SAND2013-10292J

Classifier-Guided Sampling for Discrete Variable, Discontinuous Design Space Exploration: Convergence and Computational Performance

Peter B. Backlund, David W. Shahan, and Carolyn C. Seepersad*

Peter B. Backlund
R&D Scientist and Engineer
Sandia National Laboratories
pbbackl@sandia.gov

David W. Shahan
Research Scientist
HRL Laboratories, LLC
dwshahan@hrl.com

Carolyn C. Seepersad*
Associate Professor
Department of Mechanical Engineering
University of Texas at Austin
ccseepersad@mail.utexas.edu

* Corresponding author

Classifier-Guided Sampling for Discrete Variable, Discontinuous Design Space Exploration: Convergence and Computational Performance

Abstract

A classifier-guided sampling (CGS) method is introduced for solving engineering design optimization problems with discrete and/or continuous variables and continuous and/or discontinuous responses. The method merges concepts from metamodel-guided sampling and population-based optimization algorithms. The CGS method uses a Bayesian network classifier for predicting the performance of new designs based on a set of known observations or training points. Unlike most metamodeling techniques, however, the classifier assigns a categorical class label to a new design, rather than predicting the resulting response in continuous space, and thereby accommodates non-differentiable and discontinuous functions of discrete or categorical variables. The CGS method uses these classifiers to guide a population-based sampling process towards combinations of discrete and/or continuous variable values with a high probability of yielding preferred performance. Accordingly, the CGS method is appropriate for discrete/discontinuous design problems that are ill-suited for conventional metamodeling techniques and too computationally expensive to be solved by population-based algorithms alone. The rates of convergence and computational properties of the CGS method are investigated when applied to a set of discrete variable optimization problems. Results show that the CGS method significantly improves the rate of convergence towards known global optima, on average, when compared to genetic algorithms.

Keywords: Classifier-guided sampling, sequential sampling, metamodeling, direct search, stochastic optimization, Bayesian classification

1 Introduction

Many engineering design optimization problems involve discrete variables and discontinuous responses and are governed by simulation models of non-trivial complexity. There are many forms of discrete variable design problems, as noted by Huang and Arora [1]. The most challenging type involves variables that are not only discrete but also categorical in value, accompanied by responses that are discontinuous and non-differentiable. For example, the design of a suspension system requires selecting the types of components and their connectivity. Similarly, the system-wide design of a ship's power generation and storage capabilities and supporting infrastructure requires selection of component types, quantities, and

connectivity, and responses such as energy consumption and reliability are discontinuous functions of these discrete and categorical variables [2]. In many of these engineering design problems, the system models are too computationally expensive to support search algorithms that require large populations of solutions.

In response to this challenge, a classifier-guided sampling (CGS) method is introduced in this paper for solving engineering design optimization problems with discrete and/or continuous variables and continuous and/or discontinuous responses. The method merges concepts from metamodel-guided sampling and population-based optimization algorithms. The CGS method uses a Bayesian network classifier, in place of a metamodel, for predicting the performance of new designs based on a set of known observations or training points. However, the classifier assigns a categorical class label to a new design, rather than predicting the resulting response in continuous space. The CGS method uses these classifiers to guide a population-based sampling process towards combinations of continuous and/or discrete variable values with a high probability of yielding preferred performance. Accordingly, the CGS method is appropriate for discrete/discontinuous design problems that are ill-suited for conventional metamodeling techniques and too computationally expensive to be solved by population-based algorithms alone.

The CGS method is similar to metamodel-based design because it uses a surrogate model, in the form of a classifier, to guide the search for preferred design solutions. The general procedure for metamodel-based design is to generate an initial set of designs or training points, evaluate them with a computationally expensive simulation model, and then use them to train a metamodel to predict the performance of alternative designs [3,4]. Many different metamodeling techniques have been developed and compared [5-11]; some of the most frequently studied techniques for engineering design applications include polynomial regression, support vector regression (SVR) [12], kriging [13], multivariate adaptive regression splines (MARS) [14], NURBs-based metamodels [15], and radial basis functions (RBF). Since the metamodel is computationally inexpensive relative to the underlying simulation, it is

useful for efficiently searching for better designs, provided it is sufficiently accurate. Metamodel-based design optimization strategies, as reviewed by Wang and Shan [3], typically incorporate sequential sampling of the design space [16] to improve the accuracy of the metamodel and/or the quality of the resulting solutions. Optimization could be performed on the metamodel itself [17-19], or the metamodel could be used to guide a process of sampling towards optimal designs [20,21]. The focus of this paper is on the latter strategy.

One of the challenges in applying metamodel-guided sampling and optimization methods to discrete/discontinuous design problems is that metamodel-based methods are typically restricted to approximating responses that are smooth and continuous [22]. Methods have been developed to accommodate either discrete input variables or discontinuous responses, but not both. For example, Meckesheimer *et al.* [23], introduce a metamodeling approach for approximating models with continuous variables and a response with combined discontinuous/continuous behavior. Multiple metamodels are combined to model the entire response in a piecewise manner, with a state selecting "logic function" to determine which metamodel to use based on the values of the input variables. This method is effective on problems in which there are multiple regions of continuous responses, separated by discontinuities. Sharif, *et al.* introduce the discrete-variable mode pursuing sampling (D-MPS) method [24]. D-MPS is a direct sampling method that uses a metamodel to generate a set of so-called "cheap points". A cumulative distribution function (CDF) is used to sample from these points, based on the likelihood of resulting in high performance responses. The D-MPS method is best suited for problems with underlying behavior that is sufficiently continuous to support interpolation of the response across the discrete input variable values.

The CGS method focuses on design problems for which discrete and categorical input variables, combined with discontinuous responses, do not allow commonly used metamodeling techniques to interpolate between design points. For solving these types of problems, a classifier-guided method is developed that shares

the adaptive sequential sampling approach of metamodel-based design but also utilizes a combinatorial search technique with similarities to population-based stochastic search algorithms.

There are several types of stochastic search algorithms that are appropriate for discrete/discontinuous design problems, including simulated annealing [25] and genetic algorithms [26,27]. Like sequential sampling approaches, genetic algorithms maintain a set or population of candidate designs and seek to improve that population with each iteration of the algorithm. Unlike metamodel-based sequential sampling approaches, however, they do not use explicit models of the expected performance of candidate designs to select the next generation of candidate designs and rely on simple rules, such as crossover and mutation, instead. As a result, these algorithms often require large numbers of underlying function evaluations to identify high quality solutions. Metamodels are sometimes used to reduce the computational expense (e.g., [28]), but they are not suitable for discrete/discontinuous problems, as discussed previously.

In response to this challenge, a class of methods called estimation of distribution algorithms (EDAs) have been developed [29]. EDAs are based upon the genetic algorithm paradigm in that they operate on a population of solutions. However, they generate the offspring that constitute the next generation by sampling from a probability distribution that is based upon the highest performers. These methods vary primarily in the probability distributions that they use. The simpler methods, such as the compact genetic algorithm, assume independence between the design variables, leading to simple distributions whose parameters are easily determined [30]. The more sophisticated methods, such as the Bayesian optimization algorithm, can model interdependencies between design variables and consequently take longer to construct [31].

The method presented in this paper differs from existing EDAs in two important ways. First, it is based upon classifiers, which are models that seek to explicitly predict if a candidate design is in one of two or more categories, or classes,

based upon a set of points of known class called the training set. The classifier is used to guide the search towards promising solutions based on these categorical predictions of the quality of candidate designs. The resulting method is analogous to metamodel-guided sampling and optimization methods, except classifiers are used in place of other metamodeling techniques to accommodate discrete and categorical variables and discontinuous responses. Second, the classifier has a more sophisticated memory than GAs or EDAs, because it is trained on the entire set of training points, rather than limiting itself exclusively to the points in the current generation, resulting in less likelihood of converging to a local minimum. Specifically, the use of multiple classes with a probabilistic Bayesian network classifier allows the algorithm to distinguish between solutions with a high probability of preferred performance, solutions with a high probability of poor performance, and solutions for which the class membership is uncertain. The CGS method uses this information to explore the design space broadly in the early stages of the search process and then to guide the sequential sampling process increasingly towards combinations of continuous and/or discrete design variables with a high probability of yielding preferred performance. The search process occurs in a series of stages that are similar to the generations in population-based algorithms.

The details of the CGS method are described in Section 3, following a discussion of Bayesian network classifiers in Section 2. The rates of convergence of the CGS method for a set of standard test problems are presented in Section 4, followed by a discussion of these results in Section 5. Lastly, time complexities of the CGS method are investigated in Section 6.

2 Bayesian Network Classifiers

In machine learning, a classifier is used to assign categorical class labels to test points that have known feature attributes but unknown class labels [32]. The classifier is trained using a set of feature vector / class label pairs that are generally obtained experimentally. There are numerous methods available for classification,

including decision trees [33], learned rules [34], neural networks [35], Bayesian network classifiers [36], and support vector machines [37]. In this section, we present the details of a classifier that uses Bayesian networks (BN) [38] to create probability distributions that can be used for classification. Bayesian network classifiers are selected for use in the CGS method because they are a probabilistic method that provides the user with a probability for each class, in addition to a simple class label as an output. This property is critical to the sampling step of the CGS method that is explained in Section 3.

Using probability distributions for classification has a theoretical foundation in Bayesian decision theory [39]. Consider a K category classification, where c_k represents class k and $k = [1,2,\ldots,K]$. The classification is over a bounded D-dimensional design space for which a single design instance can be represented by a vector, $\mathbf{x} = [x_{i=1\ldots D}]^T$. If we can express the class conditional probability of a design instance given a category, $P(\mathbf{x}|c_k)$, then Bayes formula can be used to find the posterior probability of the class c_k given design parameters $\mathbf{x}$, $P(c_k|\mathbf{x})$, according to:

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

Design $\mathbf{x}$ is classified as a member of the class c_k that has the highest $P(c_k|\mathbf{x})$ when compared to all other $P(c|\mathbf{x})$. There are two key parameters of the classifier: the prior probability, $P(c_k)$, and the class conditional probability, $P(\mathbf{x}|c_k)$.

In general, the user may define the prior probabilities of each class however they see fit. For example, if there is no reason to believe that one class is more probable than any other, each $P(c_k)$ can be set equal for all k. In this research, the prior probabilities, $P(c_k)$, are estimated using the frequency of occurrence of each class in the training set according to

$$P(c_k) \cong \frac{N_k + 1}{N + K} \tag{2}$$

where N is the total number of training points, N_k is the total number of training points for class k, and K is the total number of classes. Equation (2), known as the "rule of succession" [40], precludes prior probabilities of zero that can result from small sample sizes by adding a single observation of each class prior to sampling. Equation (2) is appropriate for this application because there is prior knowledge that K classes are represented in the pool of unevaluated candidate solutions. For the task of estimating the class conditional probability, P(**x**|c), Bayesian Networks are described next.

Bayesian networks (BN) encode a factored joint probability distribution as a directed acyclic graph (DAG) where the edges from the parent nodes to a child node mean that the child node's probability is conditionally independent of its non-descendants, given its parent nodes [38]. In other words, setting the values of a node's parents makes that node dependent only upon its descendent nodes, i.e. the nodes that are reachable following any chain of arcs from that node.

Two extreme cases of network connectivity represent very well studied classifiers: the fully independent Bayesian network, also known as "Naïve Bayes" (NB), and the fully dependent case. Figure 1 depicts the graphical representation of the fully dependent BN classifier where the dependency on class is represented by the root C node of the graph.

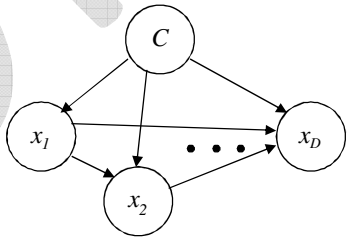

Figure 1: Fully dependent BN classifier

In the case of the fully connected BN classifier, the class conditional probability is given by

$$P(\mathbf{x} \mid c) = P(x_D \mid x_{D-1}, \ldots, x_1, c) \ldots P(x_2 \mid x_1, c) P(x_1 \mid c) \tag{3}$$

A drawback of the fully dependent case is that estimation of the class conditional probabilities is not practical in high dimensional space when a limited number of training points is available. The problem is greatly simplified by assuming the design variables are independent given the class, resulting in the Naïve Bayes classifier, as shown in Figure 2.

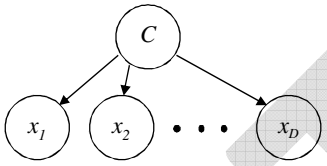

Figure 2: Naïve Bayes classifier

By assuming independence among all design variables, the class conditional probability is given by

$$P(\mathbf{x} \mid c) = P(x_D \mid c) \ldots P(x_2 \mid c) P(x_1 \mid c) \tag{4}$$

Although the assumption of independence is often incorrect, the NB classifier frequently performs well in practice because its classification decision may be correct even if its class conditional probability estimates are inaccurate [41]. Furthermore, Zhang [42] showed that NB is optimal even when dependencies exist if those dependencies cancel each other out. For the reasons described above, along with the need to estimate class conditional probabilities with relatively few training points, the NB classifier is used for the tests performed in this research. Frequency-based multinomial distributions are used to estimate the values of each $P(x_i|c_k)$. There are D distributions for each class, where D is the number of design variables.

3 The Classifier-Guided Sampling Method

The classifier-guided sampling (CGS) method uses a Bayesian network classifier to provide predictions of total design space performance. The classifier prediction is based on a set of training points from the expensive base model. Unlike a metamodel, a classifier cannot provide quantitative predictions on a continuous scale; however, it can provide a qualitative estimate of an objective function value by

pairing each candidate solution with a categorical class label. A classifier is used in the CGS method to assess a large set of candidate solutions quickly without requiring an expensive simulation for each point. The classifier outputs are used to guide the sampling process towards optimal or near optimal solutions. This method is especially useful for cases in which the concept of distance between points is irrelevant or undefined.

The CGS method (Figure 3) begins by executing expensive simulations for a set of randomly generated training points. The outputs of the expensive base simulation are assigned qualitative class labels (*e.g.* 'low' / 'high' quality) based on their objective function values. This task is achieved by defining a class threshold, T_C, which serves as a decision boundary for assigning class labels to training points based on their objective function values. If the design goal is to maximize an objective function and the training point has an objective function value higher than T_C, it is given a class label of 'high' quality. A training point is given a class label of 'low' quality if it is less than T_C. The rule is reversed for minimization problems in which case a training point is given a class label of 'high' if it is less than T_C.

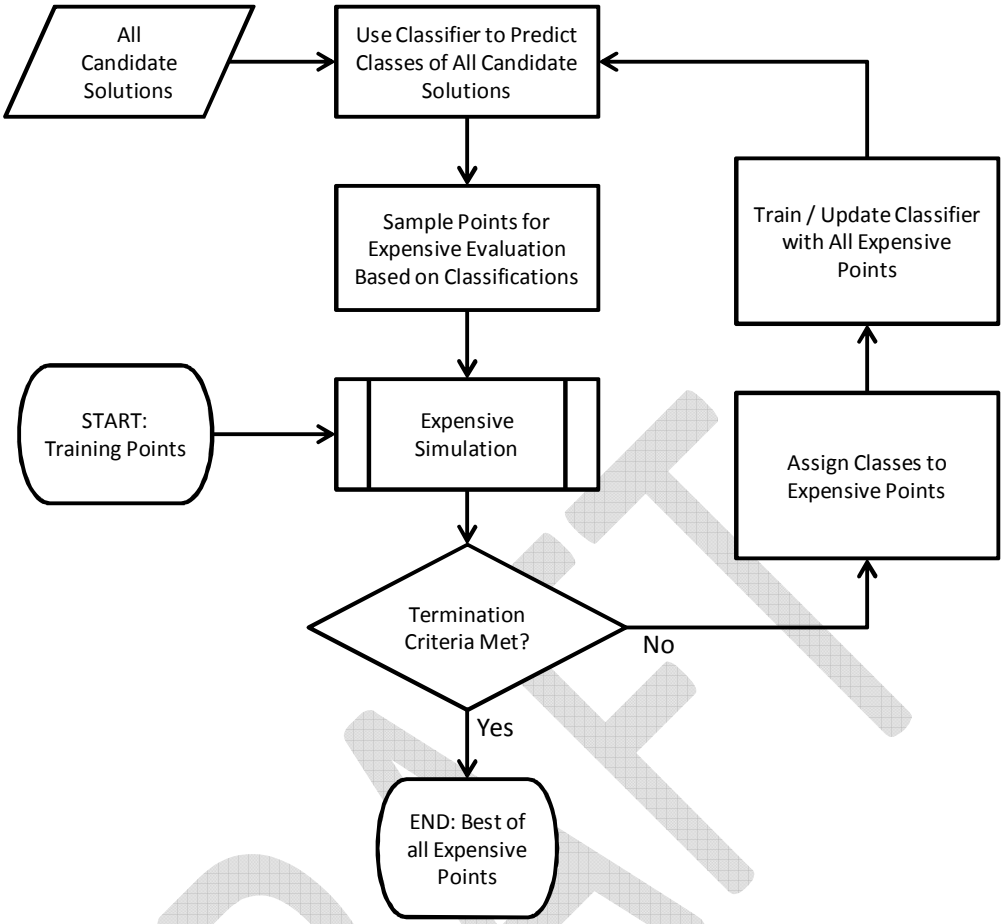

Figure 3: Classifier-guided sampling method

The class threshold, T_C, can remain fixed throughout the solution process, but performance is improved by allowing it to change as the classifier learns more about the performance space. In this research, T_C is allowed to change to help the classifier better differentiate between 'high' and 'low' quality solutions. When candidate points are evaluated by the classifier, no points are classified as 'high' if there are no points in the training set with this class label. Therefore, T_C is initially set so that 5% of the training points are assigned a class label of 'high' to give the classifier a sample of the characteristics of higher quality solutions. As more training points become available from expensive function evaluations and the current best known solution improves, the threshold is made more stringent to reduce the number of candidate points that are assigned a class label of 'high'. Conversely, if zero or very few candidate points are classified as 'high' at an intermediate iteration, T_C is made less stringent to allow

more points to be allocated to the 'high' class. This strategy of changing T_C is intended to avoid fixation on suboptimal solutions or local minima. The effect is similar to that of an annealing or cooling schedule in simulated annealing algorithms.

The classifier is trained using training point pairs that include both the training point design variable values and the corresponding class labels. After training, the classifier is used to predict the classes of all candidate solutions. For each unexplored point in the set of candidate solutions, the classifier returns the class label as an output. These so-called 'cheap points' provide categorical predictions of the quality of all candidate solutions that have not been evaluated with the expensive simulation.

Once the set of cheap points is generated using the classifier, their class labels and posterior probabilities, $P(c_k|\mathbf{x})$, are used to determine which unexplored points are to be sampled for the next batch of expensive simulations. In general, priority should be given to the points that are classified as 'high'. However, it is also important, especially early in the solution process, to sample points throughout the entire design space to improve classifier accuracy. The classifier is trained with a relatively small number of points, and it may not predict high quality solutions reliably without more knowledge of the design space. Therefore, three types of points are designated for sampling to achieve the necessary balance between depth and breadth of the search: high-certainty high-class points, high-certainty low-class points, and uncertain points. The high-certainty points are those with posterior probabilities greater than 0.6 for that particular class. For example, a point for which $P(c_{high}|\mathbf{x}) = 0.75$ and $P(c_{low}|\mathbf{x}) = 0.25$ is considered a high-certainty high-class point. Likewise, a point with $P(c_{high}|\mathbf{x}) = 0.10$ and $P(c_{low}|\mathbf{x}) = 0.90$ is considered a high-certainty low-class point. However, a point for which $P(c_{high}|\mathbf{x}) = 0.55$ and $P(c_{low}|\mathbf{x}) = 0.45$ is considered an uncertain point. Uncertain points can be from either class, as long as the maximum $P(c_k|\mathbf{x})$ of both classes is less than or equal to 0.6.

These three types of points are strategically sampled to infuse both depth and breadth into the sampling step of the CGS method. Let N_s be the number of points to be sampled during each iteration. In the CGS method, N_s is composed almost entirely

of high-certainty high-class points and uncertain points. The percentage of high-certainty high-class points in N_s is designated P_{hs}, and its value can change with each iteration. One possible progression of P_{hs} is shown in Figure 4. The user defines the value of P_{hs} for the first iteration, and its value increases linearly to a maximum of 90% with subsequent iterations. If several iterations occur with no improvement to the current best known solution, P_{hs} is adjusted to 50% or less and continues to increase linearly to 90% with each subsequent iteration. High-certainty high-class points are sampled randomly up to the prescribed proportion of N_s, and the remaining N_s points are selected randomly from the pool of cheap points that meet the criteria for uncertain points. Sampling of the third type of point, the high-certainty low-class points, is rare. These points are only sampled in the unusual event that the total number of high-certainty high-class points and uncertain points in the entire pool of cheap points is less than N_s. In this case, the high-certainty low-class points are sampled randomly until there are N_s points selected for expensive simulation in the next step of the CGS algorithm. Deliberately sampling low-certainty low-class solutions may seem counterintuitive, because the goal of optimization is to pursue the highest performing solutions with the least number of function evaluations. However, like many heuristic methods, CGS must implement a balance between exploration and exploitation in its search process. With CGS, exploration is especially important because it improves the accuracy of the classifier and enables it to better distinguish between high-performance and low-performance solutions. Therefore, low-certainty low-class points are sampled in this rare case to keep N_s constant and continuously improve the accuracy of the classifier.

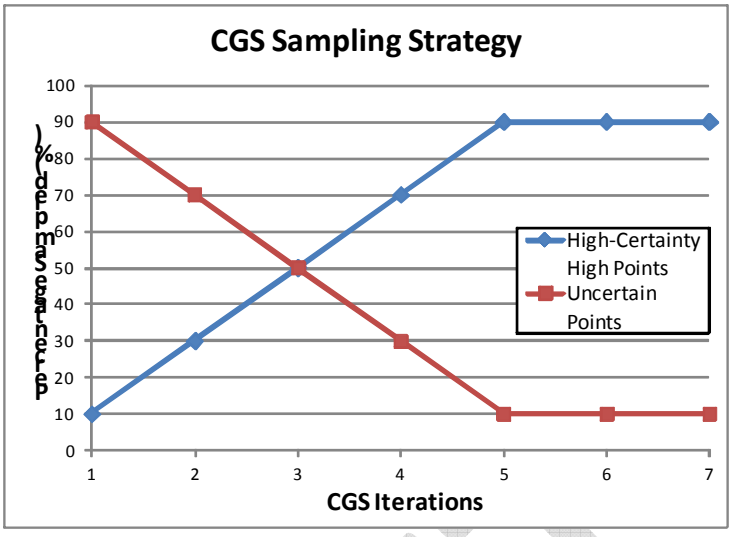

Figure 4: Sampling strategy

Once the expensive simulations are performed, termination criteria are evaluated. Some options for termination criteria include upper limits on the number of expensive simulation evaluations or algorithm iterations, or achievement of a desired objective function value. The CGS algorithm repeats until the termination criteria are met, at which point the best known solution is provided as the output.

The CGS method shares many of the same characteristics as direct sampling metamodel-based design optimization (MBDO) algorithms. The key difference in the CGS method is that a classifier is used to provide categorical estimates of the fitness of candidate solutions, while a metamodel is used in direct sampling MBDO algorithms to approximate a continuous function and provide quantitative estimates of the fitness of candidate solutions. In many combinatorial problems, the discrete variables have a discontinuous, non-differentiable effect on the response. A classifier is appropriate as an approximation of such functions because it provides categorical outputs that are discontinuous by nature.

The CGS method is a direct search optimization algorithm, similar to simulated annealing, genetic algorithms, and tabu search. That is, it requires no explicit knowledge of an analytical objective function. The unique property of the CGS method is its use of a classifier to reduce the total number of evaluations of the expensive base simulation. The CGS method uses a classifier as a substitute, or

surrogate of the base model. Although the classifier is unable to provide a quantitative estimate of the objective function output, it can give a qualitative estimate of the fitness of a candidate solution. Therefore, the hypothesized advantage of the CGS method is that it is able to reduce expensive base function evaluations by limiting expensive simulations to points that are predicted by the classifier to result in favorable objective function values.

The CGS method is a tool for rapid design space exploration for finding acceptable solutions quickly. It is best described as a direct sampling metamodel-based design method for discrete variable / discontinuous response problems, in which a classifier is used in place of a traditional continuous variable metamodel. In the following sections, the performance of the CGS method is compared to genetic algorithms when applied to discrete variable optimization problems.

4 Test Problems

In this section, the CGS method is applied to a selection of discrete variable, discontinuous response optimization problems. Genetic algorithms (GAs) and random search are also applied to the test problems and performance comparisons are made. These problems are selected to provide a broad range of problem types on which to test the CGS method. The problems feature a variety of variable types, including binary variables (knapsack problem), combinatorial variables (traveling salesperson problem), and a mix of categorical variables and continuous variables that are constrained to discrete values (welded beam problem). Each test problem features discrete variables that have a discontinuous effect on the problem's objective function, thus making them suitable for testing the performance of CGS.

4.1 20-Item Knapsack Problem

The objective of the knapsack problem is to select items from a set of available items that will maximize the combined value, V, of all selected items without exceeding a total weight limit, W. In the knapsack problem used in this study, there are 20 different items, and there is only one of each type of item available for

selection. Denoting a vector of binary variables $\mathbf{x} = (x_1, x_2,\ldots,x_{20})$ to represent the selection of items, the problem is formulated as follows:

$$\text{Maximize} \quad V(\mathbf{x}) = \sum_{i=1}^{n} v_i x_i \tag{5}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} w_i x_i \le W,\ x_i \in \{0,1\} \tag{6}$$

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 of the 20 available items are given in Table A.1 in Appendix A.

4.2 11-City Traveling Salesperson Problem

The traveling salesperson problem (TSP) is a frequently studied combinatorial optimization problem. Given a set of cities and their Cartesian coordinates, the objective is to find the shortest possible tour that visits each city exactly once and returns to the city of origin. If there are n cities to visit, there are $n!$ possible solutions to this problem. The problem size is reduced to $(n\text{-}1)!/2$ by specifying a city of origin at which the tour will always begin and end and by assuming symmetry, *i.e.* the distance between any two cities is the same regardless of direction traveled. A tour is represented by a vector of integer variables $\mathbf{x} = (x_1, x_2,\ldots,x_{10})$ where x_i is an integer from one to ten and each integer value can appear only once in each solution. By specifying the variables in this way, the objective function can be formulated as:

$$\text{Minimize} \quad D(\mathbf{x}) = d_o(x_1) + \sum_{i=1}^{n-2} d(x_i, x_{i+1}) + d_o(x_{10}) \tag{7}$$

where d is the Euclidean distance between cities x_i and x_{i+1} and d_o is the distance from the city of origin to the first or last city in the tour. The Cartesian coordinates of the origin and the 10 visited cities are given in Table A.2 in Appendix A.

4.3 Welded Beam Design Problem

The welded beam design problem, adapted from [43] and [44], is an engineering optimization problem that combines categorical and quantitative discrete variables. A rectangular bar is welded at one end and serves as a cantilever beam to carry a point load at the opposite end. The objective is to select the weld type, material, and geometric parameters that minimize the cost of fabrication. The two weld types and geometric parameters are shown in Figure 5.

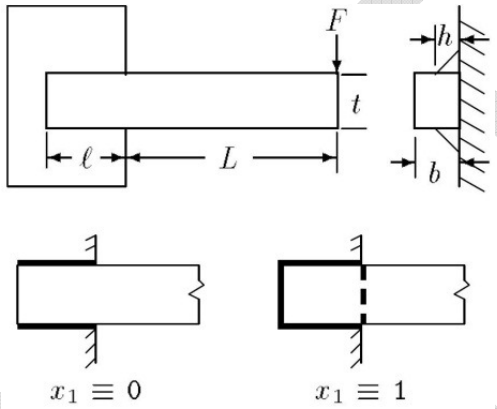

Figure 5: Welded beam problem [44]

The weld configuration is binary and describes whether two ($x_1 = 0$) or four ($x_1 = 1$) of the contact edges between the beam and base are to be welded. The weld and beam material is represented by one of four integers: $x_2 = 1$ (steel), $x_2 = 2$ (cast iron), $x_2 = 3$ (brass), and $x_2 = 4$ (aluminum). 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 variables that describe the geometric parameters are restricted to a finite set of discrete values (Table 1):

Table 1: Welded beam geometric parameter ranges

Variable	*Symbol*	*Min (in.)*	*Max (in.)*	*Step Size (in.)*
x_3	h	0.0625	0.5000	0.0625
x_4	t	7.500	10.000	0.125
x_5	b	0.0625	1.0000	0.0625
x_6	l	0.125	3.000	0.125

If the six design variables described above are represented by the vector $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6)$, the objective function and constraints are given by

$$\text{Minimize} \quad f(\mathbf{x}) = (1+c_1)x_3^2(x_6 + x_1x_4) + c_2x_4x_5(L+x_6) \tag{8}$$

$$\text{Subject to} \quad \begin{cases} g_1(\mathbf{x}) = S - \sigma(\mathbf{x}) \geq 0 \\ g_2(\mathbf{x}) = P_c(\mathbf{x}) - F \geq 0 \\ g_3(\mathbf{x}) = \delta_{\max} - \delta(\mathbf{x}) \geq 0 \\ g_4(\mathbf{x}) = 0.577S - \tau(\mathbf{x}) \geq 0 \end{cases} \tag{9}$$

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 force of the load F is 6,000 lb., the extended length L of the beam is 14 in., and the maximum allowable deflection $\delta_{\max}$ is 0.25in. The material costs and properties are provided in the Appendix.

4.4 Test Metholology and Implementation Details

Performance comparison of the CGS method and a GA is achieved by executing a set of rate of convergence tests in which the current best solution versus the number of objective function evaluations is recorded. This test provides a visual measure of how quickly each method converges towards known global optima.

The performance of the CGS method and the GA depends in part on user-defined tuning parameters. Care must be taken when selecting these parameters to ensure that a fair comparison is conducted and presented. For the CGS method, the three parameters of interest are the number of training points in the initial training set, N_{tr}, the number of new points to sample for expensive evaluation at each iteration, N_s, and the initial value of the percentage of high-certainty high-class points to sample from the pool of cheap points, P_{hs}. For the GA used in this study, the three parameters of interest are the population size, N, the percentage of population members selected for reproduction via crossover operations, P_c, and the percentage of encoded bits to mutate in each generation, P_m.

The optimal settings for these parameters are highly problem-specific. For example, there is a tradeoff between the size of the initial CGS training set and overall rate of convergence. This phenomenon is a direct result of the fact that the initial

CGS training points in this study are selected randomly and without guidance from the classifier. If the size of the initial training set is too small, the classifier used in the first iteration does not have sufficient information about the behavior of the design space and may be unable to effectively guide subsequent sampling. If the initial training set is large, a significant amount of time may be wasted searching the design space ineffectively (randomly), because during this phase the classifier is not yet being leveraged to guide the search towards superior solutions.

For both CGS and GAs, each parameter is evaluated at three different levels for each test problem, resulting in 27 possible parameter combinations. Each of the 27 parameter combinations are tested ten times for each test problem, and the parameter combinations that enable the GA or CGS to identify a predetermined objective function value with the fewest function evaluations, on average, are selected for inclusion in the main comparison study. The three parameters levels for each test problem are shown in Tables 2 and 3. The parameter values in boldface are those for which the CGS method and GA identify a predetermined objective function value with the fewest function evaluations on average.

Table 2: Classifier-guided sampling user-defined parameters

Test Problem	N_{tr}	N_s	P_{hs}
Knapsack	[50, **100**, 200]	[**50**, 100, 200]	[0.1, 0.5, **0.9**]
Traveling Salesperson	[**100**, 200, 400]	[**100**, 200, 400]	[0.1, **0.5**, 0.9]
Welded Beam	[**50**, 100, 200]	[50, **100**, 200]	[0.1, **0.5**, 0.9]

Table 3: Genetic algorithm user-defined parameters

Test Problem	N	P_c	P_m
Knapsack	[25, **50**, 100]	[0.6, 0.8, **1**]	[0.005, 0.010, **0.015**]
Traveling Salesperson	[25, **50**, 100]	[**0.6**, 0.8, 1]	[0.005, **0.010**, 0.015]
Welded Beam	[25, 50, **100**]	[0.6, **0.8**, 1]	[0.005, **0.010**, 0.015]

A detailed discussion of the GA encodings, selection method, crossover, and mutation method is provided by [2].

In addition to the parameters described above, the Bayesian network structure must be selected. For the knapsack and the welded beam problems, the Naïve Bayes (NB) classifier is used in the CGS algorithm. This Bayesian network structure is chosen because relatively few training points are needed to populate the class

conditional probability distributions with adequate density. Furthermore, although the assumption of independence is often incorrect, the Naïve Bayes classifier frequently performs well in practice because its classification decision may be correct even if its class conditional probability estimates are inaccurate [41]. Furthermore, Zhang [42] shows that Naïve Bayes is optimal even when dependencies exist, if those dependencies cancel each other out.

Due to the strong conditional dependence among variables in the TSP, the NB classifier does not provide adequate classification accuracy, and the CGS method fails to converge towards the known global optimum in some trials. This result is directly related to the strong assumption of independence built into the Naïve Bayes classifier. In light of the above discussion, an Augmented Naïve Bayes (ANB) network structure [42] is used for the TSP in this research. In the ANB classifier, each variable node is conditionally dependent on its adjacent variable node (Figure 6). While this BN structure does not capture the highly dependent nature of the variables in the TSP, it improves performance by enabling the classifier to recognize dependencies between adjacent cities in the TSP tour.

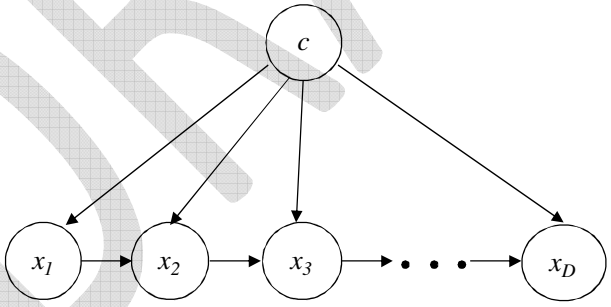

Figure 6: Bayesian network structure for the TSP

As was discussed in Section 3, the CGS class threshold, T_c, is allowed to evolve to improve performance. In these studies, if the number of high-certainty high-class points is greater than N_s, then the class threshold for the next iteration, $T_{c,i+1}$ is reduced (for minimization problems) or increased (for maximization problems) by half the difference between the current class threshold, $T_{c,i}$, and the current best known solution:

$T_{c,i+1} = T_{c,i} - 0.5\left(T_{c,i} - f^*\right)$	(10)

where f^* is the current best known solution. Likewise, if the number of high-certainty high-class points is less than or equal to 10% of N_s, then the class threshold is increased (for minimization problems) or reduced (for maximization problems) by 5% of the difference between the current class threshold and the initial class threshold:

$T_{c,i+1} = T_{c,i} + 0.05\left(T_{c,0} - T_{c,i}\right)$	(11)

where $T_{c,0}$ is the initial class threshold, set so that 5% of the initial training set is assigned a class label of "high." If neither of these conditions are true, the class threshold does not change.

In addition to the class threshold, the percentage of high-certainty high-class points, P_{hs}, can also change with each iteration. In these experiments, the user defines the value of P_{hs} for the first iteration according to the values indicated in Table 2. Its value increases linearly by 20% with each iteration up to a maximum of 90%. If two iterations occur with no improvement to the current best known solution, P_{hs} is adjusted to 50% and continues to increase linearly to 90% with subsequent iterations.

A purely random search is also performed on the three test problems to provide an additional comparison. For the random search method, each new trial begins with the set of all possible solutions. Samples are randomly pulled from the exhaustive solution space without being replaced to avoid repeat evaluations of identical solutions. As the iterations proceed, the solution with the best objective function value is retained until a solution with a better objective function value is randomly sampled.

Random number generation is a key element of the solution process in all three of the compared methods, and performance varies with each repeat solution of the same test problem. Therefore, the convergence rate tests are repeated 50 times for each method on all test problems, and the primary performance comparison is based on the mean performance. In addition to the mean performance, the 10^{th} and 90^{th}

percentiles of all 50 tests are calculated at select points to provide an indication of the extent to which performance varies about the mean.

For benchmarking purposes, the global optima for each test problem are identified using exhaustive enumeration. The 20-item knapsack problem has two optima with identical objective function values, while the TSP and welded beam problem have single global optima.

4.5 Results

The results for the rate of convergence tests for the knapsack, traveling salesperson, and welded beam problems are shown in Figures 6-8. The error bars at select function evaluation values represent the $10^{th}/90^{th}$ percentiles. The percentiles are slightly offset from each other to prevent overlapping and improve visual clarity.

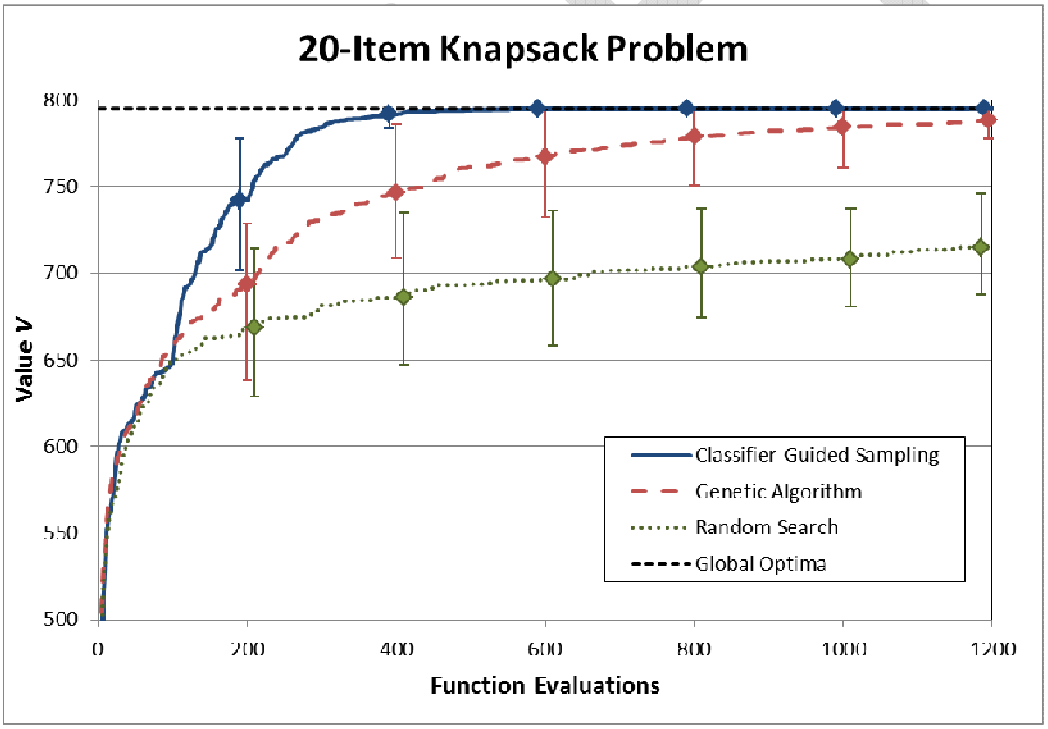

Figure 7: Knapsack problem convergence test results with $10^{th}/90^{th}$ percentile bars

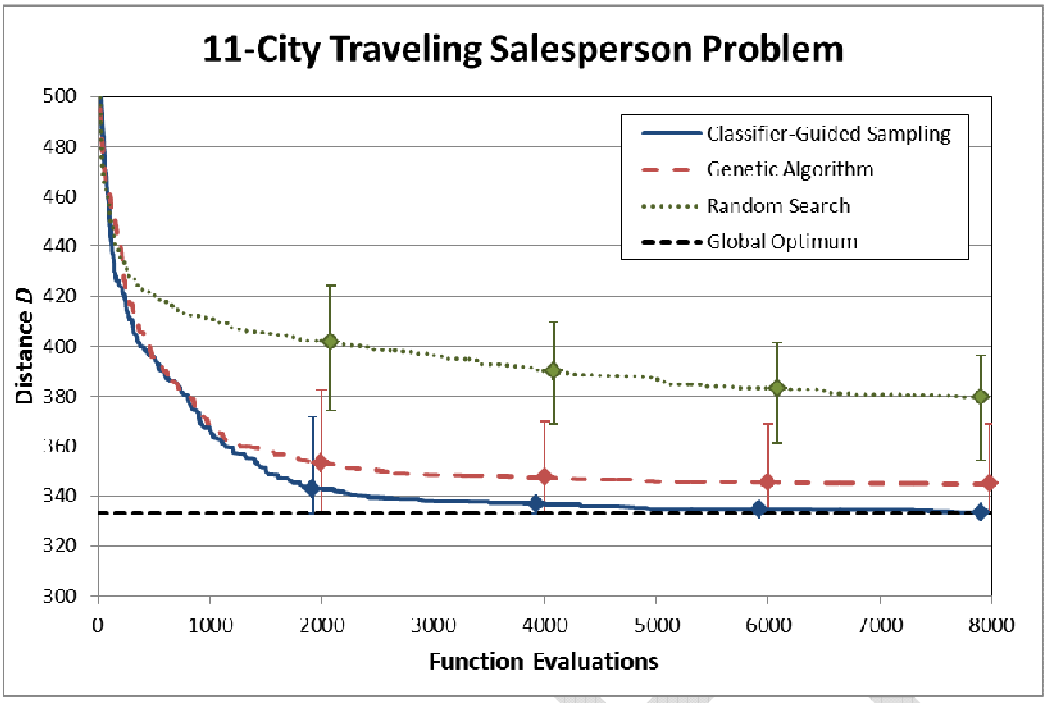

Figure 8: 11-city TSP convergence test results with 10^{th}/90^{th} percentile bars

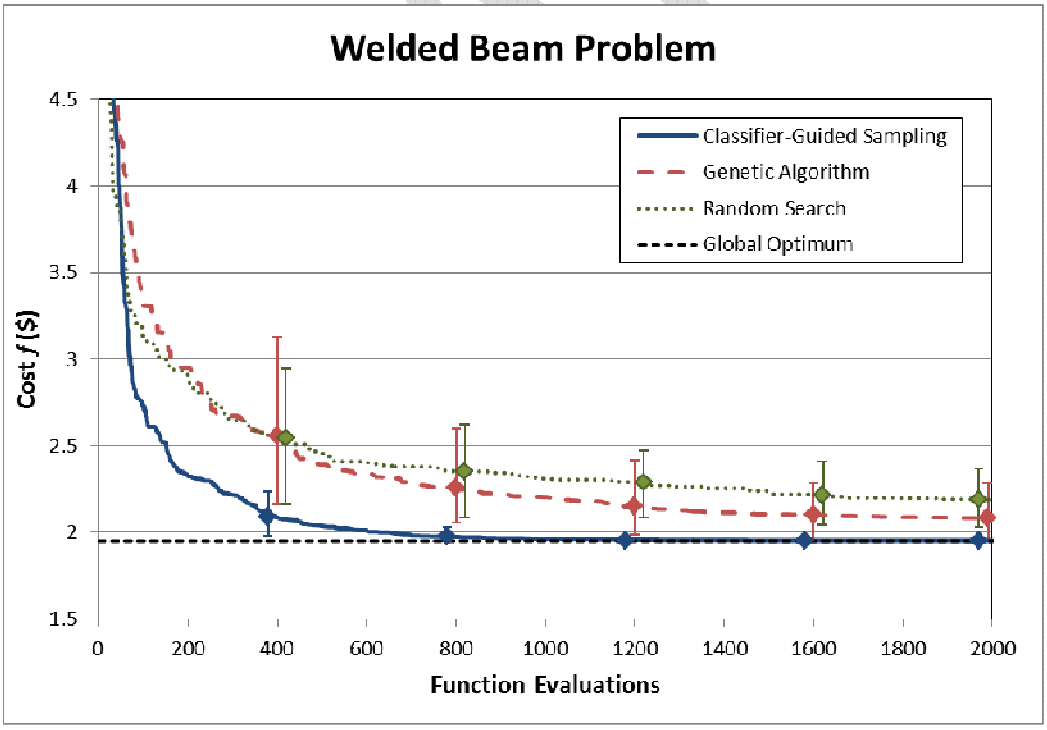

Figure 9: Welded beam problem convergence test results with 10^{th}/90^{th} percentile bars

Initially, for the first few function evaluations, the CGS method performs no better than random search. This behavior is expected because the first batch of points

selected for function evaluation is selected randomly to serve as the initial training set for the classifier. Once the CGS becomes active after evaluating the initial training set (100 function evaluations for the knapsack and TSP, 50 function evaluations for the welded beam), the average best known solution approaches the known optimum rapidly compared to random search. This behavior can be seen on all three of the test problems.

For the 20-item knapsack problem (Figure 7), the CGS method converges on one of the two global optima with just over 400 function evaluations, while it takes the GA over 1,200 function evaluations, on average. In addition to faster average rate of convergence, the CGS method is less prone to fixation on suboptimal solutions. One of the two global optima is found with the CGS method with fewer than 1,200 function evaluations in all 50 of the trials. However, in three of the 50 trials the GA fails to find the optimal solution in fewer than 2,500 function evaluations.

For the 11-city TSP (Figure 8), the performances of the CGS method and the GA are similar up to about the first 1000 function evaluations. However, beyond this point the GA begins to flatten out while the CGS method continuous to converge towards the optimal solution. The CGS method finds the global optimum with fewer than 3,000 function evaluations in the majority of the 50 trials, and in all 50 trials it does so in fewer than 8,000 function evaluations. However, in many of the trials the GA fails to converge on the global optimum with fewer than 10,000 function evaluations.

For the welded beam problem (Figure 9), the CGS method drastically outperforms the GA, and the GA only marginally outperforms random search. In most cases the CGS method identifies the global optimum in fewer than 800 objective function evaluations. However, in three of the 50 trials the CGS method fails to find the optimum with fewer than 2,000 function evaluations. The GA struggles with the welded beam problem; the average best solution it identifies in 2,000 function evaluations is equal to that which the CGS method is able to find with about 450 function evaluations.

The 10th/90th percentile bars exhibit similar behavior for all three test problems. Early in the solution process, the CGS solutions in the 10th percentile are inferior to the GA solutions in the 90th percentile. This trend indicates that if a limited number of function evaluations are allowed, the CGS method would outperform the GA in most, but not all cases. However, as the number of function evaluations increases, the CGS solutions that are in the 10th percentile are superior or equal to the GA solutions that are in the 90th percentile. Therefore, given an adequate number of function evaluations, the CGS method will perform better than the GA in most cases, and equal to the GA in a few select cases.

5 Discussion

The results of this study show that the CGS method converges to the known global optima with significantly fewer objective function evaluations. These results support the hypothesis that a discrete variable classifier can be used in place of a continuous variable metamodel to solve computationally expensive, discrete variable / discontinuous response design problems.

Although the CGS method is effective for solving the test problems presented in this study, the number of initial training points and the average number of function evaluations required to find the optimal solution are high. A minimum of several hundred function evaluations is required to achieve convergence to a global optimum in the problems tested here. If hundreds of function evaluations are required for accurate classification and convergence toward a global optimum, then the method would be useful only for problems of moderate computational expense (i.e., with execution times on the order of minutes, rather than days).

Convergence to a precise known global optimum is not the only measure of optimization algorithm performance. In many practical situations, designers are tasked with identifying performance-improving solutions in a limited amount of time. In such cases, a more meaningful measure of optimization algorithm performance is how close the algorithm can get to the global optimum with a fixed number of

allowable objective function evaluations. By this measure, the CGS method performs favorably for the problems presented here. For example, for the welded beam problem in Figure 9, the CGS method converges to within approximately 10% of the global optimum with only 400 function evaluations, on average, whereas the GA requires over 1500 function evaluations, on average, to reach a similar solution.

The superior performance of the CGS method compared to the GA can be partially attributed to the lack of repeated objective function evaluations for identical solutions. Repeat evaluations are completely avoided with the CGS method for the particular problems solved here, because all possible candidate solutions can be enumerated in the initial list of unevaluated points, also referred to as "cheap points." When a cheap point is sampled for expensive evaluation with the guidance of the classifier, it is removed from the list of cheap points and appended to the list of expensive points that are used to train the classifier. Therefore, it is impossible for previously sampled points to be sampled and evaluated with the objective function more than once.

In contrast to the CGS method, repeat evaluations are a common occurrence with the traditional implementation of the GA, which is used in this research. For the 50 trials that are performed on the three test problems in this study, there is an *average* of 394.3 repeats during the first 1,200 function evaluations for the knapsack problem, 3,379 repeats during the first 8,000 function evaluations for the TSP, and 802.5 repeats during the first 2,000 function evaluations for the welded beam problem. The cause of many of these repeat evaluations is that, in several of the trials, the GA's population has become largely composed of identical solutions. When this occurs, mating of two identical parent solutions results in two identical children solutions that are also identical to the parents. Therefore, subsequent generations are very similar to the previous ones, and solutions that have already been evaluated are reevaluated. The homogeneity of the candidate solution pool that results often persists for generations unless a well-placed mutation operation introduces an improvement to the current best solution. As a result, the GA expends a

significant amount of computational resources as it evaluates and reevaluates solutions, introducing minor mutations, in an attempt to reintroduce diversity into the population.

The Bayesian classifier gives the CGS method the ability to simultaneously exploit high performance solutions (depth search) and explore regions of the design space that are uncertain (breadth search), while avoiding points that have a high probability of resulting in low performance. By continuously training and updating the classifier before the start of each new iteration, the CGS method is able to use knowledge gained from all of the previously evaluated candidate points to inform the search for improved solutions. In contrast, GAs rely only on the solutions contained in the most recent generation to inform the search process using a "survival of the fittest" strategy, in which only solutions that are already known to be high performers have the highest chance of being included in subsequent generations. Therefore, the GA has no memory of previously evaluated solutions other than a small number of "elites" that are guaranteed to survive from the previous generation.

In this research, training points are sampled randomly and no effort is made to develop a training set that provides the classifier with a variety of solution classes. In practice, however, every attempt should be made to provide the classifier with a training set that contains solutions from all qualitative classes. Even a very large training set will not result in a useful classifier if high quality solutions are absent from the set. Therefore, when generating the initial training set, practitioners should take advantage of any information or intuitions that are available about the problem to be solved. Previous design experience could be used to seed the initial training set with solutions that are known to result in favorable or unfavorable objective function values, thus giving the classifier an early indicator of the characteristics of high quality solutions.

6 Computational Time and Complexity of the Classifier-Guided Sampling Method

The CGS method identifies high-quality solutions with fewer objective function evaluations than genetic algorithms when applied to the three test problems in the previous section. When the objective function is expensive to evaluate, the time savings of the CGS method are considerable. However, there is additional computational expense associated with the process of training the classifier, classifying the unevaluated test points, and sampling for subsequent expensive evaluation. In most cases, these operations require more computational time than the GA selection, crossover, and mutation steps. In this section, the time complexity of the CGS method is studied to gain an understanding of when it may and may not be worthwhile, from the perspective of computational time, to use the CGS method. If the objective function is inexpensive to evaluate, it may not be worthwhile to use the CGS algorithm, because doing so would take more time than using a faster algorithm even if the competing method requires more objective function evaluations, on average, to achieve the same result.

The knapsack problem, traveling salesperson problem (TSP), and the welded beam problem are used to test the computational expense of the CGS algorithm. For each test problem, the CGS algorithm is run with 500 initial training points (N_{tr}), 500 new points sampled at each iteration (N_s), and a total of 10,000 objective function evaluations. Three measurements are taken at each iteration: training time, classification time, and sampling time. The training time is the time it takes to train the classifier with the 'expensive' points, which are points from the current iteration that have been evaluated with the objective function. Training time does not include the time needed to evaluate the objective function. The classification time is the time required to use the classifier to classify each 'cheap' point in the set of all candidate solutions that has not yet been evaluated with the expensive objective function. Lastly, the sampling time is the time required to use the outputs from the classifier (classes and posterior probabilities) to sample points from the set of cheap points for

subsequent objective function evaluation. All experiments are performed using MATLAB on a high-performance computing Linux machine with two dual-core, 3.33GHz processors and 24 GB of physical memory.

In Figure 10, the training time versus the number of training points is shown for the three test problems. In all test cases, the training time increases linearly with the number of training points. This behavior is expected, because the classifier does not become more complex with each new training point. That is, there are no additional terms or parameters in the classifier equations, and the increase in expense should be proportional only to the number of training points. The test problem with the slowest training time is the knapsack problem, followed by the TSP, and the problem with the fastest training times is the welded beam problem. The problem-specific training time, in this case, is correlated with the number of independent variables for each respective problem (20 for the knapsack problem, 10 for the TSP, and 6 for the welded beam problem). For each independent variable, the training function must cycle through all of the training points and count the number of times each variable in the training set assumes a specific discrete value. When a variable is conditionally dependent on one or more other variables (i.e., has parent nodes in the Bayesian network), the training function must count the number of times a specific combination of variable and parent variable values appear. The connectivity of the Bayesian network has a slight, but relatively inconsequential, impact on the time complexity of this counting process because the values for a particular variable and its parents can be counted simultaneously in a single step.

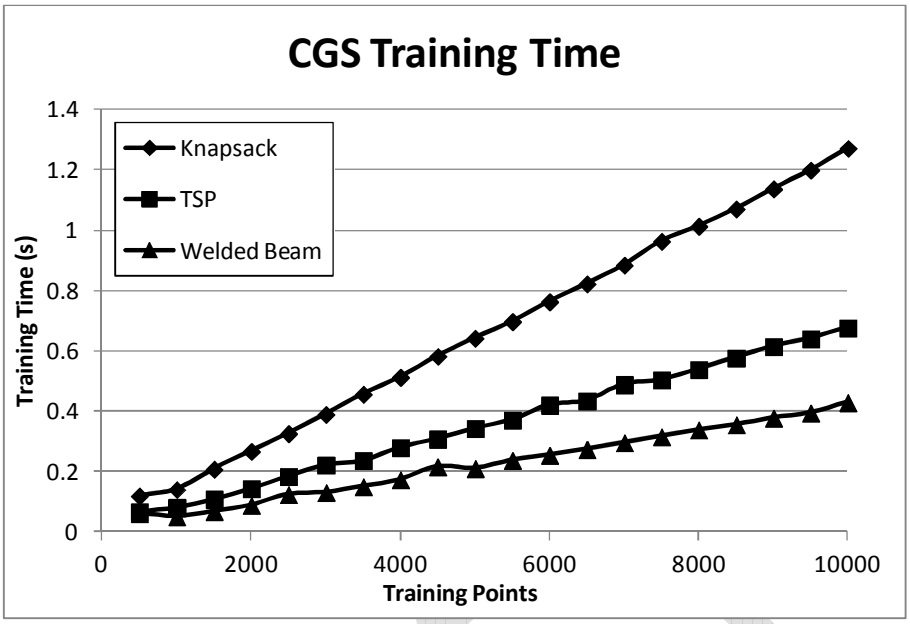

Figure 10: Training time versus number of training points

The number of training points has an insignificant effect on the classification and sampling times. However, the number of cheap points has a significant effect on these metrics. The number of cheap points changes very slightly from one iteration to the next, because it decreases by an amount equal to the number of points sampled for expensive evaluation, and this number is very small compared to the total number of iterations. Therefore, it is not very useful or informative to plot the classification and sampling times versus the number of cheap points. Instead, the average classification and sampling times of the 20 iterations performed in this time complexity study versus the average numbers of cheap points at each iteration are calculated and plotted, as shown in Figure 11. In general, the classification and sampling times increase linearly with the number of cheap points. This trend is expected, because the amount of work performed by the classifier and the sampling step of the CGS algorithm is directly proportional to the number of cheap points that need to be classified and considered for sampling.

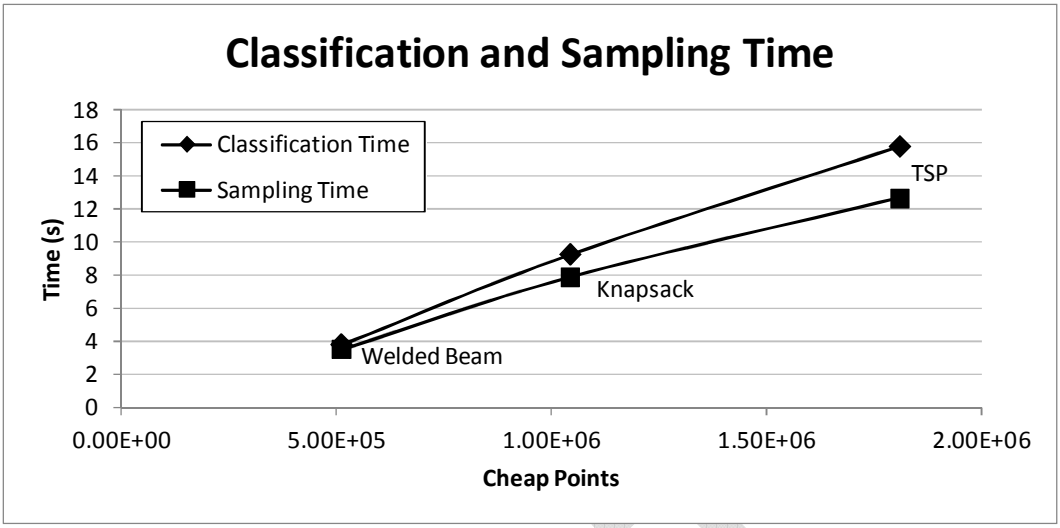

Figure 11: Classification time and sampling time versus number of classified points

In Figure 12, the total CGS computational time is decomposed into its constituents. Most of the computational expense in the CGS algorithm is associated with the cheap point classification and sampling steps. Training time is a very small percentage of the total time required by the CGS algorithm, while the classification time and sampling time are approximately equal and make up the remaining percentages. Classification and sampling times are large due to the large number of cheap points that must be handled in these steps. For the classification time, all of the cheap points must be evaluated with the classifier, and this takes a significant amount of time when there are a large number of these points. The sampling step is also somewhat computationally intensive, because all of the cheap points must be checked to determine if they qualify as high-certainty high-class points, high-certainty low-class points, or uncertain points.

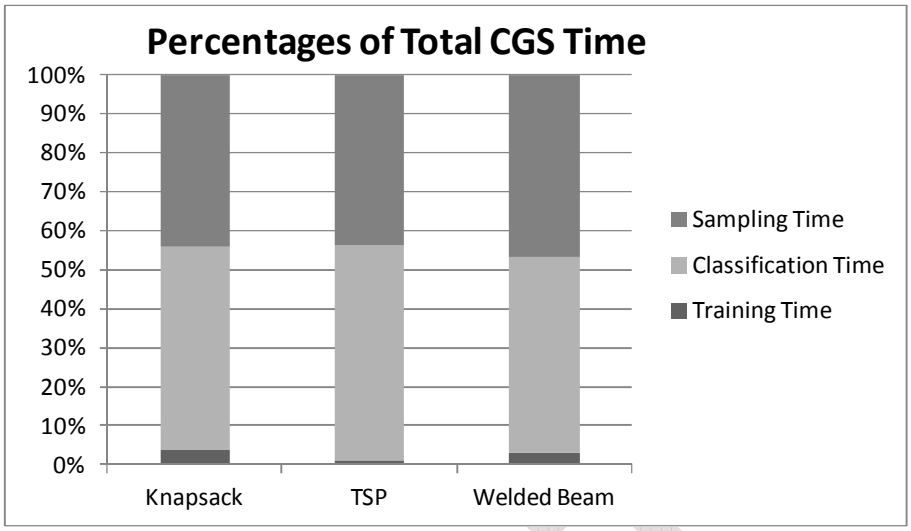

Figure 12: Percentages of total CGS time

The accumulated computational expense of the CGS algorithm over all iterations in a single run is highly dependent on the user defined parameter N_s, which is the number of points that are sampled for objective function evaluation at each iteration. When this parameter is small, more CGS iterations can be performed for a fixed number of objective function evaluations. Accumulated computational time of the CGS algorithm increases in this case, because the large set of cheap points must be classified and sampled at each iteration. On the other hand, choosing larger N_s values may have adverse effects on the rate of convergence towards optimal solutions. During the first few CGS iterations, the number of training points in the classifier is small, and the classifier may not be providing accurate predictions of categorical solution quality. If N_s is set to a large value, a large number of points are sampled based on the outputs of an inaccurate classifier. Larger N_s values may result in an inefficient search of the design space, and the total number of function evaluations may be increased. Therefore, smaller values of N_s are generally preferred, especially when the objective function is expensive to evaluate.

The CGS algorithm is more computationally expensive than a typical GA, because a GA's selection, crossover, and mutation operations involve simple manipulations of strings of integers. However, the CGS method converges towards the known global optima of the test problems significantly faster than GAs. If the

objective function is very fast and inexpensive, it may require less total time to use a GA even if more objective function evaluations are required. On the other hand, if the objective function is expensive to operate, the benefit of rapid convergence offered by the CGS method outweighs the drawback of the additional computational expense of the CGS algorithm. For a given problem, there is a specific level of objective function computational expense above which it is worthwhile to use the CGS method over GAs.

Estimations of these values for the three test problems, herein referred to as the "Expensive Evaluation Payoff Times," are provided next. For each test problem, the CGS and GA processes are terminated after a predetermined number of objective function evaluations. On average, the GA does not converge to the known global optima before the maximum number of evaluations are performed. The best objective function value found with the GA, on average, is designated as a "desired" objective function value. Next, the number of function evaluations needed by the CGS method to achieve these desired values is identified. This information is used in conjunction with the results of this time complexity study to estimate the expensive evaluation payoff time for each of these test problems. The number of training points and the number of points sampled at each iteration are the same as those for the rate of convergence tests in Section 4. The results of this process are presented in Table 4.

Table 4: Expensive evaluation payoff time of the CGS method

Test Problem	*GA Objective Function Evaluations*	*Average Best GA Objective Function Value*	*Evaluations Needed by CGS to Find Equivalent Objective Function Value*	*Expensive Evaluation Payoff Time (s)*
Knapsack	1200	788.34	350	0.10
TSP	8000	344.81	1900	0.08
Welded Beam	2000	$2.08	450	0.02

For all three test problems, the expensive evaluation payoff time of the objective function is very small (less than or equal to one tenth of a second). That is, if the time required for each objective function evaluation is greater than the expensive evaluation payoff times in Table 4, then it takes the CGS algorithm less computational time than the GA to identify a solution with an equivalent objective function value. In other words, even though the CGS algorithm is more expensive to

run than a GA, the time saved by identifying better solutions with fewer objective function evaluations is significant enough to warrant its use. In this analysis, it is assumed that the GA can be run in zero time. When the actual time of the GA is accounted for, the expensive evaluation payoff time would be even smaller.

7 Conclusions and Future Work

The CGS method was presented for performing design optimization and design space exploration on computationally expensive, discrete variable, discontinuous response problems. The CGS method uses a Bayesian classifier to provide estimates of system performance. The classifier assigns each candidate solution a categorical class label, and these class labels are used to guide the search process towards combinations of design variables that have high probabilities of yielding performance improvements. The method is compared to GAs and random search using three discrete variable test problems. When compared to GAs, the CGS method converges to known global optima with significantly fewer function evaluations. Furthermore, the GGS method is significantly more robust than the GA in all test cases. The CGS method consistently identifies the known global optima of the test functions across multiple trial applications for each test problem, whereas the GA is shown to be prone to fixation on local, suboptimal solutions.

As an optimization technique, the CGS method is very user-friendly. There are no problem-specific encodings of solutions to master, and constraints are handled by simply assigning infeasible solutions a "low quality" class label. Other user-defined parameters, such as the number of expensive points to sample on each iteration and the initial proportion of "high quality" solutions to sample, have an effect on how quickly the CGS method converges on an optimal or near optimal solution. However, if these parameters are not set perfectly, the CGS method still performs well and converges to global optima in most cases.

The CGS method presented here is applied to problems with relatively small design spaces that range from approximately 500,000 possible solutions for the welded beam problem to approximately 1.8 million possible solutions for the 11-city traveling salesperson problem. For each iteration of the CGS algorithm, the classifier assigns categorical class labels to *all* possible solutions when the problems are this size. However, many engineering problems have design spaces that are so large that classification of the entire space is prohibitively time consuming. This issue could be addressed in future work by direct sampling of the probability distributions on which the classifier is based, thus eliminating the need to classify the entire unevaluated design space.

Choosing an appropriate Bayesian network structure for the classifier is also critical. In this research, the fully independent, or Naïve Bayes, structure is used for all problems except for the traveling salesperson problem. When a Naïve Bayes structure is used, the dimensionalities of each variable's class conditional probabilities are low, and fewer training points are required to populate the distribution. For many problems, a more accurate classifier will result if a more sophisticated Bayesian network classifier is used. A careful understanding of the problem and the conditional dependencies of the variables involved could be used to manually construct a more appropriate Bayesian network structure, or network building algorithms could be used to automatically construct the Bayesian network and consequentially improve the ability of the classifier to predict solution classes accurately.

Lastly, the CGS method has strong potential to be applied to multi-objective optimization problems. This capability could be realized by following the traditional approach of aggregating the objectives into a single objective in the form of a weighted sum. Alternatively, a novel and possibly more efficient approach could be to utilize the classifier outputs to sample only the candidate solutions that are believed to be Pareto optimal.

Acknowledgements

Support from the Office of Naval Research under the auspices of the Electric Ship Research and Development Consortium is gratefully acknowledged. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Huang, M. W. and J. S. Arora, 1997, "Optimal Design with Discrete Variables: Some Numerical Examples," *International Journal for Numerical Methods in Engineering,* Vol. 40, pp. 165-188.

[2] Backlund, P. B., 2012, "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.

[3] Wang, G. G. and S. Shan, 2007, "Review of Metamodeling Techniques in Support of Engineering Design Optimization," *Transactions of the ASME,* Vol. 129, pp. 370-380.

[4] Simpson, T. W., V. Toropov, V. Balabanov and F. A. C. Viana, 2008, "Design and Analysis of Computer Experiments in Multidisciplinary Design Optimization: A Review of How Far We Have Come – or Not," *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Victoria, British Columbia, 10-12 September, 2008.

[5] Giunta, A. and L. T. Watson, 1998, "A Comparison of Approximation Modeling Techniques: Polynomial Versus Interpolating Models," *7th AIAA/USAF/ISSMO Symposium on Multidisciplinary Analysis & Optimization*, St. Louis, MO, AIAA, Vol. 1, pp.392-404. AIAA-98-4758.

[6] Simpson, T. W., T. M. Mauery, J. J. Korte and F. Mistree, 1998, "Comparison of Response Surface and Kriging Models for Multidisciplinary Design Optimization," *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization,* Vol. 1, pp. 381-391.

[7] Jin, R., W. Chen and T. W. Simpson, 2001, "Comparative Studies of Metamodelling Techniques Under Multiple Modelling Criteria," *Structural and Multidisciplinary Optimization,* Vol. 23, pp. 1-13.

[8] Clarke, S. M., J. H. Griebsch and T. W. Simpson, 2005, "Analysis of Support Vector Regression for Approximation of Complex Engineering Analyses," *Journal of Mechanical Design,* Vol. 127, pp. 1077-1087.

[9] Fang, H., M. Rais-Rohani, Z. Liu and M. F. Horstemeyer, 2005, "A Comparative Study of Metamodeling Methods for Multiobjective Crashworthiness Optimization," *Computers and Structures,* Vol. 83, pp. 2121-2136.

[10] Ely, G. R. and C. C. Seepersad, 2009, "A Comparative Study of Metamodeling Techniques for Predictive Process Control of Welding Applications," *International*

Manufacturing Science and Engineering Conference, West Lafayette, Indiana, ASME, Paper Number MSEC2009-84189.

[11] Kim, B.-S., Y.-B. Lee and D.-H. Choi, 2009, "Comparison Study on the Accuracy of Metamodeling Technique for Non-Convex Functions," *Journal of Mechanical Science and Technology,* Vol. 23, pp. 1175-1181.

[12] Vapnik, V., S. Golowich and A. Smola, 1997, "Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing," *Advances in Neural Information Processing Systems,* Vol. 9, pp. 281-287.

[13] Sacks, J., S. B. Schiller and W. J. Welch, 1989, "Designs for Computer Experiments," *Technometrics,* Vol. 31, No. 1, pp. 41-47.

[14] Friedman, J. H., 1991, "Multivariate Adaptive Regression Splines," *The Annals of Statistics,* Vol. 19, No. 1, pp. 1-67.

[15] Turner, C. J., R. H. Crawford and M. I. Campbell, 2007, "Global optimization of NURBs-based metamodels," *Engineering Optimization,* Vol. 39, No. 3, pp. 245-269.

[16] Jin, R., W. Chen and A. Sudjianto, 2002, "On Sequential Sampling for Global Metamodeling in Engineering Design," *ASME Design Engineering Technical Conferences*, Montreal, Canada, Paper Number DETC2002/DAC-3492.

[17] Osio, I. G. and C. H. Amon, 1996, "An Engineering Design Methodology with Multistage Bayesian Surrogates and Optimal Sampling," *Research in Engineering Design,* Vol. 8, No. 4, pp. 189-206.

[18] Rodriguez, J. F., V. M. Pérez, D. Padmanabhan and J. E. Renaud, 2001, "Sequential Approximate Optimization Using Variable Fidelity Response Surface Approximations," *Structural and Multidisciplinary Optimization,* Vol. 22, pp. 24-44.

[19] Wang, G. G., Z. Dong and P. Aitchison, 2001, "Adaptive Response Surface Method - A Global Optimization Scheme for Computation-intensive Design Problems," *Engineering Optimization,* Vol. 33, No. 6, pp. 707-734.

[20] Wang, L., S. Shan and G. G. Wang, 2004, "Mode-Pursuing Sampling Method for Global Optimization on Expensive Black-Box Functions," *Engineering Optimization,* Vol. 36, No. 4, pp. 419-438.

[21] Shan, S. and G. G. Wang, 2005, "An Efficient Pareto Set Identification Approach for Multi-objective Optimization on Black-box Functions," *Journal of Mechanical Design,* Vol. 127, pp. 866-874.

[22] Forrester, A. I. J. and A. J. Keane, 2009, "Recent Advances in Surrogate-Based Optimization," *Progress in Aerospace Sciences,* Vol. 45, pp. 50-79.

[23] Meckesheimer, M., R. R. Barton, T. W. Simpson, F. Limayem and B. Yannou, 2001, "Metamodeling of Combined Discrete/Continuous Responses," *AIAA Journal,* Vol. 39, No. 10, pp. 1950-1959.

[24] Sharif, B., G. G. Wang and T. Y. ElMekkawy, 2008, "Mode Pursuing Sampling Method for Discrete Variable Optimization on Expensive Black-Box Functions," *Journal of Mechanical Design,* Vol. 130, pp. 021402-1-11.

[25] Aarts, E. H. L. and J. H. M. Korst, 1989, *Simulated Annealing and Boltzmann Machines,* John Wiley and Sons.

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

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

[28] Li, M., G. Li and S. Azarm, 2008, "A Kriging Metamodel Assisted Multi-objective Genetic Algorithm for Design Optimization," *Journal of Mechanical Design,* Vol. 130, No. 3, pp. 031401 (10 pages).

[29] Pelikan, M., D. E. Goldberg and F. G. Lobo, 2002, "A Survey of Optimization by Building and Using Probabilistic Models," *Computational Optimization and Applications,* Vol. 21, No. 1, pp. 5-20.

[30] Harik, G., G. G. Lobo and D. E. Goldberg, 1999, "The Compact Genetic Algorithm," *IEEE Transactions on Evolutionary Computation,* Vol. 3, No. 4, pp. 287-297.

[31] Pelikan, M., D. E. Goldberg and E. Cantu-Paz, 2000, "Linkage Problem, Distribution Estimation, and Bayesian Networks," *Evolutionary Computation,* Vol. 8, No. 3, pp. 311-340.

[32] Kotsiantis, S. B., 2007, "Supervised Machine Learning: A Review of Classification Techniques," *Informatica,* Vol. 31, pp. 249-268.

[33] Murthy, S. K., 1998, "Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey," *Data Mining and Knowledge Discovery,* Vol. 2, pp. 345-389.

[34] Furnkranz, J., 1999, "Separate-and-Conquer Rule Learning," *Artificial Intelligence Review,* Vol. 13, pp. 3-54.

[35] Zhang, G. P., 2000, "Neural Networks for Classification: A Survey," *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews,* Vol. 30, No. 4, pp. 451-462.

[36] Friedman, N., D. Geiger and M. Goldszmidt, 1997, "Bayesian Network Classifiers," *Machine Learning,* Vol. 29, pp. 131-163.

[37] Burges, C. J. C., 1998, "A Tutorial On Support Vector Machines for Pattern Recognitions," *Data Mining and Knowledge Discovery,* Vol. 2, pp. 121-167.

[38] Pearl, J., 1988, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference,* Morgan Kauffman Publishers, Inc., San Francisco.

[39] Duda, R. O., P. E. Hart and D. G. Stork, 2001, *Pattern Classification,* John Wiley and Sons, New York.

[40] Zabell, S. L., 1989, "The Rule of Succession," *Erkenntnis,* Vol. 31, No. 2-3, pp. 283-321.

[41] Rish, I., 2001, "An Empirical Study of the Naive Bayes Classifier," *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*.

[42] Zhang, H., 2004, "The Optimality of Naive Bayes," *Proceeding of the Seventeenth International Florida Artificial Intelligence Research Society Conference*, AIAA Press, Maimi Beach.

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

[44] Deb, K. and M. Goyal, 1997, "Optimizing Engineering Designs Using a Combined Genetic Search," *Proceedings of the Seventh International Conference on Genetic Algorithms*, (Morgan Kauffman Publishers, San Francisco), pp. 521-528.

Appendix: Test Problem Parameters

20-Item Knapsack Problem Parameters

Table A.1: 20-Item knapsack problem parameters

Item	*Weight* (w_i)	*Value* (v_i)
1	94	3
2	70	41
3	90	22
4	97	30
5	54	45
6	31	99
7	82	75
8	97	76
9	1	79
10	58	77
11	96	41
12	96	98
13	87	31
14	53	28
15	62	58
16	89	32
17	68	99
18	58	48
19	81	20
20	83	3

Traveling Salesperson Problem Parameters

Table A.2: 11-City TSP city coordinates

City	*Abscissa*	*Ordinate*
Origin	14.8	42.7
1	98.6	29.8
2	5.0	4.5
3	39.9	12.0
4	28.0	22.6
5	57.0	79.2
6	26.2	90.7
7	67.9	55.1
8	76.0	47.1
9	86.4	67.5
10	47.3	98.5

Welded Beam Design Problem Material Properties and Costs

Table A.3: Welded beam problem material properties and costs [44]

Material	*S* (*kpsi*)	*E* (*Mpsi*)	*G* (*Mpsi*)	c_1 (*$/in.*^{3})	c_2 (*$/in.*^{3})
Steel	30	30	12	0.1047	0.0481
Cast Iron	8	14	6	0.0489	0.0224
Aluminum	5	10	4	0.5235	0.2405
Brass	8	16	6	0.5584	0.2566

In Table A.3, *S* is the design stress, *E* is the Young's modulus, and *G* is the shear modulus.