of discrete design variables in the face of one or more objective functions. Existing evolutionary algorithms, such as genetic algorithms, may require a large number of objective function evaluations to identify optimal or near-optimal solutions. Reducing the number of evaluations can result in significant time savings, especially if the objective function is computationally expensive. CGS reduces the evaluation count by using a Bayesian network classifier to filter out non-promising candidate designs, prior to evaluation, based on their posterior probabilities. In this project, both the single-objective and multi-objective versions of the CGS are developed and tested on a set of benchmark problems. As a domain-specific case study, CGS is used to design a microgrid for use in islanded mode during an extended bulk power grid outage.

ACKNOWLEDGMENTS

The Classifier-Guided Sampling (CGS) LDRD team would like to thank Bruce Thompson for his time and support as Program Manager of this effort. The team also would like to thank Stephen Henry, Frank Muldoon, and Steve Verzi for their time and input during the CGS peer-review and idea brainstorming sessions. The time and feedback from Laura Swiler are also greatly appreciated. The team thanks Jason Stamp, Michael Baca, Ross Guttromson, Jordan Henry, Richard Jensen, Karina Muñoz-Ramos, Ben Schenkman, and Mark A. Smith for their contributions to the City of Hoboken Energy Surety Analysis effort. Lastly, Carolyn Seepersad of the University of Texas at Austin and David Shahan of HRL Laboratories are acknowledged for their contributions to the early development of CGS.

CONTENTS

FIGURES

TABLES

1 INTRODUCTION

1.1 Problem Background

The difficult task of designing a complex engineering system is significantly aided with the use of computer models and optimization software. For example, in a modern energy distribution system, integration of traditional fossil and renewable sources, unpredictable loads, and the inclusion of energy storage elements are only a few of the design decisions that could be informed with the use of modeling and optimization applications. High-fidelity computer models of such systems are invaluable resources for understanding and evaluating system behavior. However, computational expense limits their usefulness for optimization and decision space exploration because metaheuristic optimization methods may require numerous evaluations of the expensive model to identify promising solutions.

Reducing the time required to solve complex optimization problems is a valuable endeavor. If a single optimization run requires multiple days to complete, a significant reduction in the solution time will enhance our ability to support stakeholders. This is especially true in light of the fact that several analyses are often required; model or data errors may be discovered on initial trials, or additional runs may be requested upon stakeholder review of initial results. Reducing the amount of time required to perform an optimization run will enable more runs to be performed, and may mean the difference between addressing a specific question or not, given a hard deadline such as a final out brief to leadership or an annual budgeting cycle.

1.2 Goals and Objectives of the Project

The goal of this project is to improve a relatively new optimization technique known as Classifier-Guided Sampling (CGS) and to develop a novel optimization method that is capable of solving large, computationally expensive, discrete variable, single- and multi-objective optimization problems. CGS reduces objective function evaluations by using an inexpensive Bayesian network (BN) classifier to leverage knowledge gained from all prior evaluations and to predict the qualitative performance of candidate solutions. Using a classifier in this way helps the search process focus on high-performance designs and to avoid wasteful evaluations of poor-performing ones.

Prior to the start of this project, CGS was a new technology with significant shortcomings that rendered it incapable of handling large-scale, multi-objective optimization problems. First, in its initial form [1], CGS was only suitable for small design spaces because it required enumeration and processing of every solution in the discrete design space at each iteration. This approach is impractical for moderate to large design spaces due to the combinatorial explosion that is associated with an increase in the number of design variables. To address this issue, the probability distributions on which the BN classifier is based are sampled directly to guide the search towards the most promising designs in the solution space. Initial steps to address this issue were taken in [2], and this LDRD continues this work by implementing a more scalable and flexible computer implementation of this approach. Second, accounting for interactions between variables is critical to the performance of the CGS algorithm. Prior to this work, these dependencies had to be programmed into the algorithm manually. While doing so is feasible for small problems that are well-understood, identifying and assigning these dependencies when dealing with large problems with hundreds of variables is a nearly impossible task. This issue has been addressed by incorporating an algorithm that automatically builds the BN based on a set of training data. Lastly, multi-objective capability has not previously been developed for the CGS method. This capability is realized by utilizing the classifier outputs to sample only the candidate solutions that are believed to be Pareto optimal. The goal of this research has been to address the three aforementioned shortcomings of CGS. In its new form, CGS has the ability to solve large-scale, computationally expensive, multi-objective optimization problems.

The improved version of CGS is tested on a set of single- and multi-objective test problems. The rate at which the algorithm converges towards optimal designs is recorded, and the performance is compared to a standard genetic algorithm. As a domain-specific case study, CGS is used to design a microgrid for use in islanded mode during an extended bulk power grid outage. The microgrid design problem has a single- and multi-objective problem formulation. In the single-objective case, the average load not served (LNS) during the outage is minimized, subject to a constraint on installation cost. In the multi-objective case, a set of Pareto optimal designs is sought that efficiently trade minimization of installation cost and LNS.

2 CLASSIFIER-GUIDED SAMPLING

CGS is a randomized optimization technique suitable for discrete variable problems. CGS achieves efficient global optimization by using a Bayesian classifier to provide categorical predictions of the performance of candidate solutions prior to expensive evaluation. In the following section, an overview of related optimization technologies is provided and the differentiating characteristics of CGS are highlighted. In Section 2.2, the mathematical details of a Bayesian network classifier are given. Finally, in Sections 2.3 and 2.4, the CGS algorithm and its multi-objective extension, respectively, are explained.

2.1 Related Work

Over the past several decades, significant research effort has been dedicated to the study and development of evolutionary algorithms (EAs). The term EA describes a broad category of population-based optimization techniques that are loosely inspired by natural evolutionary processes in which a population of individuals evolves according to a randomized natural selection process. EAs make no assumptions about the objective functions they seek to optimize. Therefore, they are well-suited for black-box optimization problems where only the inputs and outputs of the objective functions are known, gradient information is unavailable, and linearity or convexity cannot be assumed.

The most well-known type of EA is the genetic algorithm (GA). GAs explore the design space by selecting and combining solutions from the current population [3]. Ideally, combinations of high-performing schemata, or partial solutions, are created and combined to form ever-improving designs with each iteration. These high-quality partial solutions are known as building blocks [4]. When building blocks are of a high order, or if they span a significant length of the encoded solution, the GA crossover and mutation operators are likely to inadvertently disrupt them, a common phenomenon known as the linkage problem [5]. One approach to addressing this issue is to develop a probabilistic model using the population of best known designs, and subsequently use this model to generate new designs. The general approach of replacing traditional GA recombination operators with a probabilistic model is the defining characteristic of a class of EAs known as estimation of distribution algorithms (EDAs) [6].

Several EDAs have been published since the mid-1990s. For example, the population-based incremental learning algorithm (PBIL) [7] and compact genetic algorithm (cGA) [8] use a probability vector to generate new solutions. However, both assume that there are no interactions between variables. The mutual information maximizing input clustering (MIMIC) [9] algorithm uses a 2nd order density estimation, and is therefore capable of capturing pairwise variable interactions. Other EDAs model the factorization of distributions with a Bayesian network, the structure of which is learned and updated incrementally throughout the optimization process. Perhaps the best known algorithm of this type is the Bayesian optimization algorithm (BOA) [10]. EDAs like BOA are able to capture variable interaction of arbitrary order without the need for expert knowledge of the underlying optimization problem; a practitioner using BOA need only specify an upper limit on the order of the interactions amongst design variables.

CGS builds on the work of previous EDAs by using a Bayesian classifier to perform two important tasks in the optimization. First, similar to other EDAs, the probability distributions that comprise the classifier are sampled directly to generate new designs for exploration. Like the BOA, the distributions are factored according to a Bayesian network, the structure of which is learned based on a population of promising designs. Second, CGS uses a classifier to predict the performance of each candidate design, prior to objective function evaluation. The second task described above is what makes CGS different from all other existing EDAs. CGS is therefore able to take advantage of the properties of the most advanced EDAs, while also leveraging information gained from all previous evaluations in a machine learning fashion to filter out designs that have a low likelihood of being a high-performance design after evaluation of the objective function.

2.2 Bayesian Classifiers

Classification is a machine learning problem in which the goal is to predict the categorical class label of a specific instance of a set of attributes. Prior to classification, a classifier must be trained using a set of known feature vector / class label pairs. In the context of CGS, the feature vectors are specific design instances, and the class labels are assigned according to their known objective function values (e.g. ‘good’/’bad’).

A Bayesian classifier uses a factorization of probability distributions to predict the categorical performance of a candidate configuration based on all previously evaluated points. Let K be the number of classes, and let c_k represent the class k for $k \in \{1, 2, ..., K\}$. The classification is performed in a D-dimensional design space, and $\mathbf{x} = [x_1, x_2, \ldots, x_D]$ is a vector of design variables. If $\hat{\mathbf{x}}$ is a specific design instance of $\mathbf{x}$, Bayes’ formula is used to estimate the probability that $\hat{\mathbf{x}}$ is a member of the class c_k, (i.e., the probability of c_k given $\hat{\mathbf{x}}$**)** according to:

$$P(c_k \mid \hat{\mathbf{x}}) = \frac{P(\hat{\mathbf{x}} \mid c_k) P(c_k)}{P(\hat{\mathbf{x}})} = \frac{P(\hat{\mathbf{x}} \mid c_k) P(c_k)}{\sum_{k=1}^{K} P(\hat{\mathbf{x}} \mid c_k) P(c_k)} \tag{1}$$

where $P(\mathbf{x}|c_k)$ is the conditional probability of a design instance given the class label, and $P(c_k)$ is the prior probability of any randomly selected point belonging to class c_k. The LHS of Eq. (1) is called the *posterior* probability, and design $\hat{\mathbf{x}}$ is classified as a member of the class c_k that has the highest posterior when compared to all other classes.

CGS sets the prior probabilities, $P(c_k)$, according to a constant discrete uniform distribution:

$$P(c_k) = \frac{1}{K}, \quad \forall k \in \{1, 2, ..., K\} \tag{2}$$

where K is the number of performance categories.

$P(\mathbf{x}|c)$ is a D-dimensional joint distribution that is estimated from a training set of design vector / class label pairs. It is generally impractical to model the full joint distribution. Whenever possible, it is advantageous to make conditional independence assumptions about the design variables and refactor $P(\mathbf{x}|c)$ into a product of univariate distributions. While no variable can be independent of the class variable *c*, variables may or may not be independent of each other. In its simplest form, CGS assumes that all design variables are independent, and $P(\mathbf{x}|c)$ reduces to:

$$P(\mathbf{x}|c) = P(x_1|c)P(x_2|c)\ldots P(x_D|c) \tag{3}$$

Using the factorization in Eq. (3) is a special case of Bayesian classifier known as the naïve Bayes classifier [11]. A particular factorization of $P(\mathbf{x}|c)$ can be graphically represented using a BN, which is a directed acyclic graph. For example, the BN representation of the factorization in Eq. (3) is shown in Figure 1.

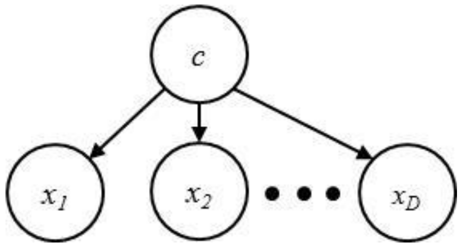


Figure 1: Bayesian network for the naïve Bayes classifier

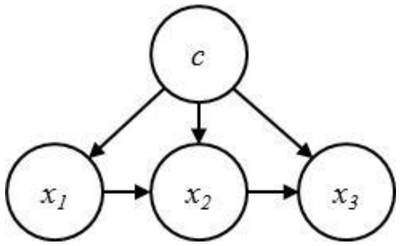


Figure 2: Bayesian network for a classifier with non-independent design variables

Dependencies between variables can be represented by including directed edges between variable nodes in the BN. For example, the BN depicted in Figure 2 is a graphical representation of the joint probability factorization in Eq. (4).

$$P(\mathbf{x}|c) = P(x_1|c)P(x_2|c,x_1)P(x_3|c,x_2) \tag{4}$$

CGS is currently only for use with discrete variables, and therefore uses *D* discrete probability distributions to estimate $P(\mathbf{x}|c)$. The task of estimating the distribution parameters is achieved by assigning class labels to all design points that have been evaluated with the expensive simulation, and then counting the number of instances of each possible combination of that variable and its parent variables in the BN (including the class variable).

For example, assume design variables x_1 and x_2 both have domain {0, 1} and x_2 has parent nodes x_1 and *c* in the BN. We need to estimate $P(x_2 \mid c = \text{'good'}, x_1 = 1)$, where 'good' is a categorical class label to indicate that a design is promising. This distribution is modeled with a probability mass function with two parameters: $\theta_{x2=0,c=\text{good},x1=1}$ and $\theta_{x2=1,c=good,\text{ x1}=1}$. The distribution parameter $\theta_{x2=0,c=\text{good},x1=1}$ is estimated according to:

$$\theta_{x_2=0,c=good,x_1=1} = \frac{\beta + \#(x_2 = 0, c = good, x_1 = 1)}{\alpha + \#(x = 1, c = good, x_1 = 1) + \beta + \#(x = 0, c = good, x_1 = 1)} \tag{5}$$

where #(x_2 = 0, c = 'good', x_1 = 1) and #(x = 1, c = 'good', x_1 = 1) are the number of instances of (x_2 = 0, c = 'good', x_1 = 1) and (x_2 = 1, c = 'good', x_1 = 1) in the training set, respectively. The parameters α and β represent initial counts and can be used to initialize the class conditional probability distributions when prior knowledge of the distribution probabilities is available. The distribution parameter $\theta_{x2=1,c=\text{good},x1=1}$ is estimated similarly.

The process is generalized to variables with larger domain sizes as follows. If the discrete design variable x_i has cardinality (domain size) C, then x_i^j is the j^{th} level in the domain of x_i, where $j \in \{1,2,\ldots,C\}$. Furthermore, if the initial counts are represented by the vector $\boldsymbol{\alpha}$ = [α_1,α_2,…,α_C], then the discrete distribution parameters for x_i^j given a specific instantiation of its parent variables $\hat{\mathbf{p}}$ is given by Eq. (6):

$$\theta_{x_i=x_i^j,\hat{\mathbf{p}}} = \frac{\alpha_j + \#\left(x_i = x_i^j, \hat{\mathbf{p}}\right)}{\sum_{j=1}^{C}\left[\alpha_j + \#\left(x_i = x_i^j, \hat{\mathbf{p}}\right)\right]} \tag{6}$$

The simplest setting for the initial counts is to set them all to unity [12]. Doing so sets all of the class conditional probability distributions to uniform before any training points are added to the classifier. This approach generally makes sense unless there is reason to believe, possibly through prior experimentation, that some variable values have higher likelihood than others given the class label.

2.3 Using a Bayesian Classifier to Perform Black-Box Optimization

In this section, the single-objective version of CGS is described. The extension to multi-objective optimization is detailed in Section 2.4. CGS uses a Bayesian classifier to achieve efficient design space exploration and optimization in two ways. First, new candidate points are generated by sampling the class conditional probability distributions that comprise the classifier. By sampling the distributions that are trained with high-performance solutions, new points are generated that are likely to be similar to those that are already known to be good and that may improve the objective function. Second, the updated classifier is then used to screen each new candidate solution based on the posterior probability of the design's class prior to expensive evaluation. With each new point that is evaluated, assigned a class label, and added to the classifier training set, the classifier improves its ability to generate high-performance solutions and filter out low-performance ones.

2.3.1 Main Loop of the Algorithm

Figure 3 depicts the CGS main loop. Initially, (Step 0) the class-conditional probability distributions are instantiated as uniform discrete distributions. In Step 1, the class conditional probabilities of the high-performance class are sampled to generate a candidate solution. In Step 2, the candidate's posterior probability of being 'good' is computed by the classifier. In Step 3, the candidate solution is accepted or rejected for evaluation based on two criteria. First, the solution is checked against all previously evaluated solutions to avoid repeat evaluations of the same solution. Second, if the candidate solution's posterior probability of being 'good' is below a threshold, it is rejected. The threshold value is determined for each candidate solution by sampling from a uniform distribution ranging from zero to one. If the candidate solution is accepted, the new design point is added to the set of new designs to be evaluated with the expensive simulation in Step 4. Otherwise, the method returns to Step 1 to generate a new point. Steps 1-3 are repeated until N_{batch} new designs are generated and accepted for evaluation, where N_{batch} is a user-defined parameter that can be any positive integer and represents the number of new designs to evaluate at each iteration.

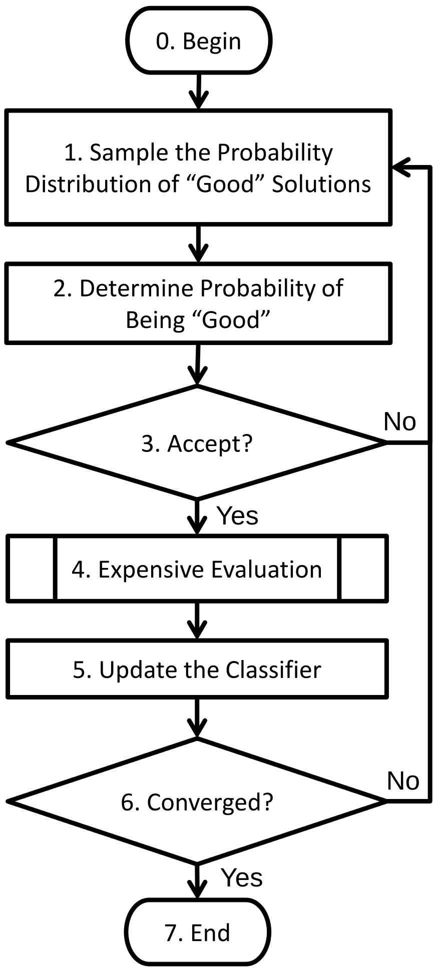

0. Begin
1. Sample the Probability Distribution of "Good" Solutions
2. Determine Probability of Being "Good"
3. Accept?
No
Yes
4. Expensive Evaluation
5. Update the Classifier
6. Converged?
No
Yes
7. End

Figure 3: Classifier-guided sampling algorithm

In Step 5, the classifier is updated according to two separate processes. First, the newly evaluated designs must be assigned class labels and added to the classifier training dataset. Note that some of the previous training points may require class label reassignment (i.e., a 'good' point is now 'bad' due to an overall improvement in the set of the best designs). Second, the structure of the classifier's Bayesian network may be reconstructed from the current set of classifier training points using a learning algorithm. These two processes are described in detail in subsequent sections.

The main loop repeats if a convergence or stopping criterion does not end the cycle (Step 6). Stopping criteria can include reaching a user-defined upper limit on the evaluation count, iteration count, or wall-clock time. Any common convergence criteria may also be used, such as achieving a desired objective function value, or failing to improve the best design by some percentage after a constant quantity of most recent evaluations.

2.3.2 Assigning Class Labels

Class labels are assigned to newly evaluated designs according to how desirable they are relative to all those that have been evaluated thus far. At a minimum, two classes are required, but more may be used if desired. In the work presented here, each evaluated design is given a class label of either ‘good’ or ‘bad’. Without loss of generality, we assume in this discussion that we seek to minimize the objective function.

Class labels are determined by assigning the top N_{best} evaluated solutions a label of ‘good’ and all others a label of ‘bad’, where N_{best} is a user-defined constant. In other words, the newly evaluated designs are added to the previous set of N_{best} best designs, the set is sorted by ascending objective function value, and the top N_{best} designs are assigned the ‘good’ label and all other are assigned the ‘bad’ label. All new designs are then added to the training set with their assigned class labels. Designs that were in the previous iteration’s set of N_{best} best designs will either remain as they were (if they are still member of the ‘good’ class) or their class label will be reassigned from ‘good’ to ‘bad’ (they are no longer in the set of N_{best} best designs).

For constrained optimization problems, a method is needed to rank and sort a set of designs that has a mix of feasible and infeasible solutions. One approach to constraint handling is to use a penalty function that increases the effective value of the objective function. However, this approach distorts the objective function and requires additional user-defined parameters. Therefore, in this work, we implement the method proposed in [13]. Using this technique, feasible solutions are always preferred to infeasible ones. When comparing two infeasible solutions, the one that violates the constraints by a lesser overall amount is preferred. That is, for each constraint, the amount of the violation is scaled by dividing it by the average violation for that constraint of all designs evaluated thus far and then multiplied by 100. The overall violation is then computed by summing all scaled violations. This procedure prevents CGS from unfairly focusing on constraints that tend to have higher raw violation values. As before, when comparing two feasible solutions, the one with the lower objective function value is preferred.

2.3.3 Learning the Bayesian Network Structure

The ability for CGS to explicitly capture interactions between variables offers a significant advantage over GAs. In practice, expert knowledge of both BN theory and of the problem being solved is required for manual construction of the BN. Therefore, one of the major goals of this LDRD was to implement a BN learning algorithm that updates the structure of the BN with each passing iteration of the CGS main loop.

In general, learning the structure of a BN given a data set is NP-hard [14]. Therefore, it is not possible to construct the BN with an algorithm that is both fast and correct. In this context, a fast learning algorithm is critical, because CGS requires the BN to be learned repeatedly during a single optimization run. The K2 algorithm [15], a greedy probabilistic network building heuristic, is selected due to its speed and its strong theoretical basis. K2 seeks to maximize an overall network score, which is given by Eq. (7):

$$P\left(B_S\right)=\sum_{i=1}^{D} g\left(i,\pi_i\right) \tag{7}$$

where D is the number of design variables, B_S is the candidate BN structure, and $g(i,\pi_i)$ is the node score (given in Eq. (8)).

$$g\left(i,\pi_i\right)=\sum_{j=1}^{q_i}\left(\log\left(\frac{\left(r_i-1\right)!}{\left(N_{ij}+r_i-1\right)!}\right)+\sum_{k=1}^{r_i}\log\left(N_{ijk}!\right)\right) \tag{8}$$

A thorough discussion of the theory behind Eq. (8) is provided by [15]. Table 1 provides descriptions of the terms in the node score calculation

Table 1. Description of symbols in the K2 node score equation

π_i	The parent variables of the variable x_i, where $(1 \le i \le D)$
q_i	The number of possible configurations of π_i
r_i	The cardinality (domain size) of variable x_i
N_{ij}	The number of instances in the dataset where the variables in π_i take their j^{th} configuration, where $(1 \le j \le q_i)$
N_{ijk}	The number of instances in the dataset where the variable x_i takes its k^{th} value and the variables in π_i take their j^{th} configuration, where $(1 \le k \le r_i)$

The K2 algorithm requires two parameters as user inputs: an upper limit on the number of parents each node may have, and a sequence order in which to process the nodes. K2 visits each node in the network according to the provided search order to drive the network structure towards an approximately maximal scoring configuration. When a node is visited, it is assumed that it initially has no parents; parents are added if doing so can increase the node score. Candidate parents for that node are restricted to its predecessors in the search order. The predecessor node that increases the node score by the greatest amount is added as a parent until the upper limit on parents is reached or until no additional candidate parents exist that would increase the node score. Note that the most complex networks are not necessarily preferred, as edges are only added if doing so increases the overall score of the network.

In the CGS main loop, automatic learning of the classifier BN can be optionally performed in Step 5 (Figure 3). Whether or not network learning occurs depends on the user-defined parameter P_L, which is the upper limit on the number of parents a node may have in the BN (not including the class variable node, which is always a parent to all variables). If P_L is set to zero, no learning will occur. If it is set to any positive value between 1 and D-1, network learning is performed. When automatic learning is enabled, CGS uses the set of N_{best} designs as the dataset to learn the BN. Only the set of N_{best} are used because, of all previously evaluated designs, the best ones are most likely to respect any variable interactions that may exist. The K2 search order changes on each iteration, and is given as a random permutation of the design variables.

After K2 learns a new network on the current iteration, the network score for the previous iteration's network is computed using the latest dataset. The network structure is updated only if the new structure has a higher network score than the previous network. Finally, if the network's structure changes, the classifier is retrained with all current design vector / class label pairs.

2.4 Multi-Objective Classifier-Guided Sampling

CGS has thus far been described in the context of single-objective optimization. That is, the goal of the optimization has been to minimize (or maximize) a single performance metric subject to zero or more constraints. In practice, however, it is often necessary to optimize more than one competing objective simultaneously for the purpose of identifying and understanding the impact of decisions on performance tradeoffs. The goal of multi-objective optimization is to identify the set of *non-dominated* designs, i.e., those for which there are no known designs outside of this set that are better in at least one objective and not worse in any others.

Multi-objective CGS (MOCGS) proceeds almost identically to the single-objective version with the main difference being the manner in which class labels are assigned after each new batch of solution evaluations. Recall from Section 2.3.2 that class labels are assigned by sorting a set of designs with known objective function values and assigning the class label of 'good' to the top N_{best} designs. The size of this set is equal to the sum of N_{best} and N_{batch} and is sorted according to the value of the single objective function (and feasibility if constraints are present). When multiple-objectives are present, an alternate sorting scheme is needed. To perform this task, MOCGS uses the same sorting procedure that is implemented by NSGA-II (non-dominated sorting genetic algorithm) [16], which is a multi-objective genetic algorithm that is popular for its speed and simplicity.

NSGA-II sorts solutions first by increasing domination rank and next by decreasing crowding distance. To determine the domination rank (Figure 4), the set of solutions that are non-dominated are assigned a ranking of 1. The solutions that would be non-dominated after removing all of the solutions of rank 1 are assigned a ranking of 2. This process continues until there are no more rankings to assign. The crowding distance is a measure of the density of other solutions of a given rank that surround a particular solution. Solutions that are crowded by fewer other solutions are preferred in order to promote diversity in the population. The crowding distance of a particular solution is determined by adding the distances of the two designs on either side of it along each of the objectives. The extreme solutions of a particular domination rank are always given a crowding distance of infinity to maintain the broadest spread of the non-dominated front. For a detailed explanation and pseudocode of the NSGA-II algorithm for determining domination ranks and crowding distances, refer to [16].

MOCGS handles constraints using the same approach as the single-objective version. Feasible solutions are always preferred to infeasible ones. When comparing two infeasible solutions, the one that violates the constraint by a lesser overall amount is preferred (regardless of domination rank and crowding distance).

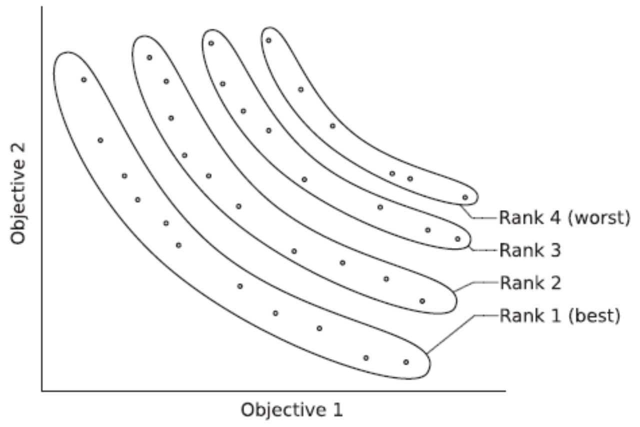


Figure 4: Domination raking for a two-objective minimization problem [17]

Figure 5 illustrates how CGS uses the NSGA-II sorting procedure to assign class labels after each new batch of new designs is evaluated. First, the new N_{batch} new designs are combined with the current set of N_{best} 'good' designs to form one unsorted set. Next, the set is sorted by increasing domination ranking. If a domination rank exists whose designs can only be partially accommodated into the top N_{best} designs (rank 2 in Figure 5), members of that rank are sorted by decreasing crowding distance. Finally, the top N_{best} designs in the set are assigned to the 'good' class, and all others are assigned to the 'bad' class, and the classifier is updated appropriately.

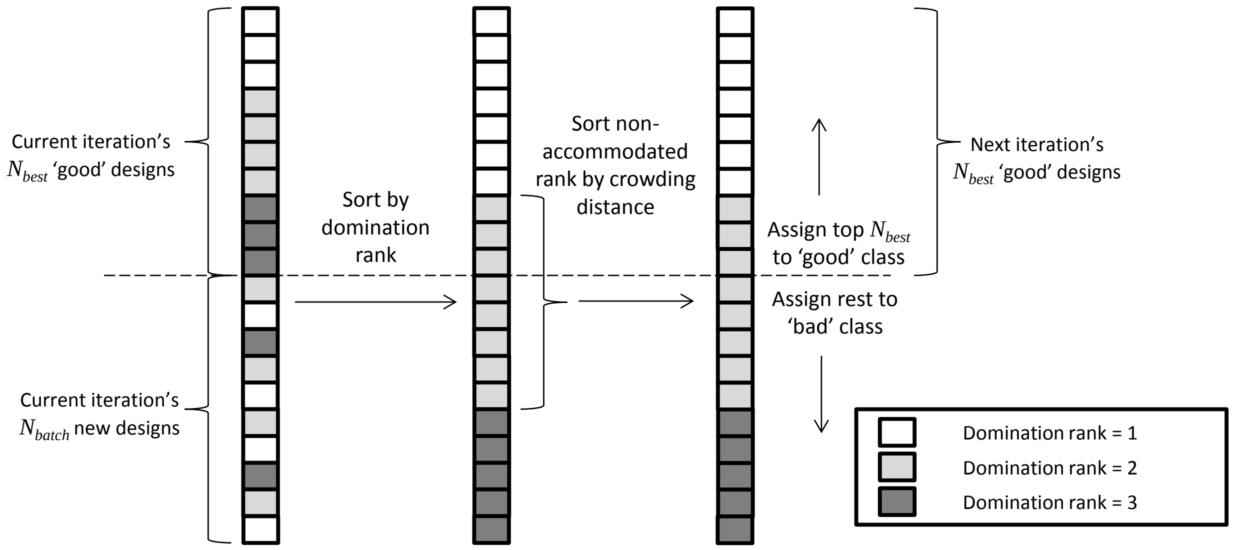

Current iteration's N_{best} 'good' designs
Current iteration's N_{batch} new designs
Sort by domination rank
Sort non-accommodated rank by crowding distance
Assign top N_{best} to 'good' class
Assign rest to 'bad' class
Next iteration's N_{best} 'good' designs
Domination rank = 1
Domination rank = 2
Domination rank = 3

Figure 5: NSGA-II non-dominated crowding sorting procedure applied to MOCGS

The final output of a MOCGS run should be the set of all non-dominated solutions that the optimization was able to discover. However, it is likely that the quantity of these solutions will exceed the constant N_{best}, and returning the set of N_{best} 'good' designs would not provide the complete desired output. Therefore, MOCGS maintains an external set of solutions that contains the set of all non-dominated designs that have been identified. It is updated after each iteration, and it does not participate in or influence the optimization in any way. When MOCGS terminates it will contain all non-dominated designs that the optimizer was able to identify. If no feasible solutions were discovered, it will contain all non-dominated infeasible designs without regard to the extent of constraint violation.

3 COMPARISON TO GENETIC ALGORITHMS

In this section, the performances of single- and multi-objective CGS are compared to single- and multi-objective GAs. The GA implementation used in these tests is the JEGA solver available in the DAKOTA toolkit [18]. For the single-objective comparison, six constrained and unconstrained single-objective problems are used. For the multi-objective experiments, one of the constrained single-objective problems (50-item knapsack problem) is reformulated as a multi-objective problem where the weight capacity constraint is treated as an objective rather than a constraint. The test problems are described next.

3.1 Test Problems

3.1.1 30-Variable Deceptive Function of Order 3

The deceptive function of order 3 [5] is a maximization problem with strong dependencies among the decision variables, and is formulated as

$$f_{deceptive}^{3}(X) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise} \end{cases} \tag{9}$$

where X is a vector of three binary variables, and u is the sum of the input variables. Eq. (9) can be summed over an arbitrary number of variables to scale the problem to larger sizes. In the experiments that follow, a 30-variable objective function is generated by summing Eq. (9) a total of ten times, as shown in Eq. (10):

$$f_{30deceptive}(X) = \sum_{i=0}^{n/3-1} f_{deceptive}^{3}(S_i) \tag{10}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of variables, $S_i = (X_{3i}, X_{3i+1}, X_{3i+2})$, and $n = 30$. Due to the strong coupling of the input variables, this function is ideal for demonstrating the effectiveness of using a BN-based optimization technique such as CGS when compared to GAs.

3.1.2 50-Item 0-1 Knapsack Problem

The objective of the 0-1 knapsack problem is to select a subset of available items that each have a constant weight and value such that the total value, V, of the selected items is maximized without exceeding a predetermined upper limit on the combined weight, W. For the experiments performed here, there are 50 different items to choose from, and only one or zero of each item may be included. Denoting a vector of binary variables $\mathbf{x} = (x_1, x_1, \ldots, x_{50})$ to represent a solution to the problem, the problem is formulated as follows [1]:

$$\begin{aligned}&\text{Maximize } V(\mathbf{x}) = \sum_{i=1}^{n} v_i x_i \\ &\text{Subject to } \sum_{i=1}^{n} w_i x_i \le W, \quad x_i \in \{0,1\}\end{aligned} \tag{11}$$

where v_i and w_i are the value and weight of item *i*, respectively. *W* is chosen to be 50% of the total weight of all available items. The weights and values used for this problem are provided in Appendix A.

For the multi-objective experiments, this problem is reformulated to a 2-objective problem by treating the total weight as an objective that is sought to be minimized.

3.1.3 System Design Problem

The goal of this problem is to identify the best selection of subsystems that combine in a modular fashion to form a complete system. Each subsystem has two or more technology options to choose from, and each technology option has an associated cost and utility associated with it. Only one technology option from each subsystem may be chosen. In addition, technology options from one subsystem may necessitate or obviate technology options from another subsystem. In the single-objective version of this problem, the goal is to maximize the total utility *U* of all subsystems without exceeding a fixed upper limit on total system cost *C* while respecting all necessitation and obviation constraints.

For the experiments performed here, the overall system is to be comprised of 15 subsystems, each with varying numbers of available technology options to choose from. In addition, there are 25 necessitation constraints and 15 obviations constraints that cannot be violated. The utilities and costs of each technology option as well as the necessitation and obviation constraints are detailed in Appendix A.

3.1.4 30-City Traveling Salesman Problem

The traveling salesmen problem (TSP) is a frequently studies combinatorial optimization problem. Given a set of cities and their Cartesian coordinates, the goal is to find the shortest possible tour that visits each city exactly once and returns to the city of origin. The distance between two cities is assumed to be the same regardless of direction traveled.

A candidate solution to this problem is encoded as a binary string of length $n \cdot m$, where n is the number of cities in the tour and $m = \log_2 n$ (rounded up) [7]. Each city is represented by a contiguous substring of length m witch are decoded into integers that represent positions in the tour. Ties are resolved by assuming that the city whose substring appears first in the bit string is the city that is visited first. This encoding/decoding strategy ensures that all possible binary strings of length $n \cdot \log_2 n$ bits can be decoded into valid TSP solutions (i.e., each city is visited exactly once). For the experiments performed here, the 30 city locations are obtained from [19] and are tabulated in Appendix A.

3.1.5 Warehouse Location Problem

In the warehouse location problem (WLP), a set of stores exist which each must be supplied by exactly one warehouse. Warehouses, each of which may supply one or more stores, may be opened in a variety of locations. There is a fixed cost associated with opening a warehouse, and the cost to supply a store depends on which warehouse is supplying it. Furthermore, each warehouse has a maximum capacity that specifies how many stores it can supply. The goal of the WLP is to decide which warehouses to open and which stores they should supply such that the total warehouse opening and store supply costs are minimized.

In the experiments performed here, a candidate solution is represented by a vector $\mathbf{x} = [x_1, x_2, \dots, x_n]$, where n is the number of stores. Each x_i represents a store and its value can be any integer between 1 and m, where m is the number of warehouses. The problem specific parameters are taken from the example in the CPLEX OPL user manual [20]. There are ten stores and five warehouses. The fixed cost of opening each warehouse is 30. The warehouse capacities and store supply costs are tabulated in Appendix A.

3.1.6 Welded Beam Design Problem

The welded beam problem [21, 22] is a popular engineering design optimization problem in which the goal is to minimize the cost of welding a metal cantilever beam. A rectangular bar is to be welded at one end to a wall and it will support a point load at the opposite end (Figure 6). The objective is to select the weld style, beam/weld material, and geometric properties that minimize the total material cost without violating any of four constraints.

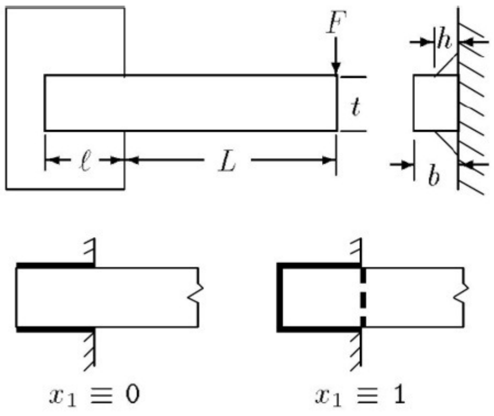


Figure 6: Welded beam problem [22]

The weld style describes whether two ($x_1 = 0$) or four ($x_1 = 1$) of the contact edges between the beam and wall are to be welded. The weld filler metal and beam material will be of the same type, and the options are steel, cast iron, brass, and aluminum (x_2 = 1, 2, 3, or 4, respectively). The geometric parameters are the thickness of the weld ($x_3 = h$), the width of the beam ($x_4 = t$), the thickness of the beam ($x_5 = b$), and the length of the welded portion of the beam ($x_6 = l$). The geometric parameters are restricted to discrete values, as shown in Table 2.

Table 2. Welded beam design problem geometric parameters

Variable	*Minimum (inches)*	*Maximum (inches)*	*Step size (inches)*
x_3	0.0625	0.5000	0.0625
x_4	7.500	10.000	0.125
x_5	0.0625	1.0000	0.0625
x_6	0.125	3.000	0.125

Representing the six design variables by a vector **x**, the objective function and constraints are computed by Eq. (12):

$$\begin{aligned} &\text{Minimize} \quad f(\mathbf{x}) = (1+c_1)\,x_3^2\,(x_6 + x_1 x_4) + c_2 x_4 x_5 (L + x_6) \\ &\text{Subject to} \quad \begin{cases} g_1(\mathbf{x}) = \quad S - \sigma(\mathbf{x}) \ge 0 \\ g_2(\mathbf{x}) = \quad P_c(\mathbf{x}) - F \ge 0 \\ g_3(\mathbf{x}) = \quad \delta_{\max} - \delta(\mathbf{x}) \ge 0 \\ g_4(\mathbf{x}) = \quad 0.577S - \tau(\mathbf{x}) \ge 0 \end{cases} \end{aligned} \tag{12}$$

Where c_1 and c_2 are material costs, and g_1, g_2, g_3, and g_4 are constraints on the bending stress $\sigma(\mathbf{x})$, buckling load $P_c(\mathbf{x})$, beam deflection $\delta(\mathbf{x})$, and weld shear stress $\tau(\mathbf{x})$, respectively. The design stress, S, is dependent on the material choice. The force, F, of the point load at the end of the beam is 6000 lb, the extended length (portion of beam not welded to the wall) L is 14 inches, and the maximum allowable deflection, $\delta_{\max}$, is 0.25in. The material costs, properties, and formulas for calculating $P_c(\mathbf{x})$, $\delta(\mathbf{x})$, and $\tau(\mathbf{x})$ are available in [23].

3.2 Single-Objective Performance Comparison Results

The single-objective optimization performance between CGS and GAs is assessed using rate of convergence tests in which, during an optimization run, the current single best solution is recorded after each evaluation of the objective function. Plotting the best known objective function vs. the number of function evaluations then provides a visual and analytical indicator of how quickly each method converges towards optimal solutions. Both CGS and GA perform differently across multiple runs due to their reliance on random number generation. Therefore, each optimization problem is solved 50 times using both optimization techniques, and the average results are plotted. In addition, vertical bars are added at select points to indicate the standard deviation of the best known objective function found across the 50 tests. Repeat evaluations of previously-evaluated designs are precluded for both methods.

In all tests, the GA used a two-point crossover operator. The mutation operator used randomly selects a design variable, converts it to a binary representation, randomly selects a bit from that string, and toggles it. The selector used always favors feasible solutions to infeasible ones, and less infeasible solutions to more infeasible ones. Documentation of these operators is provided in the DAKOTA reference manual [18].

There are several user-defined parameters that must be specified for both optimization techniques. For CGS, they include N_{best}, N_{batch}, and P_L (or a constant user-specified BN if automatic network learning is not enabled). For GA, the user-defined parameters include the population size, crossover rate, and mutation rate. To get the best possible performance of both CGS and GA, their respective parameters were tuned in a set of preliminary optimization runs.

In Sections 3.2.1-3.2.6, the user-defined parameters that are used by both CGS and GA are tabulated and the rate of convergence curves are plotted. The results of all test problems are discussed together in Section 3.2.7.

3.2.1 30-Variable Deceptive Function of Order 3

For this problem, automatic network learning was not enabled for CGS. Instead, the rate of convergence experiments were run with two fixed, manually set BNs. In one case, a naïve Bayes classifier (i.e., all variables are independent of each other) is used. In the second case, *a priori* knowledge of the underlying problem structure is used to construct the BN. Recall from Eq. (9) and (10) that the overall objective function is a sum of sub-functions whose values depend on non-overlapping sets of three consecutive variables. The variables in each set of are not independent of each other. Therefore, the BN used model x_1 as a parent of x_2 and x_3, and x_2 as a parent of x_3, with this pattern continuing for every set of three consecutive variables (Figure 7). As always, the class variable c is a parent of all design variables.

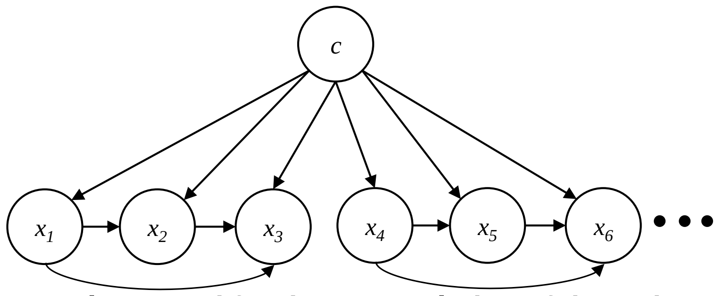


Figure 7: Manual BN used for the CGS solution of the order 3 deceptive

The other user-defined CGS and GA parameters that were used for the 30-variable deceptive function are shown in Table 3, and the results of the rate of convergence tests are shown in Figure 8.

Table 3. User-defined parameters for the deceptive function of order 3

CGS Parameters			*GA Parameters*	
N_{best}	20		Population Size	50
N_{batch}	20		Crossover Rate	0.8
P_L	n/a (fixed BN)		Mutation Rate	0.01

3.2.2 50-Item 0-1 Knapsack Problem

For this problem, automatic network learning was disabled for CGS, and a fixed naïve Bayes classifier was used. The other parameters used for both optimization methods are shown in Table 4, and the results of the rate of convergence tests are shown in Figure 9.

Table 4. User-defined parameters for the 50-item 0-1 knapsack problem

CGS Parameters			*GA Parameters*	
N_{best}	20		Population Size	50
N_{batch}	20		Crossover Rate	0.8
P_L	n/a (naïve Bayes)		Mutation Rate	0.01

3.2.3 System Design Problem

The user-defined optimization parameters for the system design problem are shown in Table 5, and the results of the rate of convergence experiments are shown in Figure 10.

Table 5. User-defined parameters for the system design problem

CGS Parameters			*GA Parameters*	
N_{best}	50		Population Size	100
N_{batch}	50		Crossover Rate	0.8
P_L	2		Mutation Rate	0.01

3.2.4 30-City Traveling Salesman Problem

The user-defined optimization parameters for the 30-city TSP are shown in Table 6, and the results of the rate of convergence experiments are shown in Figure 11.

Table 6. User-defined parameters for the 30-city TSP

CGS Parameters			*GA Parameters*	
N_{best}	50		Population Size	100
N_{batch}	50		Crossover Rate	0.8
P_L	2		Mutation Rate	0.01

3.2.5 Warehouse Location Problem

The user-defined optimization parameters for the warehouse location problem are shown in Table 7, and the results of the rate of convergence experiments are shown in Figure 12.

Table 7. User-defined parameters for the warehouse location problem

CGS Parameters			*GA Parameters*	
N_{best}	20		Population Size	50
N_{batch}	20		Crossover Rate	0.8
P_L	1		Mutation Rate	0.01

3.2.6 Welded Beam Design Problem

The user-defined optimization parameters for the welded beam problem are shown in Table 8, and the results of the rate of convergence experiments are shown in Figure 13.

Table 8. User-defined parameters for the welded beam design problem

CGS Parameters			*GA Parameters*	
N_{best}	50		Population Size	100
N_{batch}	50		Crossover Rate	0.8
P_L	2		Mutation Rate	0.01

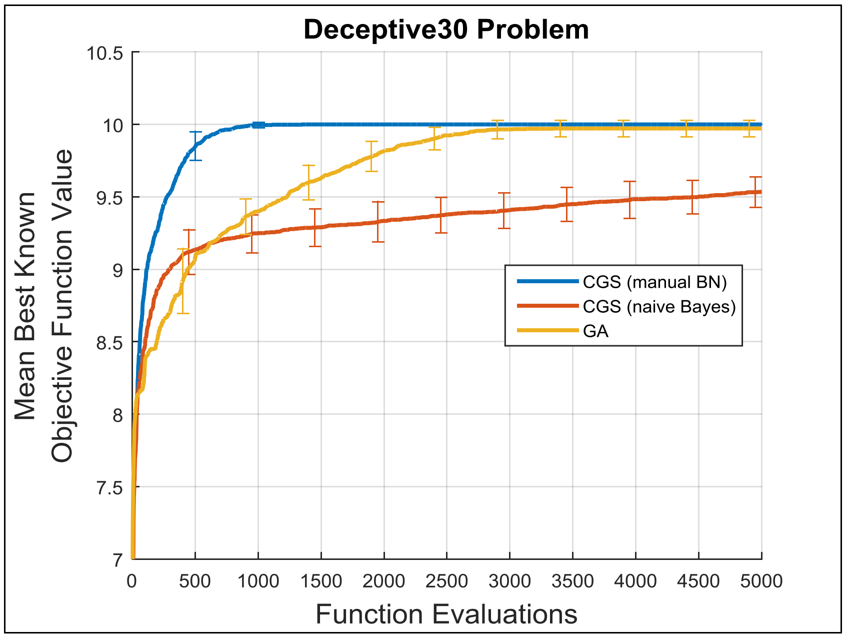


Figure 8: Deceptive function results

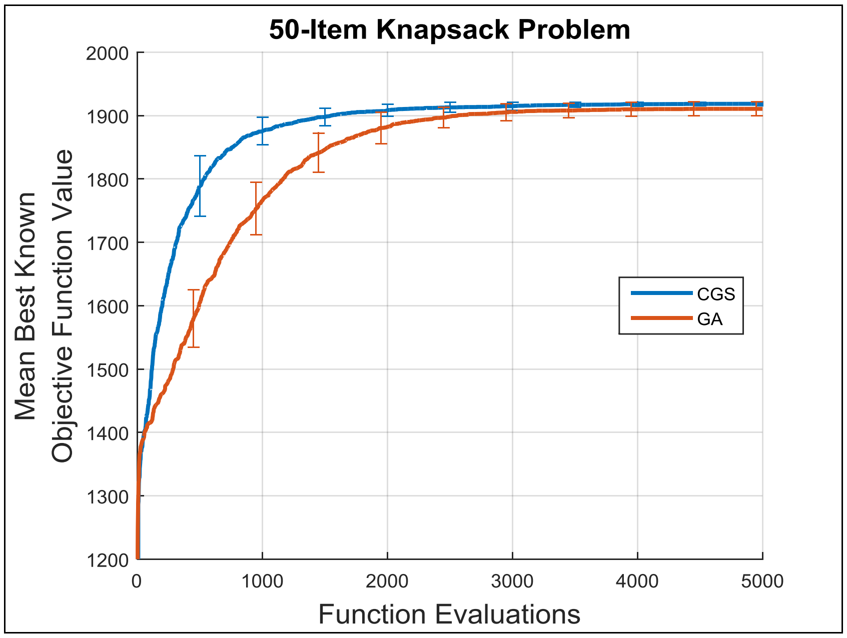


Figure 9: 50-Item knapsack results

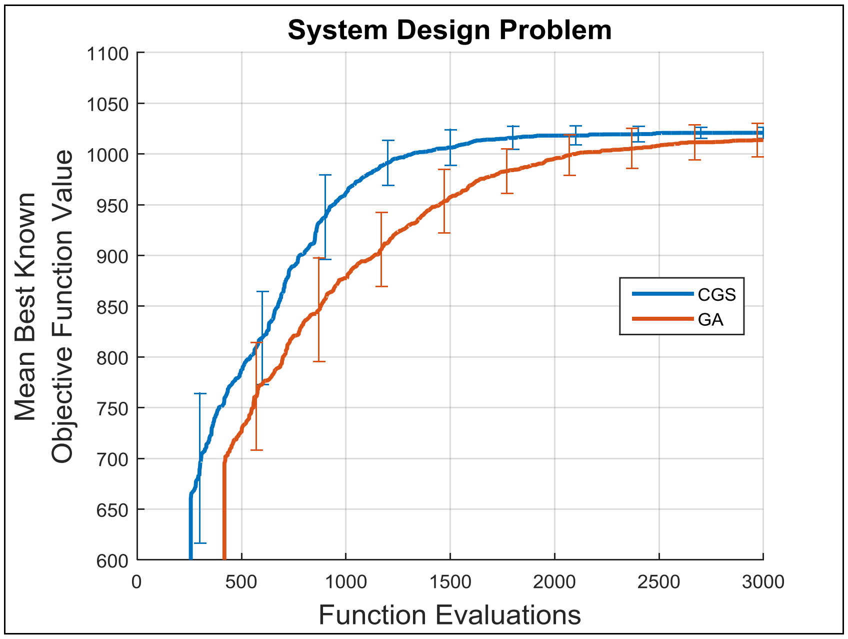


Figure 10: System design problem results

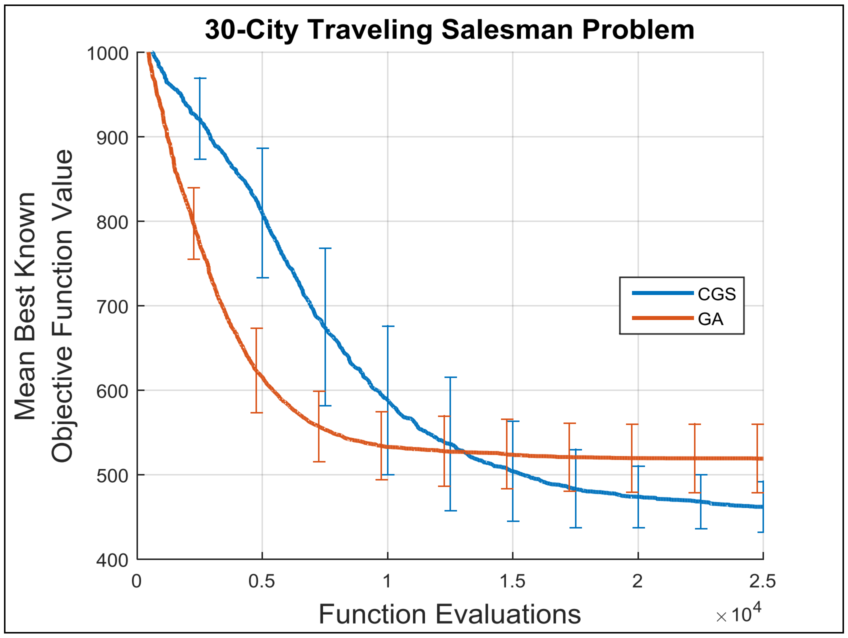


Figure 11: 30-city traveling salesman problem results

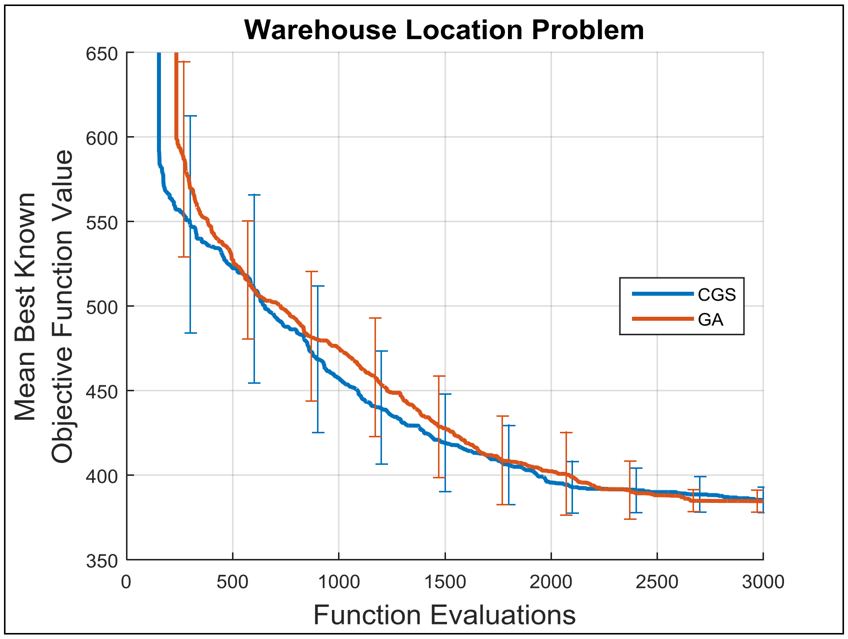


Figure 12: Warehouse location problem results

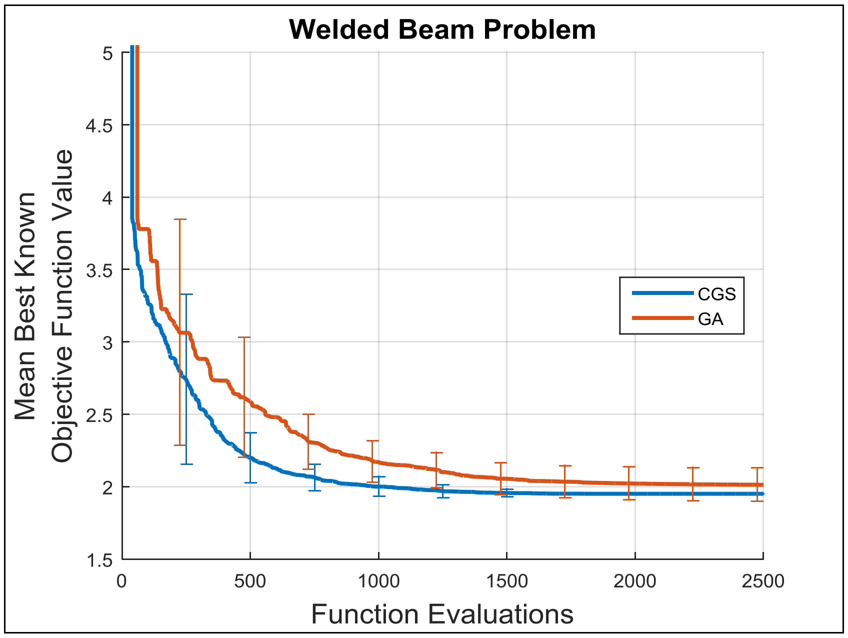


Figure 13: Welded beam problem results

3.2.7 Single-Objective Optimization Results Discussion

The results of the rate of convergence tests for the 30-variable deceptive function are shown in Figure 8. Initially, both instances of CGS outperform GA, but GA is able to find better solutions than the naïve Bayes implementation of CGS after roughly 700 function evaluations. However, when the manual BN of Figure 7 is used to capture the variable dependencies for CGS, the performance is significantly better than that of GA. On average, CGS with a well-constructed BN converges to the known global optimal solution (f = 10) quickly and consistently (all converged in less than 1300 evaluations). While it is difficult to tie this particular test problem to a real-world example, it effectively demonstrates the potential of CGS to solve difficult optimization problems provided that it is given a BN that accurately captures the dependencies amongst the design variables (or that it is able to discover those dependencies on its own).

The results of the rate of convergence tests for the 50-item knapsack problem are shown in Figure 9. For this problem, even though a fixed naïve Bayes classifier was used, CGS outperformed GA consistently at all stages of the optimization runs on average. These results demonstrate that it is not always necessary to force CGS to capture variable interactions to ensure good performance. That is, if the dependencies among variables are weak or non-existent, CGS may perform well on average relative to GA.

For the other test problems, which used the K2 algorithm for automatically learning the BN structure at each iteration, a common trend exists. In general, CGS converges towards the global optimum more quickly early in the solution process and remains ahead of GA up until termination. Near the upper limits on function evaluations, when both methods have effectively leveled off, CGS finds a better objective function value and has tighter standard deviation bars than GA, indicating better and more consistent performance even when both methods are given a large budget of function evaluations. The one exception to this trend occurs with the 30-city TSP (Figure 11). For this problem, GA initially improves towards the optimal design significantly faster than CGS. However, after approximately 13,000 function evaluations, CGS overcomes GA and identifies superior solutions, on average, given a larger allowance of function evaluations. It should be noted that neither of the two optimization techniques is particularly effective at solving this problem as it is formulated here. The vertical standard deviation bars for both methods are quite large, even near 25,000 evaluations, indicating inconsistent performance and low confidence that the global optimum is ever identified.

3.3 Multi-Objective Performance Comparison Results

Recall from Section 2.4 that the goal of multi-objective optimization is identify a set of non-dominated solutions, i.e. those for which there are no known designs outside of this set that are better in at least one objective and not worse in any others. With single-objective optimization, the best known objective function value provided a convenient way to plot a scalar, runtime performance metric vs. the number of function evaluations. Accessing or computing such a metric is less straightforward with multi-objective analysis. Some approaches include measuring the spread of the Pareto set (i.e., distance between extreme points), computing the hyper-volume under the Pareto set relative to an arbitrary reference point, or (when comparing two Pareto sets) computing the percentage of each Pareto set that is not dominated by the other. However, no single metric can capture all Pareto performance attributes that may be of interest [24]. Therefore, in this work, we simply plot the progress of the MOCGS and GA algorithms at specified intervals. Furthermore, due to the difficulty of specifying an “average” Pareto set across multiple runs, results of only a single optimization run for both MOCGS and GA are presented.

3.3.1 2D 50-Item Knapsack Problem

The 2D multi-objective version of the 50-item knapsack problem is similar to the single-objective version. The only difference is that the constraint on the maximum weight of all selected items is now an objective that is sought to be minimized. As before, the total value of all selected items is to be maximized.

The algorithmic details of MOCGS are provided in Section 2.4. For the GA, the fitness of an individual solution is based on the number of designs in the population that dominate that design in the multi-objective space, with a lower domination count being better. The same mutation and crossover operators from the single-objective experiments were used (random bit mutation and two-point binary crossover, respectively). A selector is used that selects designs that have fitness that is below a certain limit. In this case, the chosen limit is a domination count of 3. Note that this allows the population size to grow beyond the user-specified initial population size. Documentation of these operators is provided in the DAKOTA reference manual [18].

The user-defined MOCGS and GA parameters for the 2D 50-item knapsack problem are shown in Table 9. Both methods are set to terminate after performing 5,000 function evaluations.

Table 9. User-defined parameters for the multi-objective 50-item knapsack problem

CGS Parameters			*GA Parameters*	
N_{best}	20		Population Size	50
N_{batch}	20		Crossover Rate	0.8
P_L	2		Mutation Rate	0.1

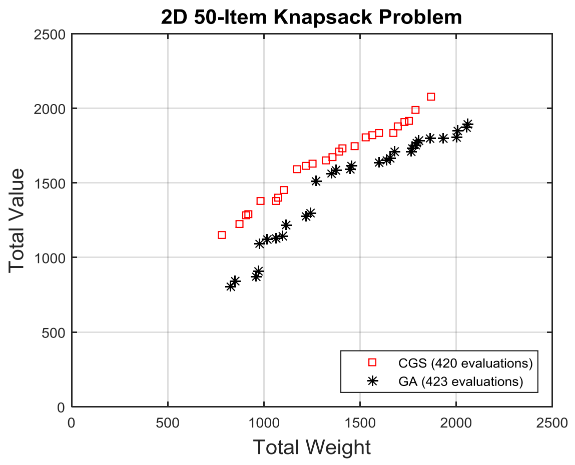
Results of the MOCGS and GA optimization runs are presented in Figure 14. Each plot (a-f) in the figure represents a snapshot of the current state of the optimization after specified number of function evaluations. While MOCGS evaluates exactly N_{batch} new designs with each iteration, the number of evaluations per iteration varies in this particular GA implementation. Therefore, the numbers of function evaluations for each method don't necessarily match in each snapshot. In all plots, the number of GA evaluations is greater than or equal to the number of MOCGS evaluations at the completion of that iteration.

3.3.2 Multi-Objective Results Discussion

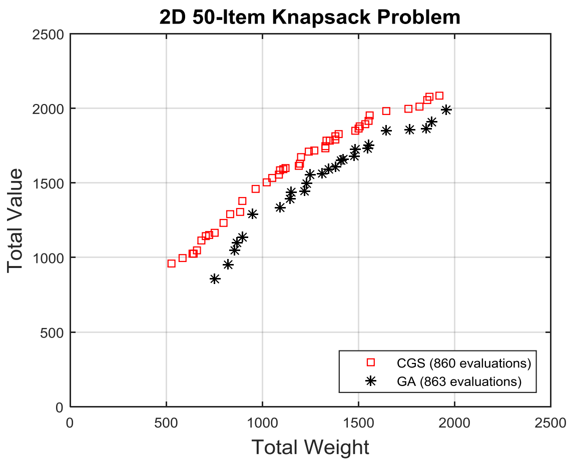
The objective of the 2D 50-item knapsack problem is to simultaneously minimize the total weight and maximize the total value of the selected items. Therefore, the goodness of the Pareto sets in Figure 14 can be assessed by identifying the ones that progress upwards and to the left of the plot area the fastest. When neither set appears to dominate the other, the set with a larger spread (distance between end points) and a larger quantity of non-dominated designs is best.

By viewing the plots in order of increasing function evaluations, it's clear for this particular optimization problem that MOCGS outperforms the GA. At the initial snapshot at approximately 420 function evaluations (Figure 14a), the entire MOCGS set dominates that of the GA. The MOCGS set remains dominant in every snapshot up to the final plot in Figure 14f. In other words, not only is the final solution after (roughly) 5,000 evaluations better, but it would be better even the optimizers were terminated early (which may be necessary in practice, for example, if the objective function were computationally expensive to evaluate). In addition, the spread of solutions is slightly better for MOCGS in the final plot. Lastly, MOCGS identifies 127 non-dominated solutions, while GA identifies only 104 non-dominated solutions.

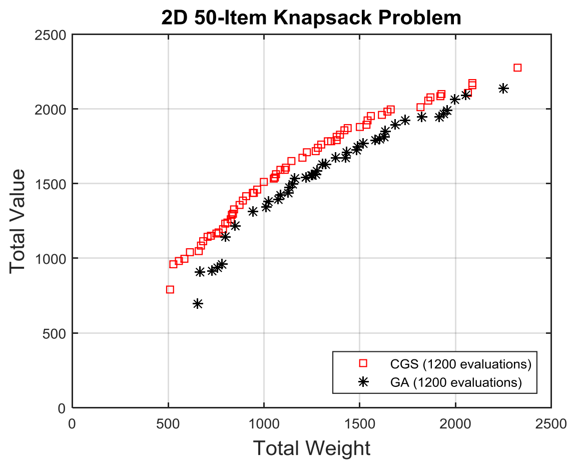
The results of this experiment provide strong evidence that MOCGS has the ability to outperform GA on some multi-objective problems. However, this isn't sufficient evidence to conclude that MOCGS is always or more often preferred. The results presented here are from one run with one particular setting of GA parameters and operators. It is also difficult, based on this one study, to make general conclusions about the kinds of problems for which one technique would be better. Additional case studies and experiments would help to provide this type of guidance.



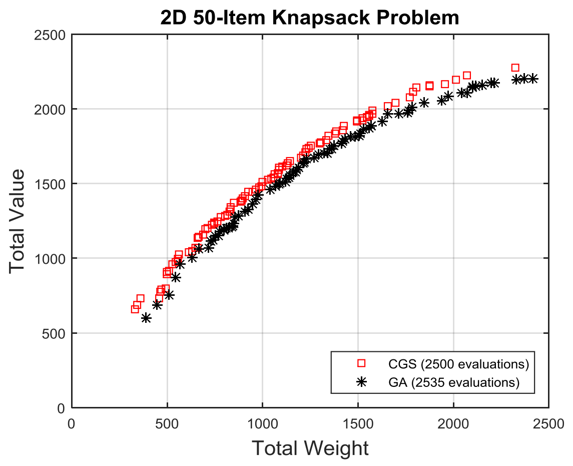
(a)



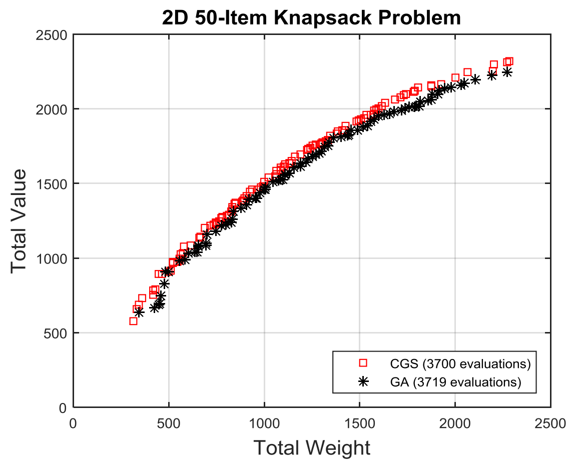
(b)



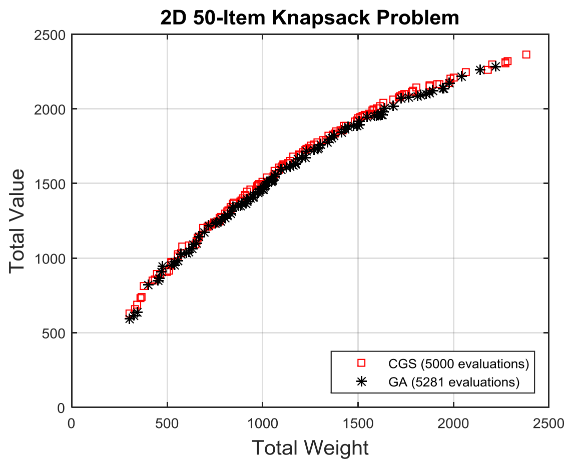
(c)



(d)



(e)



(f)

Figure 14: Multi-objective 50-Item knapsack problem Pareto progression

4 APPLICATION TO MICROGRID DESIGN

In October of 2012, Hurricane Sandy brought intense wind, rain, storm surges and claimed hundreds of lives. In the U.S., it is estimated that Sandy damaged or destroyed 650,000 houses, left 8.5 million customers without power, and caused 72 deaths [25]. It also left millions of homes and businesses without electric power. The storm was particularly devastating to the city of Hoboken, NJ, which was flooded for several days following the storm. The electric power grid was unavailable during this time. A well-designed microgrid (MG) may have been able to provide autonomous support to critical infrastructure during this outage. However, the decisions about what distributed energy resources (DERs) to install, where to install them, and how they should be interconnected have significant implications on cost and the ability to serve critical loads during autonomous MG operation. In this section, this problem is formulated as a single- and multi-objective optimization problem, and CGS is used to seek optimal MG configurations. The single-objective optimization results were published in the proceedings of the 2015 ASME Design Engineering Technical Conference [26] (SAND2015-0353C). This work is also an extension of a previous study in which a thorough energy surety analysis was performed for the City of Hoboken by Sandia National Laboratories [27].

4.1 Performance and Reliability Model Overview

For this work, the Performance and Reliability Model (PRM) was used to simulate grid outages and assess the performance of candidate MG designs. The PRM is a simulation code written in C++ at Sandia National Laboratories that is used to statistically quantify the performance and reliability of an MG operating in autonomous (islanded) mode. The PRM allows the performance of an MG to be quantified in terms of fuel usage, renewables penetration, renewables spillage, and other operational characteristics.

The PRM simulation relies on a representation of an unreliable power utility, specifically through the definition of failure and repair modes. In a typical PRM simulation, thousands of utility outages are simulated to ensure that the calculated statistics are representative of average behavior. The PRM models system behavior as a discrete sequence of events in time. At each event, logic is executed which may result in the scheduling of more events. The simulation proceeds until all events have been executed or until some stopping criterion is satisfied.

The primary input to the PRM is an MG topology specification. A topology specification includes descriptions of components such as electrical lines, busses, switches, transformers, generation assets and fuel sources, batteries, and inverters, in addition to the details of how these components are interconnected. Most components in the PRM can be modeled as potentially unreliable. To specify an unreliable component, failure modes must be defined where each mode represents a specific type or mode of component failure. Each failure mode specifies a failure time and a repair time statistical distribution that is sampled to generate the time characteristics of the failures and repairs, respectively. In addition to the physical layout and reliability characteristics of an MG, the PRM requires three configured controllers to define how the grid operates during different phases. A controller implements and executes the logic applied during the three phases of operation: grid-tied operation, MG startup operation, and autonomous MG operation. For additional details about the controllers and how they function in the context of PRM, refer to [26].

The primary outputs of the PRM are computed quantities and statistics, any of which can be used as optimization objective functions. All unreliable entities gather statistics detailing the outages they suffer, their durations, and the amount of downtime attributable to each failure mode. All levels of the grid for which it makes sense to express load service statistics track load served and not served. In the context of emergency autonomous operation, which is the focus of this work, the impact of MG topology (generators, lines, busses, and how these components are interconnected) on average load not served (LNS) is the quantity of interest, which is provided in kW-h per hour of bulk grid outage.

4.2 Hoboken Microgrid Design Challenge

There are 55 buildings (Figure 15) in Hoboken that are designated as critical and should remain operational during an emergency. The buildings include emergency services, pump stations (for flood control), affordable and senior housing units, grocery stores, gas stations, and government buildings. A complete listing of the buildings, their locations, and their estimated continuous loads is given in [26, 27].

The problem addressed here is to design a microgrid that minimizes average LNS, in kWh per hour of outage, of the 55 buildings without exceeding a limit on installation cost. There are two types of design variables that define this problem. First, the sizes and locations of natural gas generators are considered. Natural gas generators are chosen as the generator type because the high-pressure natural gas pipeline that runs throughout Hoboken is assumed to remain operational during an emergency, as it did during Hurricane Sandy. If installed, the generators are to be co-located with the 55 critical buildings. Generator installation costs are based on their size. The second type of design variable defines the topology of the microgrid. K-means clustering was used to group the 55 generator sites into 13 sub-grids based on geographic location. All buildings are assumed to be connected within each sub-grid, and each sub-grid can only be connected to adjacent sub-grids. The cost to connect each sub-grid is based on the cost per foot of underground cable. The building sub-grid assignments are tabulated in [26, 27].

In total, there are 153 design variables. Each variable is binary in the sense that the choice for it is to either do nothing or install something (either a generator or MG connection). There are 131 generator installation design variables, and 22 MG connection design variables. All 153 design variables and their installation costs are tabulated in [26]. Because of the emergency nature of this problem, fuel cost is not considered because fuel use is a short-term need that will be a relatively low cost in the long run.

LNS is evaluated by running a PRM simulation on candidate microgrid designs that are proposed by the optimizer. Time-dependent hourly load data for each building is estimated based on estimated peak loads. Utility outage frequencies and durations are sampled from probability distributions such that they have an average duration of one week and occur on average every 100 years. The PRM is used to simulate grid-tied and autonomous operation continuously over a period of 100,000 years. Therefore, the expected number of outages that are simulated with each run is 1,000. The expected outage count was selected by increasing the total simulation time until stable results were observed between runs of identical configurations.

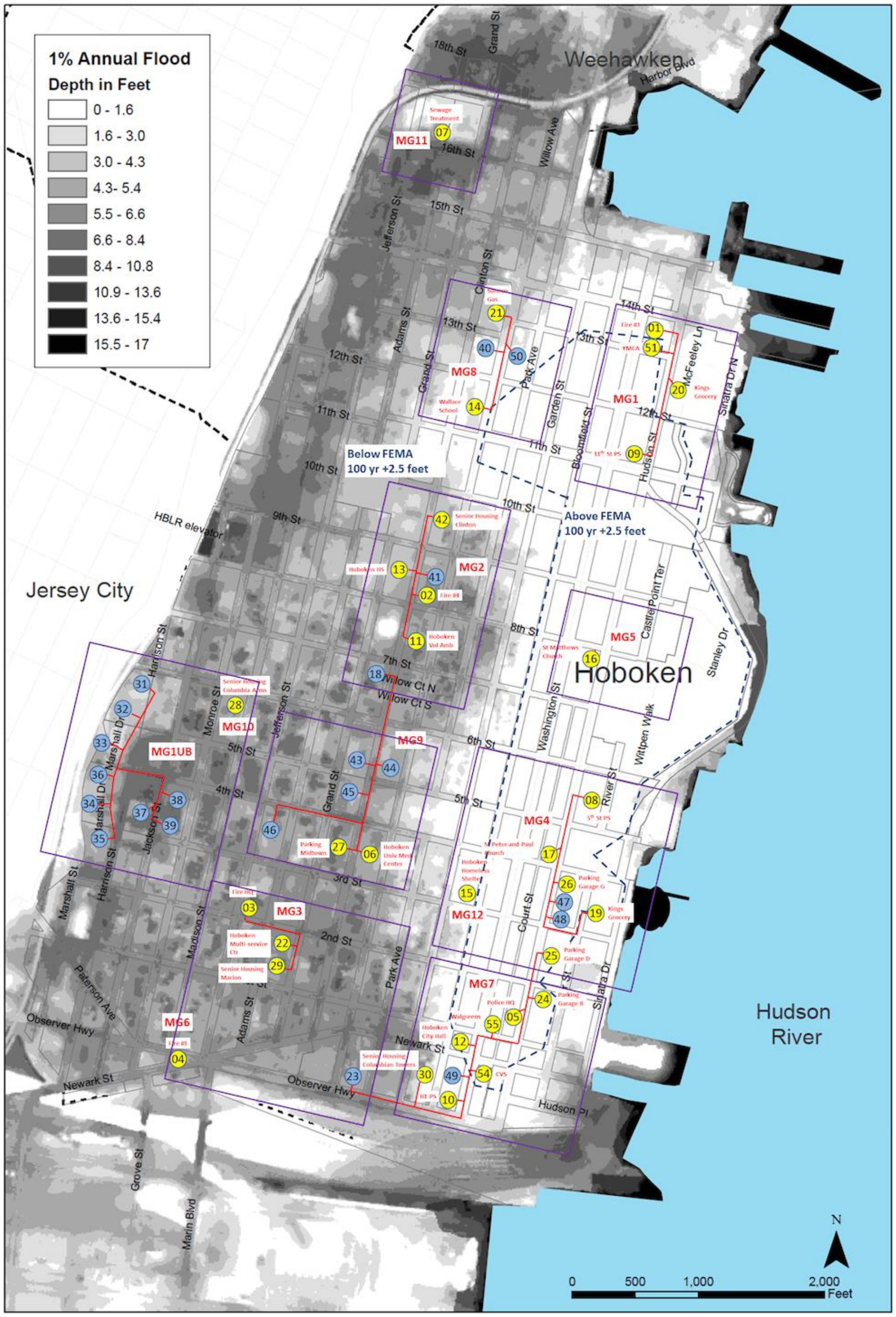


Figure 15: Critical Hoboken buildings shown on a flood map [27]

4.3 Single-Objective Results

In this section, CGS is used to identify Hoboken MG configurations that reduce LNS and do not violate the cost constraint. An upper limit on installation cost is set at $8M. GA was also used to solve the optimization problem for comparison purposes.

For CGS, N_{best} and N_{batch} are set to 50, and a fixed naïve Bayes classifier is used. To promote broad search early in the solution process, the first 100 candidates are sampled randomly from uniform distributions. After these initial 100 samples, the classifier is used to guide the search for the remainder of the optimization. For the GA, the population size is 50, and the crossover and mutation rates are 0.8 and 0.1, respectively. Similar to the single-objective experiments presented in 3.2, the GA used a two-point crossover operator, the random bit flipping mutation operator, and the "favor feasible" selector.

Rate of convergence tests for both methods are executed five times each with a fixed upper limit of 10,000 objective function evaluations, and the average result of the 5 trials is computed (Figure 16). Repeat evaluations of previously assessed solutions are not performed and are therefore not included in the rate of convergence results.

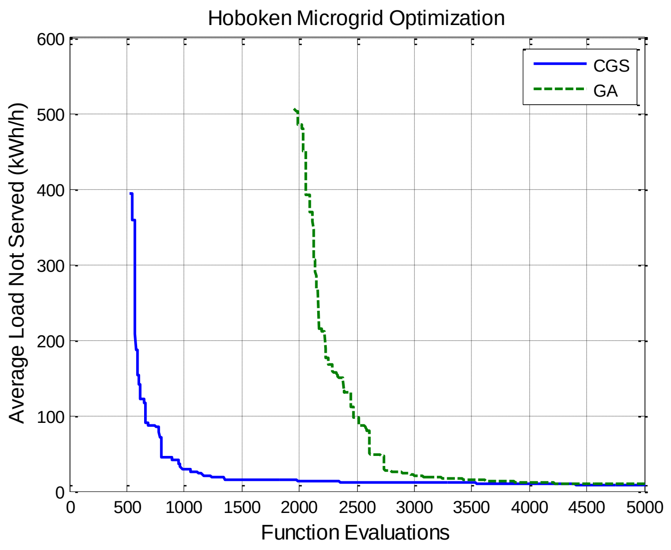


Figure 16: Single-objective rate of convergence results

On average, CGS identifies a near-optimal solution with roughly 2,000 fewer objective function evaluations than the GA. With the PRM configured as it is here, 2,000 evaluations equates to roughly 12 hours of simulation time. While 12 hours may not seem significant, this time may add up to many additional days of analysis time when analysis runs are being performed as part of an iterative investigation. Neither method immediately identifies solutions that satisfy the cost constraint in all five trials. Therefore, the curves in the figure begin at the lowest number of evaluations needed for all five trials to identify feasible solutions. Beyond the first 5,000 function evaluations, the curves level off at almost identical levels of average LNS until the algorithms were terminated at 10,000 evaluations. A discussion of the characteristics of the overall best solution found is provided in [26].

4.4 Multi-Objective Results

For the multi-objective version of the Hoboken MG design problem, the constraint on cost is treated as an additional objective to be minimized. For CGS, preliminary trials found that the algorithm performed best for this problem when N_{best} and N_{batch} are set to 50, and P_L is set to 3. Similarly, the GA performed well with an initial population size of 50, a crossover rate of 0.8, and a mutation rate of 0.1. As with the multi-objective knapsack problem that was presented in Section 3.3, the fitness of an individual solution is determined by its domination count, the random bit mutation operator was used, and the two-point binary crossover operator was used. Lastly, the "below limit" selector was used in which designs that have a domination count below 3 are chosen. Both methods are terminated after 60,000 function evaluations.

Results of the MOCGS and GA optimization runs are presented in Figure 17. Each plot (a-f) in the figure represents a snapshot of the current state of the optimization after a specified number of function evaluations. In all plots, the number of GA evaluations is greater than or equal to the number of MOCGS evaluations at the completion of that iteration.

In contrast to the 2D knapsack problem, CGS did not outperform GA by all measures. Early in the solution process (Figure 17a-b) the GA's non-dominated set appears to dominate some of the designs in CGS's non-dominated set; it also has broader spread and higher quantity of solutions. In the final four Pareto snapshots (Figure 17c-f), CGS and GA hold with similar-looking non-dominated sets. Upon close inspection of the final snapshot (Figure 17f), the GA set is slightly better. First, there are several CGS solutions in the "knee" of the curve that are dominated by designs in the GA set. Second, the spread of GA solutions in the objective space is greater overall. Lastly, GA finds a larger quantity of non-dominated designs (210 vs. 147 for CGS).

While the GA results are better in this one case, the difference is very slight. Given the small number of trials presented here, and the randomness that is inherent to the PRM simulations and in both optimization techniques, there is insufficient evidence to make general conclusions about which technique is better, on average, for this particular problem.

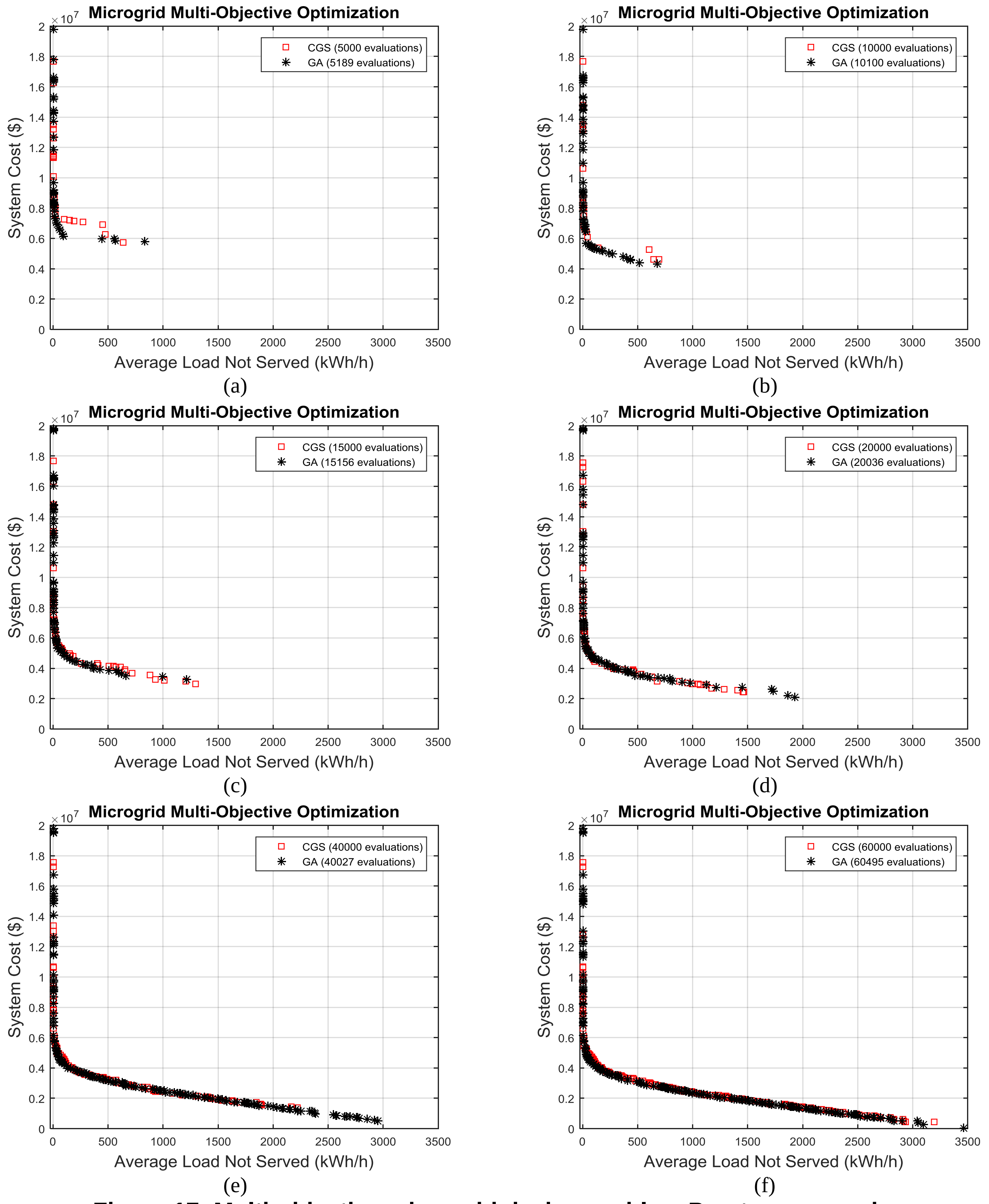


Figure 17: Multi-objective microgrid design problem Pareto progression

5 CONCLUSIONS AND FUTURE WORK

The overall goal of this LDRD project was to advance the state of the art of the CGS optimization algorithm to a level at which it offers a competitive alternative to GAs for solving large, non-linear, discrete-variable, black-box, single- and multi-objective optimization problems. The project was successful in that algorithmic improvements and the software implementation that resulted can significantly reduce the time required to solve such problems, especially if the objective function is computationally expensive to evaluate. Three major tasks were completed to achieve the goals of this effort:

- **Scalability to large problems** – Previously, CGS required that all discrete solutions in the design space be enumerated and processed individually by the classifier at each iteration of the algorithm. This issue was addressed by directly sampling the distributions that comprise the classifier to generate new exploratory designs. The idea was first published by Shahan et al. [2], and it was slightly modified and implemented as part of this LDRD.
- **Automatic learning of variable interactions** – A significant benefit of CGS over some other optimization techniques is that it requires no assumptions be made about the functional form of the problem constraint(s) or objective function(s) to be optimized. However, the results of the deceptive function of order 3 (Section 3.2.1) demonstrated that capturing variable interactions with the classifier's BN significantly improves optimization performance. In general, it is difficult or impossible to construct the BN manually when little is known about the objective function. Therefore, the K2 algorithm [15], an automatic BN learning algorithm, was implemented as part of this LDRD that learns these interactions on the fly and updates the classifier accordingly.
- **Extension to multi-objective optimization** – Prior to this work, CGS had only been developed for optimization problems with one objective. Many real world problems have multiple, competing objectives that must be traded off against each other. Furthermore, many constrained single-objective problems may yield more insightful results if they are formulated as unconstrained multi-objective problems. Therefore, this LDRD extended the ability of CGS to include multi-objective optimization by assigning class labels to the best solutions according the NSGA-II [16] sorting algorithm.

The latest CGS algorithm, complete with the advancements listed above, was implemented into a portable C++ library that can be easily integrated with existing internal modeling and optimization software tools or as a stand-alone solver that interfaces with an objective function evaluator. Furthermore, the implementation of the BN classifier that resulted from this work can be used for machine learning and classification tasks outside of the context of optimization.

Several avenues for future work have been identified. First, CGS should be integrated into existing Sandia optimization tools such as Technology Management Optimization (TMO), Whole System Trades Analysis Tool (WSTAT), and Microgrid Design Tool (MDT) as an optional alternative to GA. Second, learning the BN on the fly has significant implication on the performance of CGS. Additional work is needed to investigate how well the K2 algorithm is working and whether other network learning algorithms would perform better. Third, unlike the JEGA solver, MOCGS only has one option for determining the "goodness" of a design in relation to a set of others in multi-objective space. Implementing additional class-label assignment strategies (e.g., based on domination count) may improve MOCGS's ability to solve multi-objective optimization problems. Fourth, it would be interesting to investigate how and whether CGS and GA could work together as a hybrid optimization technique, in which the classifier could serve as an additional operator for generating and screening new candidate solutions with each new GA population. Lastly, a methodology for determining the types of problems for which CGS is best would be hugely valuable. At present, without explicit knowledge of the underlying objective function(s), it is very difficult to know in advance which technique will perform best.

6 REFERENCES

[1] P. B. Backlund, D. W. Shahan, and C. C. Seepersad, "Classifier-Guided Sampling for Discrete Variable, Discontinuous Design Space Exploration: Convergence and Computational Performance," *Engineering Optimization*, vol. 47, pp. 579-600, 2015.

[2] D. W. Shahan, P. B. Backlund, and C. C. Seepersad, "Classifier-Guided Sampling for Discrete Variable, Discontinuous Design Space Exploration," presented at the ASME International Design Engineering Technical Conferences (DETC), Portland, OR, 2013.

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. MA: Addison-Wesley, 1989.

[4] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, 1st ed. Ann Arbor: University of Michigan Press, 1975.

[5] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz, "Linkage Problem, Distribution Estimation, and Bayesian Networks," *Evolutionary Computation*, vol. 8, pp. 311-340, 2000.

[6] M. Pelikan, D. E. Goldberg, and F. G. Lobo, "A Survey of Optimization by Building and Using Probabilistic Models," *Computational Optimization and Applications*, vol. 21, pp. 5-20, 2002.

[7] S. Baluja, "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning," Carnagie-Mellon University, Dept. of Computer Science, Pittsburgh, PA CMU-CS-94-163, 1994.

[8] G. Harik, G. G. Lobo, and D. E. Goldberg, "The Compact Genetic Algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 287-297, 1999.

[9] J. S. De Bonet, C. L. Isbell, and P. Viola, "MIMIC: Finding Optima by Estimating Probability Densities," *Advances in Neural Information Processing Systems*, pp. 424-430, 1997.

[10] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz, "BOA: The Bayesian Optimization Algorithm," presented at the Genetic and Evolutionary Computation Conference (GECCO-99), Orlando, FL, 1999.

[11] H. Zhang, "The Optimality of Naive Bayes," presented at the Proceeding of the Seventeenth International Florida Artificial Intelligence Research Society Conference, 2004.

[12] D. Barber, *Bayesian Reasoning and Machine Learning*: Cambridge University Press, 2012.

[13] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, pp. 311-338, 2000.

[14] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data," Microsoft Research, Advanced Technology Division, MSR-TR-94-09, 1995.

[15] G. F. Cooper and E. Herskovits, "A Bayesian Method for the Induction of Probabilistic Networks from Data," *Machine Learning*, vol. 9, pp. 309-347, 1992.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorith: NSGA-II," *IEEE Transactions on Evolutionary Computation,* vol. 6, pp. 182-197, 2002.

[17] M. Pelikan, K. Sastry, and D. E. Goldberg, "Multiobjective Estimation of Distribution Algorithms," in *Scalable Optimization via Probabilistic Modeling,* 1st ed., Berlin: Springer, 2006, pp. 223-248.

[18] B. M. Adams, M. S. Ebeida, M. S. Eldred, J. D. Jakeman, L. P. Swiler, J. A. Stephens*, et al.*, "Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.2 Reference Manual," Sandia National Laboratories, Albuquerque, NM, Sandia Report: SAND2014-5015, 2014.

[19] L. D. Whitley, T. Starkweather, and D. Fuquay, "Scheduling Problems and the Traveling Salesman: The Genetic Edge Recombination Operator," *ICGA,* vol. 89, 1989.

[20] IBM, "IBM ILOG CPLEX Optimization Studio OPL Language User's Manual," IBM, 2014.

[21] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell, *Engineering Optimization: Methods and Applications*, 1st ed., New York: John Wiley and Sons, 1983.

[22] K. Deb and M. Goyal, "Optimizing Engineering Designs Using a Combined Genetic Search," in *Proceedings of the Seventh International Conference on Genetic Algorithms*, 1997.

[23] P. B. Backlund, "A Classifier-Guided Sampling Method for Early-Stage Design of Shipboard Energy Systems," Ph.D. Dissertation, Mechanical Engineering Dep., University of Texas at Austin, Austin, TX, 2012.

[24] T. Okabe, Y. Jin, and B. Sendhoff, "A Critical Survey of Performance Indices for Multi-Objective Optimisation," presented at the Congress on Evolutionary Computation, 2003.

[25] E. S. Blake, T. B. Kimberlain, R. J. Berg, J. P. Cangialosi, and J. L. Beven II, "Tropical Cyclone Report: Hurricane Sandy," National Hurricane Center, 12 February, 2013.

[26] P. B. Backlund and J. P. Eddy, "Autonomous Microgrid Design Using Classifier-Guided Sampling," presented at the ASME International Design Engineering Technical Conference (DETC), Boson, MA, 2015.

[27] J. Stamp, M. Baca, J. Eddy, R. Guttromson, J. Henry, R. Jensen, *et al.*, "City of Hoboken Energy Surety Analysis: Preliminary Design Summary," Sandia National Laboratories, Albuquerque, NM, Sandia Report: SAND2014-17842, 2014.

APPENDIX A: TEST PROBLEM PARAMETERS

Table A.1: 50-item knapsack problem parameters

Item ID	*Value*	*Weight*
1	3	94
2	41	70
3	22	90
4	30	97
5	45	54
6	99	31
7	75	82
8	76	97
9	79	1
10	77	58
11	41	96
12	98	96
13	31	87
14	28	53
15	58	62
16	32	89
17	99	68
18	48	58
19	20	81
20	3	83
21	81	67
22	17	41
23	3	50
24	62	58
25	39	61
26	76	45
27	94	64
28	75	55
29	44	12
30	63	87
31	35	32
32	11	53
33	21	25
34	45	59
35	43	23
36	46	77
37	26	22
38	2	18
39	53	64
40	37	85
41	32	14
42	78	23
43	74	76
44	61	81
45	61	49
46	51	47
47	11	88
48	85	19
49	90	74
50	40	31

Table A.2: System design problem technology option costs and utilities

Subsystem ID	*Number of Tech. Options*	*Metric*	*Technology Option ID*							
			1	2	3	4	5	6	7	8
1	4	Cost	52	54	2	86	-	-	-	-
		Utility	19	65	10	30	-	-	-	-
2	6	Cost	87	41	29	8	77	89	-	-
		Utility	84	15	27	50	36	73	-	-
3	5	Cost	66	60	1	49	56	-	-	-
		Utility	34	53	14	65	66	-	-	-
4	5	Cost	46	31	57	36	60	-	-	-
		Utility	17	5	17	99	31	-	-	-
5	2	Cost	71	50	-	-	-	-	-	-
		Utility	50	26	-	-	-	-	-	-
6	2	Cost	29	56	-	-	-	-	-	-
		Utility	7	28	-	-	-	-	-	-
7	8	Cost	49	1	48	78	76	64	52	83
		Utility	70	29	64	60	34	46	36	83
8	6	Cost	51	58	100	47	46	69	-	-
		Utility	52	25	56	55	52	11	-	-
9	5	Cost	42	41	39	40	9	-	-	-
		Utility	89	12	47	38	27	-	-	-
10	7	Cost	12	6	3	66	91	45	1	-
		Utility	4	39	70	77	93	33	72	-
11	5	Cost	33	68	13	96	45	-	-	-
		Utility	20	35	78	97	21	-	-	-
12	5	Cost	100	33	72	100	84	-	-	-
		Utility	11	30	33	16	21	-	-	-
13	2	Cost	22	51	-	-	-	-	-	-
		Utility	11	58	-	-	-	-	-	-
14	4	Cost	17	65	42	78	-	-	-	-
		Utility	92	5	4	10	-	-	-	-
15	2	Cost	81	27	-	-	-	-	-	-
		Utility	97	27	-	-	-	-	-	-

Table A.3: System design problem necessitations

Driving Technology Option			*Necessitated Technology Option*	
Subsystem ID	*Tech. Opt. ID*		*Subsystem ID*	*Tech. Opt. ID*
3	5		5	2
7	1		10	2
7	2		10	6
7	3		10	2
7	5		10	5
7	7		10	2
7	8		10	5
8	2		11	3
8	4		11	3
8	5		11	3
8	6		11	5
8	1		13	1
8	2		13	1
8	3		13	1
8	4		13	1
8	5		13	1
9	1		12	3
9	2		12	3
9	3		12	3
9	4		12	3
9	5		12	5
9	1		13	1
9	2		13	1
9	3		13	1
9	4		13	1

Table A.4: System design problem obviations

Driving Technology Option			*Obviated Technology Option*	
Subsystem ID	*Tech. Opt. ID*		*Subsystem ID*	*Tech. Opt. ID*
5	1		3	5
7	4		10	2
7	4		10	3
7	4		10	4
7	4		10	5
7	4		10	6
7	6		10	1
7	6		10	2
7	6		10	3
7	6		10	4
7	6		10	6
8	1		11	2
8	1		11	5
8	3		11	2
8	3		11	5

Table A.4: 30-city traveling salesman problem city coordinates [19]

City ID	*x-coordinate*	*y-coordinate*
1	54	67
2	54	62
3	37	84
4	41	94
5	2	99
6	7	64
7	25	62
8	22	60
9	18	54
10	4	50
11	13	40
12	18	40
13	24	42
14	25	38
15	44	35
16	41	26
17	45	21
18	58	35
19	62	32
20	82	7
21	91	38
22	83	46
23	71	44
24	64	60
25	68	58
26	83	69
27	87	76
28	74	78
29	71	71
30	58	69

Table A.5: Warehouse location problem warehouse capacities and warehouse-to-store supply costs [20]

Warehouse	**Bonn**	**Bordeaux**	**London**	**Paris**	**Rome**
Capacity	1	4	2	1	3
	Supply Costs				
Store 1	20	24	11	25	30
Store 2	28	27	82	83	74
Store 3	74	97	71	96	70
Store 4	2	55	73	69	61
Store 5	46	96	59	83	4
Store 6	42	22	29	67	59
Store 7	1	5	73	59	56
Store 8	10	73	13	43	96
Store 9	93	35	63	85	46
Store 10	47	65	55	71	95

DISTRIBUTION

1	MS 1188	Backlund, Peter B.	06133
1	MS 1188	Eddy, John P.	06133
1	MS 1188	Thompson, Bruce M.	06133
1	MS 1188	Lawton, Craig R.	06133
1	MS 1188	Nanco, Alan	06114
1	MS 0899	Technical Library	09536 (electronic copy)
1	MS 0359	D. Chavez, LDRD Office	01911
1	MS 0161	Legal Technology Transfer Center	11500