Missouri Estimation of Distribution Algorithms Laboratory

A Survey of Estimation of Distribution Algorithms

Mark Hauschild and Martin Pelikan

MEDAL Report No. 2011004

March 2011

Abstract

Estimation of distribution algorithms (EDAs) are stochastic optimization techniques that explore the space of potential solutions by building and sampling explicit probabilistic models of promising candidate solutions. This explicit use of probablistic models in optimization offers some significant advantages over other types of meta-heuristics. This paper discusses these advantages and outlines many of the different types of EDAs. In addition, some of the most powerful efficiency enhancement techniques applied to EDAs are discussed.

Keywords

Estimation of distribution algorithms, model structure, model building, evolutionary computation.

Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Department of Mathematics and Computer Science
University of Missouri–St. Louis
One University Blvd., St. Louis, MO 63121
E-mail: medal@cs.umsl.edu
WWW: http://medal.cs.umsl.edu/

A Survey of Estimation of Distribution Algorithms

Mark Hauschild
Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Dept. of Math and Computer Science, 320 CCB
University of Missouri at St. Louis
One University Blvd., St. Louis, MO 63121
mwh308@umsl.edu

Martin Pelikan
Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Dept. of Math and Computer Science, 320 CCB
University of Missouri at St. Louis
One University Blvd., St. Louis, MO 63121
pelikan@cs.umsl.edu

Abstract

Estimation of distribution algorithms (EDAs) are stochastic optimization techniques that explore the space of potential solutions by building and sampling explicit probabilistic models of promising candidate solutions. This explicit use of probablistic models in optimization offers some significant advantages over other types of metaheuristics. This paper discusses these advantages and outlines many of the different types of EDAs. In addition, some of the most powerful efficiency enhancement techniques applied to EDAs are discussed.

Keywords: Estimation of distribution algorithms, model structure, model building, evolutionary computation.

1 Introduction

Estimation of distribution algorithmss (Baluja, 1994; Mühlenbein & Paaß, 1996a; Larrañaga & Lozano, 2002; Pelikan, Goldberg, & Lobo, 2002) are stochastic optimization techniques that explore the space of potential solutions by building and sampling explicit probabilistic models of promising candidate solutions. This model-based approach to optimization has allowed EDAs to solve many large and complex problems. EDAs were successfully applied to optimization of large spin glass instances in two-dimensional and three-dimensional lattices (Pelikan & Hartmann, 2006), military antenna design (Yu, Santarelli, & Goldberg, 2006), multiobjective knapsack (Shah & Reed, 2010), groundwater remediation design (Arst, Minsker, & Goldberg, 2002; Hayes & Minsker, 2005), amino-acid alphabet reduction for protein structure prediction (Bacardit, Stout, Hirst, Sastry, Llorà, & Krasnogor, 2007), identification of clusters of genes with similar expression profiles (Peña, Lozano, & Larrañaga, 2004), economic dispatch (Chen & p. Chen, 2007), forest management (Ducheyne, De Baets, & De Wulf, 2004), portfolio management (Lipinski, 2007), cancer chemotherapy optimization (Petrovski, Shakya, & Mccall, 2006), environmental monitoring network design (Kollat, Reed, & Kasprzyk, 2008), and others. It is important to stress that in most of these applications no other technique was shown to be capable of achieving better performance than EDAs or

solving problems of comparable size and complexity. This paper will review the basic principle of EDAs, and point out some of the features of EDAs that distinguish these methods from other metaheuristics and allow them to achieve these impressive results in such a broad array of problem domains.

The paper is organized as follows. Section 2 describes the basic EDA procedure. Section 3 gives a broad overview of many example EDAs, divided into four broad categories. Section 4 covers some of the advantages of EDAs over other metaheuristics. Section 5 discusses the most common efficiency enhancements that may be incorporated into EDAs to speed up their operation. Lastly, section 6 summarizes and concludes the paper.

2 Estimation of Distribution Algorithms

Suppose a researcher was presented with a large number of possible solutions to a problem and wished to generate new and (hopefully) better solutions. One way that he or she might approach this problem is to attempt to determine the probability distribution that would give higher probabilities to solutions in the regions with the best solutions available. Once this was completed, one could sample this distribution to find new candidate solutions to the problem. Ideally, the repeated refinement of the probabilistic model based on representative samples of high quality solutions would keep increasing the probability of generating the global optimum and, after a reasonable number of iterations, the procedure would locate the global optimum or its accurate approximation. In the rest of this section we discuss how EDAs do this automatically.

2.1 General EDA Procedure

Estimation of distribution algorithms (EDAs) (Baluja, 1994; Mühlenbein & Paaß, 1996a; Larrañaga & Lozano, 2002; Pelikan, Goldberg, & Lobo, 2002) are stochastic optimization algorithms that explore the space of candidate solutions by sampling an explicit probabilistic model constructed from promising solutions found so far. EDAs typically work with a population of candidate solutions to the problem, starting with the population generated according to the uniform distribution over all admissible solutions. The population is then scored using a *fitness function*. This fitness function gives a numerical ranking for each string, with the higher the number the better the string. From this ranked population, a subset of the most promising solutions are selected by the *selection* operator. An example selection operator is truncation selection with threshold $\tau = 50\%$, which selects the 50% best solutions. The algorithm then constructs a *probabilistic model* which attempts to estimate the probability distribution of the selected solutions. Once the model is constructed, new solutions are generated by sampling the distribution encoded by this model. These new solutions are then incorporated back into the old population (or replacing it entirely). The process is repeated until some termination criteria is met (usually when a solution of sufficient quality is reached or when the number of iterations reaches some threshold), with each iteration of this procedure usually referred to as one *generation* of the EDA. The basic EDA procedure is outlined in Algorithm 1.

The important step that differentiates EDAs from many other metaheuristics is the construction of the model that attempts to capture the probability distribution of the promising solutions. This is not a trivial task as the goal is not to perfectly represent the population of promising solutions, but instead to represent a more general distribution that captures the features of the selected solutions that make these solutions better than other candidate solutions. In addition, we have to ensure that the model can be built and sampled in an efficient manner.

Algorithm 1 EDA pseudocode

```
g ← 0
generate initial population P(0)
while (not done) do
  select population of promising solutions S(g) from P(g)
  build probabilistic model M(g) from S(g)
  sample M(g) to generate new candidate solutions O(g)
  incorporate O(g) into P(g)
  g ← g + 1
end while
```

2.2 Solving Onemax with a Simple EDA

Let us illustrate the basic EDA procedure with an example of a simple EDA solving the onemax problem. In onemax, candidate solutions are represented by vectors of n binary variables and the fitness is computed by

$$onemax(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} X_i \ , \tag{1}$$

where n is the number of variables and X_i is the i^{th} variable in the problem (i^{th} position in the input binary string). This function has one global optimum in the string of all 1s.

In this example our population size is set to $N = 6$, with $n = 5$ binary variables per solution. Truncation selection with threshold $\tau = 50\%$ is used to select the subset of the most promising solutions (the 50% best solutions are selected). To estimate the probability distribution of these promising solutions, a *probability vector* is used that stores the probability of a 1 in each position of the solution strings. The probability vector provides a fast and efficient model for solving the onemax problem and many other optimization problems, mainly due to the fact that it is based on the assumption that all problem variables are independent. To learn a probability vector, the probability p_i of a 1 in each position i is set to the proportion of selected solutions containing a 1 in this position. To generate a new binary string from the probability vector, for each position i, a 1 is generated in this position with probability p_i. For example, if $p_3 = 0.6$, we generate a 1 in the third position of a new candidate solution with the probability of 60%. In each generation (iteration of the algorithm), $N = 6$ candidate solutions are generated from the current model to create a new population of size $N = 6$. The simulation is outlined in Figure 1.

It is clear from the first generation that the procedure is having positive effects. The offspring population already contains significantly more 1s than the original population and also includes several copies of the global optimum 11111. In addition, the probability of a 1 in any particular position has increased; consequently, the probability of generating the global optimum has increased. The second generation leads to a probability vector that is even more strongly biased towards the global optimum and if the simulation was continued for one more generation, the probability vector would generate only the global optimum.

Even though the previous example was rather small, this procedure works on larger problems. To show this in practice, the probability of ones in each generation from an example run of an EDA is shown in Figure 2. In this case $n = 50$ and the population size is $N = 100$. We see that the proportions of 1s in different positions increase over time. While the probabilities of 1s in some positions do fluctuate in value in the initial iterations, eventually all the probability vector entries

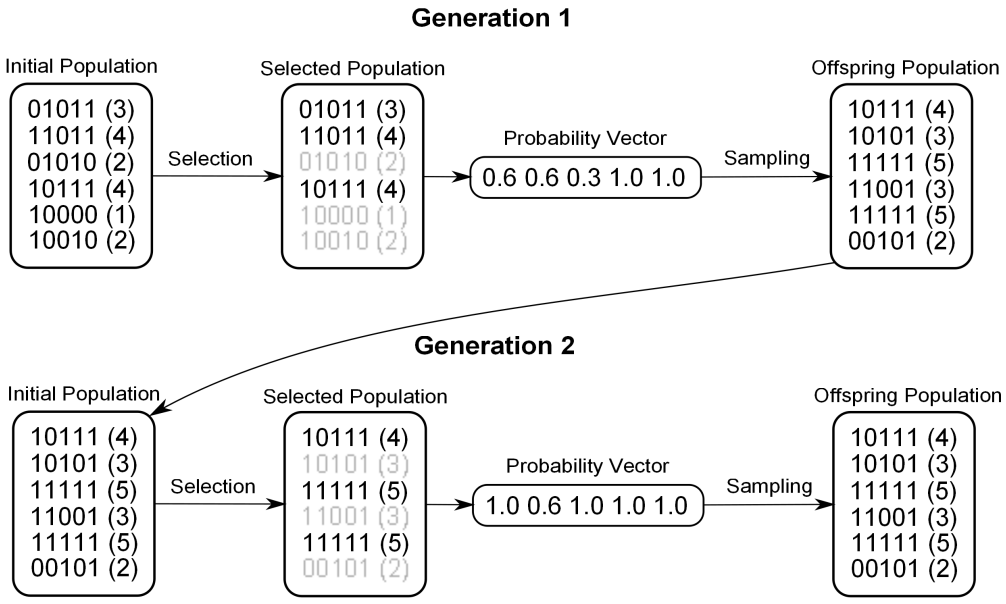


Figure 1: Two generations of a simple EDA using a probability vector to solve one-max.

become 1.

Assuming that the population size is large enough to ensure reliable convergence (Harik, Cantú-Paz, Goldberg, & Miller, 1997; Goldberg, 2002), the EDA based on the probability vector model provides an efficient and reliable approach to solving onemax and many other optimization problems. Nonetheless, is it always the case that the probability vector is sufficient for solving the problem?

2.3 Linkage Learning EDAs: Using an EDA to Solve Trap-5

To illustrate some of the limitations of EDAs based on the probability vector, let us consider a more complex problem such as the concatenated trap of order 5 (trap-5) (Ackley, 1987; Deb & Goldberg, 1991). In trap-5, the input string is first partitioned into independent groups of 5 bits each. The contribution of each group of 5 bits (trap partition) is computed as

$$trap_5(u) = \begin{cases} 5 & \text{if } u = 5 \\ 4 - u & \text{otherwise} \end{cases}, \tag{2}$$

where u is the number of 1s in the input string of 5 bits. While the trap-5 function has only one global optimum, the string of all 1s, it also has $(2^{n/5} - 1)$ other local optima, namely those strings where all bits in at least one trap partition are 0 and all bits in each of the remaining partitions are either 0 or 1. Trap-5 necessitates that all bits in each group be treated together, because statistics of lower order are misleading (Thierens & Goldberg, 1993); that is why trap-5 provides an excellent example to illustrate the limitations of the probability vector as a model.

The simple EDA shown in the previous simulation was run on trap-5. The first few generations are shown in Figure 3. Almost immediately we see a problem: The fitness function ranks solutions with many 0s higher than solutions with many 1s. By emphasizing solutions with many 0s, the

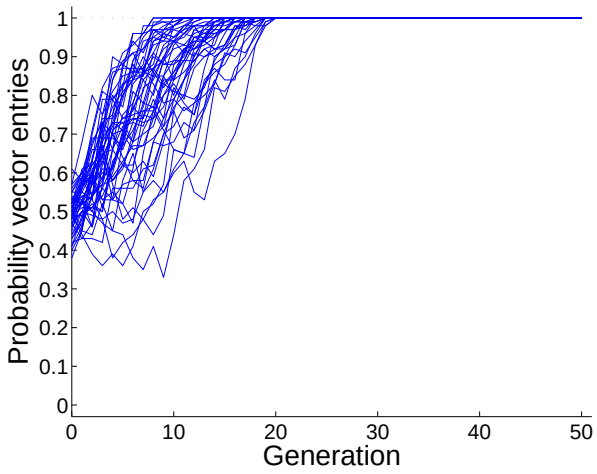


Figure 2: Proportions of 1s in a probability vector of a simple EDA on the onemax problem of $n = 50$ using a population of $N = 100$ candidate solutions.

probability vector entries get lower over the two generations. In other words, in each generation, while our average population fitness is increasing, the strings are actually getting farther and farther away from the global optimum.

To see if the same behavior can be observed on a larger problem, an EDA with a probability vector was used to solve a trap-5 problem of 50 bits with a population of size $N = 100$. Figure 4a shows the proportions of 1s in each position of the solution strings. This example confirms that the proportions of 1s in different positions of solution strings decrease over time. Some entries increase for several iterations but the bias towards 0s at any individual position is too strong to be overcome. Indeed, by generation 27 the entire population has converged to all 0s. One may hypothesize that the situation would change if our population was larger but, unfortunately, larger populations would only make the continuous decrease in the proportion of 0s more stable.

To understand the reason for the failure of the EDA based on the probability vector on trap-5, let us return to the onemax example. For onemax, the average fitness of candidate solutions with a 1 in any position is better than the average fitness of solutions with a 0 in that position. The selection is thus expected to give preference to 1s and the learning and sampling of the probability vector will ensure that these increased probabilities of 1s are reflected in the new populations of candidate solutions. However this situation is reversed for trap-5, for which the average fitness of solutions with a 0 in any position is greater than the average fitness of solutions with a 1 in that position (Deb & Goldberg, 1991), even though the optimum is still located in the string consisting of all 1s. This leads to the probability vector being strongly biased towards solutions with 0s in all positions.

All is not lost, however. What if we can change the model to respect the linkages between the bits in the same trap partition? If it was possible for the algorithm to learn the structure of trap-5, it could then treat all the bits in the same trap partition as a single variable. That is, the model would store the probability of each combination of 5 bits in any particular trap partition, and new solutions would be sampled by generating 5 bits at a time according to these probabilities. Since from the definition of trap-5 the average fitness of candidate solutions with all bits in a trap partition set to 1 is expected to be higher than the average fitness of solutions with one or more 0s in that partition, we would expect the proportions of trap partitions with all 1s to increase over time. By merging the variables in each trap partition together, the extended compact genetic algorithm

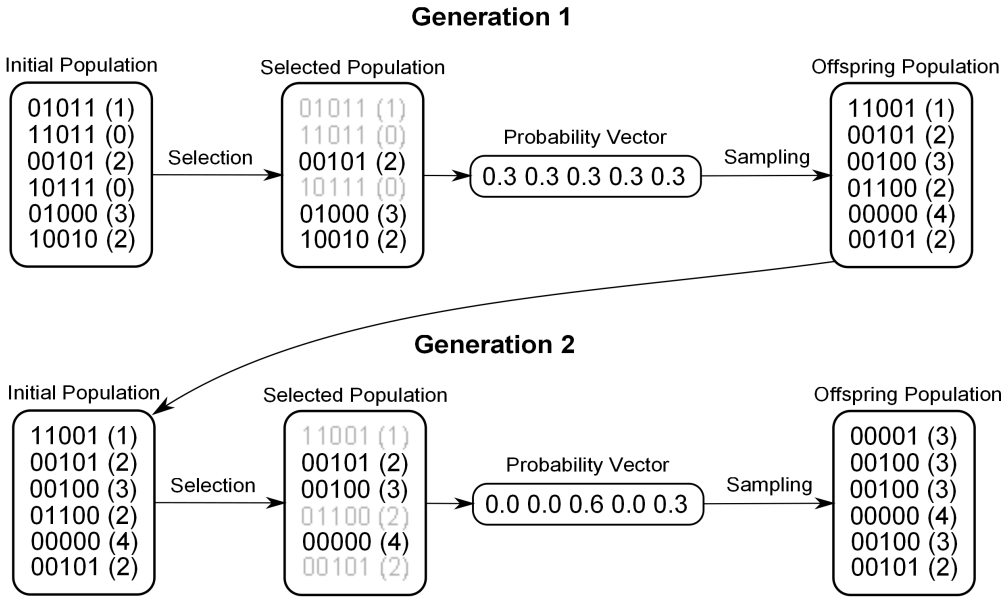


Figure 3: Two generations of a simple EDA using a probability vector to solve trap-5.

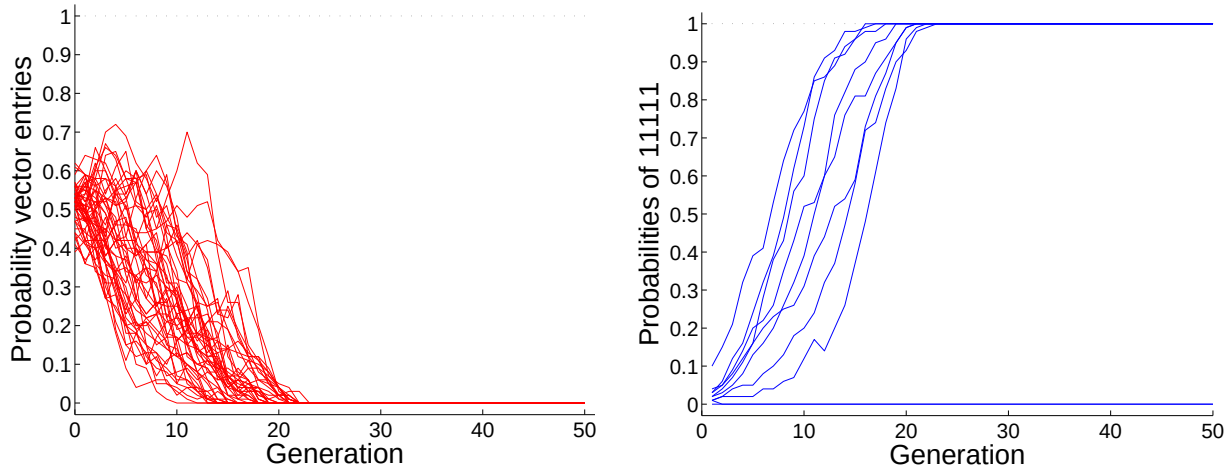
(ECGA) (Harik, 1999) explained further in section 3.1.3 does just that. Probabilistic models that combine groups of variables or bits into linkage groups and assume independence between the different linkage groups are often referred to as marginal product models (Harik, 1999).

To show the difference that the marginal product model for trap-5 can make in practice, an EDA that uses this model was applied to solve a 50-bit trap-5 problem. Figure 4b shows the proportions of 11111s in different trap partitions in the population. These results show that with an appropriate marginal product model the EDA performs similarly as it did on onemax, with the entire population converging to the global optimum in a little over 20 generations. This example illustrates that, in terms of probability distributions, the main reason for the failure of the probability vector based EDA on trap-5 is the assumption that the problem variables are independent.

These examples make it clear that the class of allowable models and the methods for learning and sampling these models are key elements in EDA design. If the model built in each generation captures the important features of selected solutions and generates new solutions with these features, then the EDA should be able to quickly converge to the optimum (Mühlenbein & Mahnig, 1999). However, as we will see later on, there is a tradeoff between the expressiveness of probabilistic models and the complexity of learning and sampling these models. Due to the importance of the class of models used in an EDA, the type of probability models used in an EDA is often how one EDA is differentiated from another.

3 EDA Overview

Because of the key impact that the probabilistic models used have on EDA efficiency and applicability, EDAs are usually categorized by the types of distributions they are able to encode. This section covers four broad categories of EDAs. Note that this is not an exhaustive survey and only a few representative algorithms are discussed for each category. For further information, please see



(a) EDA based on the probability vector

(b) EDA based on a marginal product model with linkage groups corresponding to trap partitions.

Figure 4: Statistics from two runs of an EDA on trap-5 of $n = 50$ bits using probabilistic models of different structure.

many of the individual papers for other examples.

In this section we assume the general structure of an EDA as outlined in Algorithm 1. Rather than go over all the details of every algorithm, instead we will point out what distinguishes a particular algorithm from the others. Since in most cases the primary differences between EDAs are found in the class of probabilistic models used and the methods used for learning and sampling these models, in the majority of this section we focus on these EDA components and omit less important technical details.

Section 3.1 covers EDAs that can be used for problems using discrete variables. Section 3.2 discusses EDAs for solving problems where candidate solutions are represented by permutations. Section 3.3 describes EDAs that address problems where candidate solutions are represented by real-valued vectors. Lastly, section 3.4 covers EDAs for problems in genetic programming.

3.1 Discrete Variables

This section covers the first broad category of EDAs, those that work on fixed-length strings of a finite cardinality (usually binary). We start by describing EDAs that ignore all interactions between variables and end with algorithms that are able to capture a broad variety of possible interactions.

3.1.1 Univariate Models

One of the simplest approaches is to assume that the problem variables are independent. Under this assumption, the probability distribution of any individual variable should not depend on the values of any other variables. EDAs of this type are usually called *univariate* EDAs. Figure 5a shows an illustration of this type of model.

Mathematically, a univariate model decomposes the probability of a candidate solution $(X_1, X_2, ..., X_n)$ into the product of probabilities of individual variables as

$$p(X_1, X_2, ..., X_n) = p(X_1)p(X_2), ..., p(X_n)$$

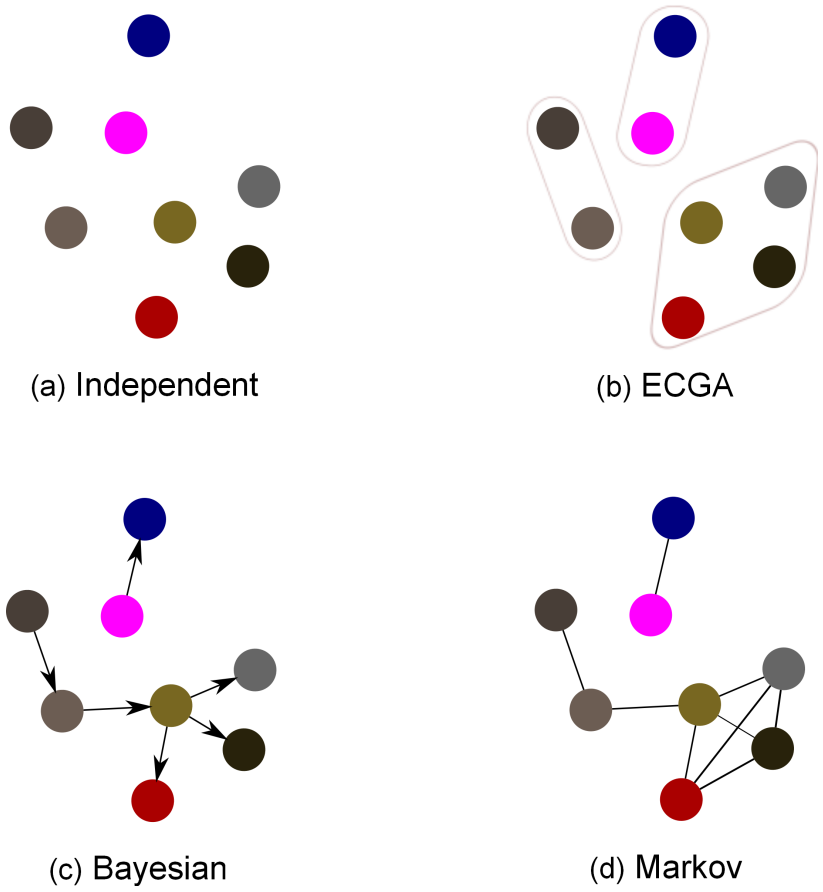


Figure 5: Examples of graphical models produced by different EDAs.

where $p(X_i)$ is the probability of variable X_i, and $p(X_1, X_2, ..., X_n)$ is the probability of the candidate solution $(X_1, X_2, ..., X_n)$. The univariate model for n variables thus consists of n probability tables and each of these tables defines probabilities of different values of the corresponding variable. Since the probabilities of different values of a variable must sum to 1, one of the probabilities may be omitted for each variable. The probability vector discussed earlier in section 2.2 is an example univariate model applicable to candidate solutions represented by fixed-length binary strings.

One example of a univariate EDA is the univariate marginal distribution algorithm (UMDA) (Mühlenbein & Paaß, 1996b), which is the algorithm we used to solve the onemax problem in section 2. UMDA works on binary strings and uses the probability vector $p = (p_1, p_2, ..., p_n)$ as the probabilistic model, where p_i denotes the probability of a 1 in position i of solution strings. To learn the probability vector, each p_i is set to the proportion of 1s in the population of selected solutions. To create new solutions, each variable is generated independently based on the entries in the probability vector. Specifically, a 1 is generated in position i with probability p_i.

Another univariate EDA is the population-based incremental learning (PBIL) (Baluja, 1994) algorithm, which works on binary strings. PBIL is an example of an *incremental* EDA, which fully replaces the population with the probabilistic model. Like UMDA, PBIL uses the probabilistic model in the form of a probability vector. The initial probability vector encodes the uniform distribution over all binary strings by setting the probabilities of 1s in each position to 0.5. In each generation of PBIL, the probability vector is sampled to generate a a small set of solutions

using the same sampling procedure as in UMDA. The best solution in this set is selected and for each variable in this solution, the corresponding entry in the probability vector is shifted towards producing that value more often by a learning rate parameter specified by the user. To prevent premature convergence, each probability vector entry is also slightly varied each generation based on a mutation rate parameter.

The compact genetic algorithm (cGA) (Harik, Lobo, & Goldberg, 1997) is another incremental univariate EDA. Much like PBIL, cGA uses a probability vector to represent the entire population of solutions encoded by fixed-length binary strings. The main difference between cGA and PBIL is in the way these EDAs update the probability vector in each generation. In each generation of the cGA, two individuals are generated and then evaluated. If at any particular position the winning solution's bit is different from the losing solution's bit, the probability vector entry is shifted by $1/N$ towards the winning bit, where N represents the theoretical population size that would be required to solve the problem for a non-incremental EDA. Note that unlike PBIL, cGA will not necessarily change all probability vector entries each iteration. One of the primary advantages of PBIL, CGA and other incremental EDAs over other EDAs is they use a much smaller memory footprint, which can be useful when trying to solve extremely large problems (Sastry, Goldberg, & Llorà, 2007); we return to this topic in section 4.

While the original versions of UMDA, PBIL and cGA assumed that candidate solutions are represented by binary strings of fixed length, it is straightforward to extend these algorithms to solve problems where candidate solutions are represented by fixed-length strings over any finite alphabet. Nonetheless, the assumption that problem variables are independent will often prevent efficient convergence to the optimum when problem variables interact strongly. The next section discusses one approach to alleviate this problem.

3.1.2 Tree-based models

The algorithms in the previous section assumed independence between problem variables. However, often the variables in a problem are related in some way. This section discusses EDAs capable of capturing some pair-wise interactions between variables by using *tree-based* models. In tree-based models, the conditional probability of a variable may only depend on at most one other variable, its parent in a tree structure.

The mutual-information-maximizing input clustering (MIMIC) (De Bonet, Isbell, & Viola, 1997) uses a chain distribution to model interactions between variables. MIMIC works by using the population of promising solutions to calculate the mutual information between all pairs of variables. Then, starting with the variable with the minimum conditional entropy, a chain dependency is added to the variable with the most mutual information. This process is repeated until all the variables are selected. The resulting tree model consists of a single chain of dependencies, with each parent having exactly one child. Given a permutation of the n variables in a problem, $\pi = i_1, i_2 \dots i_n$, MIMIC decomposes the probability distribution of $p(X_1, X_2, \dots, X_n)$ as:

$$p_\pi(X) = p(X_{i_1}|X_{i_2})p(X_{i_2}|X_{i_3}) \dots p(X_{i_{n-1}}|X_{i_n})p(X_{i_n})$$

where $p(X_{i_j}|X_{ij+1})$ denotes the conditional probability of X_{i_j} given $X_{i_{j+1}}$. New candidate solutions are then generated by sampling the probability distribution encoded by the model. The sampling proceeds by generating the variables in the reverse order with respect to the permutation π, starting with X_{i_n} and ending with X_{i_1}. After generating X_{i_n}, the values of the remaining variables are generated using the conditional probabilities with the specific value of the variable in the condition.

Baluja and Davies (1997) use dependency trees to model promising solutions, improving the expressiveness of the probabilistic models compared to the chain models of MIMIC. In dependency

trees, each parent can have multiple children. This incremental EDA works by using a probability matrix that contains all pairwise probabilities. The model is the tree that maximizes the mutual information between connected variables, which is the provably best tree model in terms of the Kullback-Leibler divergence (Chow & Liu, 1968) with respect to the true distribution. The probability matrix is initialized so that it corresponds to the uniform distribution over all candidate solutions. In each iteration of the algorithm, a tree model is built and sampled to generate several new candidate solutions. The best of these solutions are then used to update the probability matrix.

The bivariate marginal distribution algorithm (BMDA) (Pelikan & Mühlenbein, 1999) uses a model based on a set of mutually independent trees (a forest). Each generation a dependency model is created by using Pearson's chi-square statistics (Marascuilo & McSweeney, 1977) as the main measure of dependence. The model built is then sampled to generate new solutions based on the conditional probabilities learned from the population.

3.1.3 Multivariate Interactions

While the tree-based models in section 3.1.2 provide EDAs with the ability to identify and exploit interactions between problem variables, using tree models is often not enough to solve problems with multivariate or highly-overlapping interactions between variables (Bosman & Thierens, 1999; Pelikan & Mühlenbein, 1999). This section describes several EDAs that are based on probabilistic models capable of capturing multivariate interactions between problem variables.

The extended compact genetic algorithm (ECGA) (Harik, 1999) uses a model that divides the variables into independent clusters and each of these clusters is treated as a single variable. The model building starts by assuming that all the problem variables are independent. In each iteration of the model building, two clusters are merged together that improve the quality of the model the most. The quality of a model is measured by the minimum description length (MDL) metric (Mitchell, 1997). The model building terminates when no merging of two clusters improves the MDL score of the model. Once the learning of the structure of the model is complete, a probability table is computed for each cluster based on the population of selected solutions and the new solutions are generated by sampling each linkage group based on these probabilities. The model building procedure is repeated in each generation of ECGA, so the model created in each generation of ECGA may contain different clusters of variables. An example of an ECGA model is shown in Figure 5b.

Many problems contain highly overlapping subproblems that cannot be accurately modeled by dividing the problem into independent clusters. The Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 2000) uses Bayesian networks to model candidate solutions, which allow it to solve the large class of nearly decomposable problems, many of which cannot be decomposed into independent subproblems of bounded order. A Bayesian network is an acyclic directed graph with one node per variable, where an edge between nodes represents a conditional dependency. A Bayesian network with n nodes encodes a joint probability distribution of n random variables $X_1, X_2, \ldots, X_n$:

$$p(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} p(X_i|\Pi_i), \tag{3}$$

where Π_i is the set of variables from which there exists an edge into X_i (members of Π_i are called parents of X_i), and $p(X_i|\Pi_i)$ is the conditional probability of X_i given Π_i. Figure 5c shows an example Bayesian network. The difference between Bayesian networks and tree models is that in Bayesian networks, each variable may depend on more than one variable. The main difference between the marginal product models of ECGA and Bayesian networks is that Bayesian networks

are capable of capturing more complex problem decompositions in which subproblems interact.

The model building in BOA starts with a network with no edges. Edges are then added to the network one at a time, adding the edge that gives the most improvement using the Bayesian-Dirichlet (BD) metric (Heckerman, Geiger, & Chickering, 1995). Since the BD metric has a tendency to create overly complex models, usually an upper bound on the number of allowable parents or a prior bias on the network structure is set to prefer simpler models (Heckerman, Geiger, & Chickering, 1994). New candidate solutions are generated by sampling the probability distribution encoded by the built network using probabilistic logic sampling (Henrion, 1988).

The estimation of Bayesian network algorithm (EBNA) (Etxeberria & Larrañaga, 1999a) and the learning factorized distribution algorithm (LFDA) (Mühlenbein & Mahnig, 1999) also use Bayesian networks to model the promising solutions. EBNA and LFDA use the Bayesian information criterion (BIC) (Schwarz, 1978a) to evaluate Bayesian network structures in the greedy network construction algorithm. One advantage of the BIC metric over the BD metric is that it contains a strong implicit bias towards simple models and it thus does not require a limit on the number of allowable parents or any prior bias towards simpler models. However, the BD metric allows a more principled way to include prior information into problem solving, as discussed in section 5.6.

Many complex problems in the real world are hierarchical in nature (Simon, 1968). A hierarchical problem is a problem composed of subproblems, with each subproblem being a hierarchical problem itself until the bottom level is reached (Simon, 1968). On each level, the interactions within each subproblem are often of much higher magnitude than the interactions between the subproblems. Due to the rich interaction between subproblems and the lack of feedback for discriminating alternative solutions to the different subproblems, these problems cannot simply be decomposed into tractable problems on a single level. Therefore, solving these heirarchical problems presents new challenges for EDAs. First, on each level of problem solving, the hierarchical problem solver must be capable of decomposing the problem. Secondly, the problem solver must be capable of representing solutions from lower levels in a compact way so these solutions can be treated as a single variable when trying to solve the next level. Lastly, since the solution at any given level may depend on interactions at a higher level, it is necessary that alternate solutions to each subproblem be stored over time.

The hierarchical Bayesian Optimization Algorithm (hBOA) (Pelikan, 2005) is able to solve many difficult hierarchically decomposable problems by extending BOA in several key areas. In order to ensure that interactions of high order can be represented in a feasible manner, a more compact version of Bayesian networks is required. Specifically, hBOA uses Bayesian networks with local structures (Chickering, Heckerman, & Meek, 1997; Friedman & Goldszmidt, 1999) to allow feasible learning and sampling of more complex networks than would be possible with convential Bayesian networks. In addition, the preservation of alternative solutions over time is ensured by using a niching technique called restricted tournament replacement (RTR) (Harik, 1995) which encourages competition among similar solutions rather than dissimilar ones. Combined together these changes allow hBOA to solve a broad class of nearly decomposable and hierarchical problems in a robust and scalable manner (Pelikan & Goldberg, 2006).

Another type of model that is general in nature is Markov networks. The structure of Markov networks is similar to Bayesian networks except that the connections between variables are undirected. For a given decomposable function, a Markov network that ensures convergence to the global optimum may sometimes be considerably less complex than an adequate Bayesian network, at least with respect to the number of edges (Mühlenbein, 2008). Nonetheless, sampling Markov networks is more more difficult than sampling Bayesian networks. In other words, some of the

difficulty moves from learning to sampling the probabilistic model compared to EDAs based on Bayesian networks. Shakya and Santana (2008) uses Gibbs sampling to generate new solutions from its model in the Markovianity based Optimisation Algorithm (MOA). MOA was shown to have comparable performance to some Bayesian network based EDAs on deceptive test problems. An example of a Markov network model is shown in Figure 5d.

Using a more expressive class of probabilistic models allows EDAs to solve broader classes of problems. From this perspective, one should prefer tree models to univariate ones, and multivariate models to tree models. At the same time, using a more expressive class of models almost always implies that the model building and model sampling will be more computationally expensive. Nonetheless, since it is often not clear how complex a problem is before solving it and using even the most complex models described above creates only a low-order polynomial overhead even on problems that can be solved with simpler models, it is often preferable to use the more general class of models rather than the more restricted one.

The algorithms in this section are able to cover a broad variety of possible interactions between discrete variables in a problem. However, none of these algorithms is directly applicable to problems where candidate solutions are represented by permutations, which are discussed in the next section.

3.2 Permutation EDAs

In many important real-world problems, candidate solutions are represented by permutations over a given set of elements. Two important classes of such problems are the quadratic assignment problem (Koopmans & Beckmann, 1957) and the traveling salesman problem. These types of problems often contain two specific types of features or constraints that EDAs need to capture. The first is the *absolute position* of a symbol in a string and the second is the *relative ordering* of specific symbols. In some problems, such as the traveling-salesman problem, relative ordering constraints matter the most. In others, such as the quadratic assignment problem, both the relative ordering and the absolute positions matter. It certainly is possible to use non-permutation based EDAs using specific encodings to solve permutation problems. For example, one may use the random key encoding (Bean, 1994) to solve permutation-based problems using EDAs for optimization of real-valued vectors (Bosman & Thierens, 2001b; Robles, de Miguel, & Larrañaga, 2002). However, since these EDAs do not process the aforementioned types of interactions directly their performance can often be poor (Bosman & Thierens, 2001b). The following EDAs attempt to encode both of these types of features or contraints for permutation problems explicitly.

To solve problems where the solutions are permutations of a string, Bengoetxea, Larrañaga, Bloch, Perchant, and Boeres (2000) starts with a Bayesian network model built using the same approach as EBNA. However, the sampling method is changed to ensure that only valid permutations are generated. This approach was shown to have promise in solving the inexact graph matching problem. In much the same way, the dependency-tree EDA (dtEDA) of Pelikan, Tsutsui, and Kalapala (2007) starts with a dependency-tree model and modifies the sampling to ensure that only valid permutations are generated. dtEDA for permutation problems was used to solve structured quadratic assignment problems with great success (Pelikan, Tsutsui, & Kalapala, 2007). Both Bayesian networks as well as tree models are capable of encoding both the absolute position and the relative ordering constraints.

Bosman and Thierens (2001c) extended the real-valued EDA to the permutation domain by storing the dependencies between different positions in a permutation in the induced chromosome elements exchanger (ICE). ICE works by first using a real-valued EDA as discussed in section 3.3, which encodes permutations as real-valued vectors using *random keys encoding* (Bean, 1994). Ran-

dom keys are a specific type of encoding that can be used to map real-valued vectors to permutations. ICE extends the real-valued EDA by using a specialized crossover operator. By applying the crossover directly to permutations instead of simply sampling the model, relative ordering is taken into account. The resulting algorithm was shown to outperform many real-valued EDAs that use the random key encoding alone (Bosman & Thierens, 2001c).

The edge histogram based sampling algorithm (EHBSA) (Tsutsui, 2002) works by creating an edge histogram matrix (EHM). For each pair of symbols, EHM stores the probabilities that one of these symbols will follow the other one in a permutation. To generate new solutions, EHBSA starts with a randomly chosen symbol. The EHM is then sampled repeatedly to generate new symbols in the solution, normalizing the probabilities based on what values have already been generated. The EHM by itself does not take into account absolute positional importance at all. In order to address problems in which absolute positions are important, a variation of EHBSA that involved *templates* was proposed (Tsutsui, 2002). To generate new solutions, first a random string from the population was picked as a template. New solutions were then generated by removing random parts of the template string and generating the missing parts with sampling from the EHM. The resulting algorithm was shown to be better than most other EDAs on the traveling salesman problem. In another study, the node histogram sampling algorithm (NHBSA) of Tsutsui, Pelikan, and Goldberg (2006) considers a model capable of storing node frequencies at each position and again uses a template.

3.3 Real-Valued Vectors

EDAs discussed thus far were applicable to problems with candidate solutions represented by fixed-length strings over a finite alphabet. However, candidate solutions for many problems are represented using real-valued vectors. In these problems the variables cover an infinite domain so it is no longer possible to enumerate variables' values and their probabilities. This section discusses EDAs that can solve problems in the real-valued domain. There are two primary approaches to applying EDAs to the real-valued domain: (1) Map the real-valued variables into the discrete domain and use a discrete EDA on the resulting problem, and (2) use EDAs based on probabilistic models defined on real-valued variables.

3.3.1 Discretization

The most straightforward way to apply EDAs in the real-valued domain is to discretize the problem and use a discrete EDA on the resulting problem. In this way it is possible to directly use the discrete EDAs in the real-valued domain. However, a naive discretization can cause problems as some values close to each other in the continuous domain may become more distant in the discrete domain. In addition, the possible range of values must be known before the optimization starts. Finally, some regions of the search space are more densely covered with high quality solutions whereas others contain mostly solutions of low quality; this suggests that some regions require a more dense discretization than others. To deal with these difficulties, various approaches to adaptive discretization were developed using EDAs (Tsutsui, Pelikan, & Goldberg, 2001; Pelikan, Goldberg, & Tsutsui, 2003; Chen, Liu, & Chen, 2006; Suganthan, Hansen, Liang, Deb, Chen, Auger, & Tiwari, 2005). We discuss some of these next.

Tsutsui, Pelikan, and Goldberg (2001) proposed to divide the search space of each variable into subintervals using a histogram. Two different types of histogram models were used: fixed height and fixed width. The fixed-height histogram ensured that each discrete value would correspond to the same number of candidate solutions in the population; this allows for a more balanced discretization

where the areas that contain more high quality solutions also get more discrete values. The fixed-width histogram ensured that each discrete value corresponded to the interval of the same size. The results showed strong performance on the two-peaks and Rastrigin functions, which are often difficult without effective crossover operators.

Three different methods of discretization were tried (fixed-height histograms, fixed-width histograms and k-Means clustering) and combined with BOA to solve real-valued problems by Pelikan, Goldberg, and Tsutsui (2003). Adaptive mutation was also used after mapping the discrete values to the continuous domain. The resulting algorithm was shown to be successful on the two-peaks and deceptive functions.

Another way to deal with discretization was proposed by Chen, Liu, and Chen (2006). Their method uses the ECGA model and a split-on-demand (SoD) discretization to adjust on the fly how the real-valued variables are coded as discrete values. Loosely said, if an interval of discretization contains a large number of candidate solutions and these variables are biased towards one side of the interval, then that region is split into two to increase exploration. The resulting real-coded ECGA (rECGA) worked well on a set of benchmark test problems (Suganthan, Hansen, Liang, Deb, Chen, Auger, & Tiwari, 2005) designed to test real-valued optimization techniques.

While the above EDAs solved problems with candidate solutions represented by real-valued vectors, they manipulated these through discretization and variation operators based on a discrete representation. In the next section we cover EDAs that work directly with the real-valued variables themselves.

3.3.2 Direct Representation

The stochastic hill-climbing with learning by vectors of normal distributions (SHCLVND) (Rudlof & Köppen, 1996) works directly with a population of real-valued vectors. The model is represented as a vector of normal distributions, one for each variable. While the mean of each variable's distribution can be different, all the distributions share the same standard deviation. Over time the means are shifted towards the best candidate solutions generated and the deviation is slowly reduced by a multiplicative factor. Figure 6 shows an example of this type of a model.

One disadvantage of SHCLVND is that it assumes that each variable has the same standard deviation. Also, since it uses only a single normal distribution for each variable, it is only able to accurately capture distributions of samples that are all centered around a single point in the search space. In addition, it assumes that all the variables are independent. The following algorithms all attempt to alleviate one or more of these problems.

Sebag and Ducoulombier (1998) extend the idea of using a single vector of normal distributions by storing a different standard deviation for each variable. In this way it is able to perform better in scenarios where certain variables have higher variance than others. As in SHCLVND, however, all variables are assumed to be independent.

The estimation of Gaussian networks algorithm (EGNA) (Larrañaga, Etxeberria, Lozano, Pea, & na, 1999) works by creating a Gaussian network to model the interactions between variables in the selected population of solutions in each generation. This network is similar to a Bayesian network except that the variables are real-valued and locally each variable has its mean and variance computed by a linear function from its parents. The network structure is learned greedily using a continuous version of the BDe metric (Geiger & Heckerman, 1994), with a penalty term to prefer simpler models.

In the IDEA framework, Bosman and Thierens (2000) proposed models capable of capturing multiple basins of attraction or clusters of points by storing the joint normal and kernel distri-

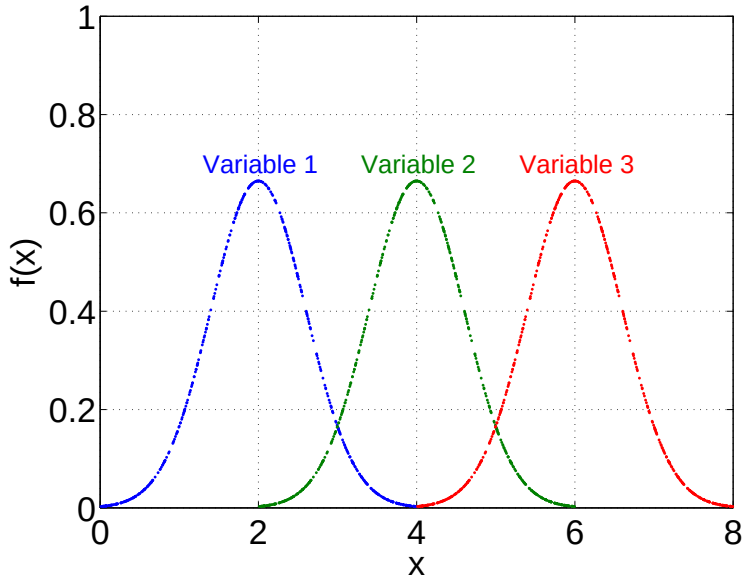


Figure 6: Example model generated by SHCLVND

butions. IDEA was able to outperform SHCLVND and other EDAs that used a single vector of Gaussian distributions on a set of six function optimization problems commonly used to test real-valued optimization techniques.

The mixed iterated density estimation evolutionary algorithm (mIDEA) (Bosman & Thierens, 2001a) uses mixtures of normal distributions. The model building in mIDEA starts by clustering the variables and fitting a probability distribution over each cluster. The final distribution used is then a weighted sum over the individual distributions. To evaluate dependencies between variables, the Bayesian information criterion (BIC) (Schwarz, 1978b) metric is used.

In EDAs described so far, the variables were treated either as all real-valued or as all discrete quantities. The mixed Bayesian optimization algorithm (mBOA) (Ocenasek, Kern, Hansen, & Koumoutsakos, 2004) can deal with both types of variables. Much as in hBOA, the probability distribution of each variable is represented as a decision tree. The internal nodes of each tree encode tests on variables that the corresponding variable depends on. For discrete variables, the branch taken during sampling is determined by whether or not the variable in the node is equal to a constant. For continuous variables, the branch taken is determined by whether the variable corresponding to that node is less than a constant. The leaves determine the values of the sampled variables. For discrete variables, the leaves contain the conditional probabilities of particular values of these variables. On the other hand, normal kernel distributions are is used for continuous variables.

The real-coded Bayesian optimization algorithm (rBOA) (Ahn, Ramakrishna, & Goldberg, 2004) tries to bring the power of BOA to the real-valued domain. rBOA uses a Bayesian network to describe the underlying structure of the problem and a mixture of Gaussians to describe the local distributions. The resulting algorithm was shown to outperform mIDEA on several real-valued deceptive problems.

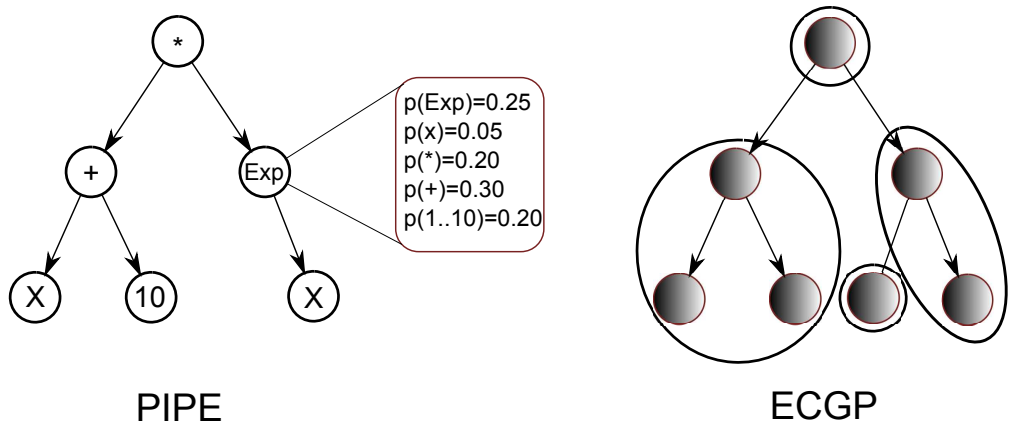


Figure 7: Example of two different models used in EDA-GP.

3.4 EDA-GP

After numerous successes in the design of EDAs for discrete and real-valued representations, a number of researchers have attempted to replicate these successes in the domain of genetic programming (GP) (Koza, 1992). In GP the task is to evolve a population of computer programs represented by labeled trees. In this domain, some additional challenges become evident. To start with, the length of candidate programs is expected to vary. Also, small changes in parent-child relationships can lead to large changes in the performance of the program, and often the relationship between operators is more important than their actual physical position in candidate programs. However, despite these additional challenges, even in this environment, EDAs have been successful. In the remainder of this section, we outline a few attempts to design EDAs for GP.

The probabilistic incremental program evolution (PIPE) (Salustowicz & Schmidhuber, 1997) uses a probabilistic prototype tree (PPT) to store the probability distribution of all functions and operators at each node of program trees. Initially the probability of all the functions and operators is set to represent the uniform distribution and used to generate the initial population. In each generation the values of the PPT are updated from the population of promising solutions. To generate new solutions, the distribution at each node is sampled to generate a new candidate program tree. Subtrees that are not valid due to invalid combination of operators or functions at the lowest level in the tree are pruned. Figure 7 shows an example PPT for PIPE. While PIPE does force positional dependence by using specific probabilities at each node, it does not take into account interactions between nodes. Nonetheless, due to the simplicity of the model used, the learning and sample procedure of PIPE remains relatively fast compared to many other approaches to EDA-based GP.

An extension of PIPE is the extended compact genetic programming (ECGP) (Sastry & Goldberg, 2000). Motivated by ECGA, ECGP splits nodes in the program trees into independent clusters. The algorithm starts by assuming that all nodes are independent. Then it proceeds by merging nodes into larger clusters based on the MDL metric similar to that used in ECGA. The individual clusters are treated as a single variable and the tree is used to generate new candidate

solutions. Figure 7 shows an example model generated by ECGP.

Due to the chaotic nature of program spaces, finding an accurate problem decomposition can be difficult over the entire search space of candidate programs. The meta-optimizing semantic evolutionary search (MOSES) (Looks, 2006) deals with this problem by first dynamically splitting up the search space into separate program subspaces called demes that are maintained simultaneously. hBOA is then applied to each individual deme to generate new programs within that deme, which can also lead to new demes being created.

Several EDAs were developed for GP using probabilistic models based on grammar rules. One such EDA is the stochastic grammar-based GP (SG-GP) (Ratle & Sebag, 2006), which starts by using a fixed context-free grammar and attaching default probabilities to each rule. Based on the promising solutions sampled during each generation, the probabilities of rules that perform well are gradually increased. While in the base algorithm, no positional information is stored, it is also possible to extend the algorithm to keep track of the level where each rule is used.

Further extending the idea of representing the probability distribution over candidate programs using probabilistic grammars, the program evolution with explicit learning (PEEL) (Shan, Mckay, Abbass, & Essam, 2003) attaches a depth and location parameter to each production rule. It starts with a small grammar and expands it by transforming a general rule that works at all locations into a specific grammar rule that only works at a specific depth and location. A metric based on ant colony optimization is used to ensure that rules are not excessively refined.

All the aforementioned algorithms based on grammatical evolution used a fixed context-free grammar. Grammar model-based program evolution (GMPE) (Shan, Mckay, & Baxter, 2004) goes beyond context-free grammars by allowing the algorithm itself to generate completely new grammar rules. GMPE starts by generating a random initial population of program trees. Then a minimal grammar is generated that is only able to generate the initial set of promising solutions. Once this is done, using the work based on theoretical natural language analysis, operators are used to create new rules and merge old rules together. The minimum message length (MML) (Wallace & Boulton, 1968) metric is used to compare grammars. This algorithm is very adaptable, being able to generate a broad variety of possible grammars. However, comparing all the grammars against each other is computationally expensive.

4 Advantages of Using EDAs

Viewing optimization as the process of updating a probabilistic model over candidate solutions provides EDAs with several important features that distinguish these algorithms from evolutionary algorithms and other, more conventional metaheuristics. This section reviews some of these important features.

Adaptive operators. One of the biggest advantages of EDAs over most other metaheuristics is their ability to *adapt their operators* to the structure of the problem. Most metaheuristics use fixed operators to explore the space of potential solutions. While problem-specific operators may be developed and are often used in practice, EDAs are able to do the tuning of the operator to the problem on their own. This important difference allows EDAs to solve some problems for which other algorithms scale poorly (Pelikan & Hartmann, 2006; Shah & Reed, 2010; Goldberg, 2002).

Problem structure. Besides just providing the solution to the problem, EDAs also provide optimization practitioners with a *roadmap of how the EDA solved the problem.* This roadmap

consists of the models that are calculated in each generation of the EDA, which represent samples of solutions of increasing quality. Mining these probabilistic models for information about the problem can reveal many problem-specific features, which can in turn be used to identify promising regions of the search space, dependency relationships between problem variables, or other important properties of the problem landscape. While gaining a better understanding of the problem domain is useful in its own right, the obtained information can be used to design problem-specific optimization techniques or speed up solution of new problem instances of similar type (Hauschild & Pelikan, 2009; Hauschild, Pelikan, Sastry, & Goldberg, 2008; Baluja, 2006; Schwarz & Ocenasek, 2000).

Prior knowledge exploitation. Practical solutions of enormously complex optimization problems often necessitates that the practitioners bias the optimization algorithm in some way based on *prior knowledge*. This is possible even with standard evolutionary algorithms, for example by injecting specific solutions into the population of candidate solutions or by biasing the populations using a local search. However, many approaches to biasing the search for the optimum tend to be ad-hoc and problem specific. EDAs provide the framework for more principled techniques to incorporate prior knowledge. For example, Bayesian statistics can be used to bias model building in EDAs towards instances that appear to more likely lead to the global optimum or towards probabilistic models that more closely correspond to the structure of the problem being solved. This can be done in a statistically meaningful way as will be demonstrated in section 5.6.

Reduced memory requirements. Incremental EDAs *reduce memory requirements* by replacing the population of candidate solutions by a probabilistic model. This allows practitioners to solve extremely large problems that cannot be solved with other techniques. For example, Sastry, Goldberg, and Llorà (2007) shows that solving a 2^{25} (over 33 million) bit onemax problem with a simple genetic algorithm takes about 700 gigabytes but the cGA described in 3.1.1 requires only a little over 128 megabytes.

5 Efficiency Enhancement Techniques for EDAs

While EDAs provide scalable solutions to many problems that are intractable with other techniques, solving enormously complex problems often necessitates that additional efficiency enhancement (EE) (Goldberg, 2002; Sastry, Pelikan, & Goldberg, 2006; Pelikan, 2005) techniques are used. There are two main computational bottlenecks that must be addressed by efficiency enhancement techniques for EDAs: (1) fitness evaluation and (2) model building.

Efficiency enhancements for EDAs can be roughly divided into the following categories (Pelikan, 2005):

1. Parallelization.
2. Evaluation relaxation.
3. Hybridization.
4. Time continuation.
5. Sporadic and incremental model building.
6. Incorporating problem-specific knowledge and learning from experience.

In the remainder of this section we will briefly review each of these approaches, with an emphasis on efficiency enhancement techniques that are specific to EDAs.

5.1 Parallelization

To enhance efficiency of any optimization technique, one may often parallelize the computation in some way. The most common approach to parallelization in EDAs and other metaheuristics is to parallelize fitness evaluation (Cantú-Paz, 2000). However, in the case of EDAs it is often also advantageous to parallelize model building. One of the most impressive results in parallelization of EDAs is the efficient parallel implementation of the compact genetic algorithm, which was successfully applied to a noisy optimization problem with over one billion decision variables (Sastry, Goldberg, & Llorà, 2007). Several approaches to parallelizing model building in advanced EDAs with multivariate models have also been proposed (Ocenasek, 2002; Ocenasek, Cantú-Paz, Pelikan, & Schwarz, 2006; Mendiburu, Miguel-Alonso, & Lozano, 2006).

5.2 Evaluation Relaxation

As previously discussed, one method to help alleviate the fitness bottleneck is parallelization. Nonetheless, to further improve performance of algorithms with expensive fitness evaluation, it is sometimes possible to eliminate some of the fitness evaluations by using approximate models of the fitness function, which can be evaluated much faster than the actual fitness function. Efficiency enhancement techniques based on this principle are called evaluation relaxation techniques (Goldberg, 2002; Smith, Dike, & Stegmann, 1995; Sastry, Goldberg, & Pelikan, 2001a; Pelikan & Sastry, 2004; Sastry, Pelikan, & Goldberg, 2004).

There are two basic approaches to evaluation relaxation: (1) endogenous models (Smith, Dike, & Stegmann, 1995; Sastry, Goldberg, & Pelikan, 2001a; Pelikan & Sastry, 2004; Sastry, Pelikan, & Goldberg, 2004) and (2) exogenous models (Sastry, 2001b; Albert, 2001). With endogenous models, the fitness values for some of the new candidate solutions are estimated based on the fitness values of the previously generated and evaluated solutions. With exogenous models, a faster but less accurate surrogate model is used for some of the evaluations, especially for those early in the run. Of course, the two approaches can be combined to maximize the benefits.

The first study (Sastry, Goldberg, & Pelikan, 2001b) that incorporated endogenous models in EDAs used the UMDA algorithm discussed in section 3.1.1. To estimate fitness, the probability vector was extended to also store statistics on the average fitness of all solutions with a 0 or a 1 in any string position. These data were then used to estimate fitness of new solutions. However, EDAs provide interesting opportunities for building extremely accurate yet computationally efficient fitness surrogate models that go way beyond the simple approach based on UMDA, because they provide the practitioners with detailed information about the structure of the problem. The endogenous-model approach when used with ECGA (Sastry, Pelikan, & Goldberg, 2004) and BOA (Pelikan & Sastry, 2004) can accurately approximate even complex fitness functions due to the additional information encoded about the problem structure in the EDA model, yielding speedups of several orders of magnitude even for only moderately sized problems (Pelikan & Sastry, 2004). This type of information is not available at all to other types of metaheuristics.

5.3 Hybridization

In many real-world applications, EDAs are combined with other optimization algorithms. Typically, simple and fast local search techniques—which can quickly locate the closest local optimum—are

incorporated into an EDA, reducing the problem of finding the global optimum to that of finding only the basin of attraction of the global optimum. As an example, consider the simple deterministic hill climber (DHC), which takes a candidate solution represented by a binary string and keeps performing single-bit flips on the solution that lead to the greatest improvement in fitness (Hart, 1994).

While even incorporating simple local search techniques can lead to significant improvements in time complexity of EDAs, sometimes more advanced optimization techniques are available that are tailored to the problem being solved. As an example, consider cluster exact approximation (CEA) (Hartmann, 1996), which can be incorporated into hBOA (Pelikan & Hartmann, 2006) when solving the problem of finding ground states of Ising spin-glass instances arranged on finite-dimensional lattices. Unlike DHC, CEA can flip many bits at once, often yielding solutions close to the global optimum after only a few iterations.

Incorporating local search is relatively straightforward in most metaheuristics. However, the use of probabilistic models of EDAs in optimization opens the door to the design of more advanced and powerful model-directed hybrids. Specifically, by exploiting the problem structure encoded by the probabilistic model, it is possible to design specialized local search techniques which can significantly outperform more conventional approaches to local search by using neighborhood operators that more closely correspond to the structure of the problem. For example, if two variables are strongly correlated and the value of one of them is modified, then it would make sense to consider modifying the value of the other variable as well. This idea was the motivation behind the building-block mutation operator used by Sastry and Goldberg (2004a) to speed up problem solving in ECGA. This operator worked by taking the best individual from the population and trying different combinations of bits in one of the independent linkage groups discovered by the model building phase, while leaving all the other linkage groups fixed; this was then repeated for each linkage group. This type of structural local search is simply not available to other metaheuristics.

Local search was also used to speed up the performance of BOA and hBOA by using information from Bayesian networks by Lima, Pelikan, Sastry, Butz, Goldberg, and Lobo (2006). In this work, substructural neighborhoods were defined as a variable and all its parents in the Bayesian network model discovered by BOA. Hillclimbing in the substructural space was then used on a proportion of the population. However, this technique did not take into account the context of possible overlapping interactions. Lima, Pelikan, Sastry, Butz, Goldberg, and Lobo (2006) also discussed other possible neighborhood structures that could be extracted from Bayesian networks. In a similar approach, Handa (2007) started with bit mutation in EBNA, but then resampled any variables that depended on mutated bits depending on the conditional probability of the new parent variable's value. In further work, Lima, Pelikan, Lobo, and Goldberg (2009) used loopy belief propogation (Pearl, 1988; Mendiburu, Santana, Lozano, & Bengoetxea, 2007) to find a more accurate substructural neighborhood based on the Bayesian model information and used that for local search.

5.4 Time continuation

In time continuation, the goal is to maximize performance of evolutionary algorithms by exploiting the trade-off between making more runs with a small population size and making fewer runs (or even only a single run) with a larger population size (Goldberg, 1999; Srivastava & Goldberg, 2001; Goldberg, 2002). For example, sometimes it is possible to solve a problem in one single generation with a large enough population size, but it may also be possible to solve this problem in many generations with a smaller population. Which is the most effective method is not always

readily apparent: The goal in time continuation is to pick the method most efficient for a particular problem, either to maximize solution quality given a fixed computational budget or to minimize time to achieve a solution of a given quality.

Problem information encoded in probabilistic models of EDAs creates opportunities for using this information to design more efficient optimization techniques by exploiting the time continuation tradeoffs. For example, Sastry and Goldberg (2004b) showed that for ECGA on separable problems of bounded difficulty, if the population was large enough for an accurate model of the underlying problem structure, an ECGA hybrid was able to solve these problems in a single generation by using a local searcher with the neighborhood structure based on the ECGA model. However, for problems with substantial amounts of noise, running the ECGA hybrid for a number of generations was preferable.

5.5 Sporadic and incremental model building

Model building in ECGA, hBOA and other similar EDAs usually consists of two parts: (1) learning the structure and (2) learning the parameters of the identified structure. Typically, learning the model structure is much more complex than learning the parameters (Ocenasek & Schwarz, 2000; Pelikan, 2005). However, since the model structure is expected to not change much between consequent iterations, one way to speed up model building is to use *sporadic model building*, in which the structure is updated only once once in a while (Pelikan, Sastry, & Goldberg, 2008).

Since the model structure is expected to not change much over time and making incremental changes to model structure is usually much simpler than building the structure from scratch, it may also be advantageous to change the model structure only incrementally without rebuilding the model from scratch in every iteration. This is the basic idea of *incremental model building* (Etxeberria & Larrañaga, 1999b).

5.6 Incorporating problem-specific knowledge and learning from experience

EDAs typically do not require any information about the problem being solved except for the representation of candidate solutions and the fitness function. Nonetheless, if problem-specific information is available, it may be possible to use this information to improve performance of these algorithms significantly. There are two basic approaches to speed up EDAs by incorporating problem-specific knowledge: (1) bias the procedure for generating the initial population (Schwarz & Ocenasek, 2000; Sastry, 2001a; Pelikan & Goldberg, 2003) and (2) bias or restrict the model building procedure (Schwarz & Ocenasek, 2000; Mühlenbein & Mahnig, 2002; Baluja, 2006). For both these approaches, we may either (1) hard code the modifications based on prior problem-specific knowledge (Hauschild, Pelikan, Sastry, & Goldberg, 2008; Baluja, 2006; Schwarz & Ocenasek, 2000) or (2) develop automated procedures to improve EDA performance by learning from previous EDA runs on problems of similar type (learning from experience) (Hauschild & Pelikan, 2009).

One technique used to bias the initial population towards good solutions (and, consequently, to also improve model quality) is called *seeding* (Schwarz & Ocenasek, 2000; Pelikan & Goldberg, 2003; Sastry, 2001a). Seeding works by inserting high-quality solutions into the initial population. These high-quality solutions can be either obtained from previous runs on similar problems, provided by a specialized heuristic (Schwarz & Ocenasek, 2000; Pelikan & Goldberg, 2003), or created in some way from high-quality solutions of smaller instances of the same problem (Sastry, 2001a).

While seeding can work with many types of algorithms, EDAs offer us a wealth of new options for using prior information in a principled way. One of the earliest attempts to bias model building

in EDAs based on prior problem-specific knowledge was made by Schwarz and Ocenasek (2000), who biased BOA model building in graph bipartitioning by giving those edges contained in the underlying graph a higher priority than other edges. Mühlenbein and Mahnig (2002) also considered graph bipartitioning but in this case only allowed edges in the underlying graph. Baluja (2006) also only allowed edges in the underlying graph in his work on graph coloring.

To develop a method that was more broadly applicable, Hauschild, Pelikan, Sastry, and Goldberg (2008) proposed two different methods to restrict or penalize the allowable edges in hBOA model building. The first method used a distance metric defined in such a way that the greater the distance between two variables in the problem, the less expected interaction between these variables. Using a parameter to specify the maximum allowable distance to still consider an edge, this method was able to cut down on the number of edges considered during model building. The second method was based on the percentage of runs in which an edge was encountered in hBOA models. Using hBOA to solve a small number of sample problem instances, the resulting data were then used to speed up hBOA model building in subsequent runs on newer problem instances. This work was later extended (Hauschild & Pelikan, 2009) to bias the BD metric itself based on the probability that an edge was used in the previous runs.

6 Summary and Conclusions

EDAs are among the most powerful evolutionary algorithms currently available, and there are numerous applications where EDAs have been shown to solve problems unsolvable with other existing techniques. Nonetheless, EDAs are capable of not only solving many difficult problems, but they also provide practitioners with a great deal of information about *how* the problem was solved. The ability to provide practitioners with useful information about the problem landscape is a feature that is highly desirable yet not offered by virtually any other general optimization technique. In addition, most EDAs offer additional advantages over the more conventional evolutionary algorithms and other metaheuristics, such as the ability to represent the population more efficiently using a probabilistic model or include prior information of various forms in a rigorous manner.

EDAs use a large variety of probabilistic models, ranging from probability vectors to Bayesian and Markov networks. This diversity allows practitoners to solve a great variety of problems, from the simple to the complex, from the real-valued domain to the discrete one. Given almost any problem, it should be possible for practitioners to select an EDA that can solve it. The key is to ensure that the class of probabilistic models used allows EDAs to effectively capture features of high quality solutions that make these solutions better.

Acknowledgments

This project was sponsored by the National Science Foundation under CAREER grant ECS-0547013 and by the University of Missouri in St. Louis through the High Performance Computing Collaboratory sponsored by Information Technology Services. 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 National Science Foundation.

References

Ackley, D. H. (1987). An empirical study of bit vector function optimization. *Genetic Algorithms and Simulated Annealing*, 170–204.

Ahn, C. W., Ramakrishna, R. S., & Goldberg, D. E. (2004). Real-Coded Bayesian Optimization Algorithm: Bringing the Strength of BOA into the Continuous World. In Deb, K. (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2004* (pp. 840–851). Springer-Verlag, Berlin.

Albert, L. A. (2001). *Efficient genetic algorithms using discretization scheduling*. Master's thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL.

Arst, R., Minsker, B. S., & Goldberg, D. E. (2002). Comparing advanced genetic algorithms and simple genetic algorithms for groundwater management. *Proceedings of the American Society of Civil Engineers (ASCE) Environmental & Water Resources Institute (EWRI) 2002 Water Resources Planning & Management Conference*, Roanoke, VA.

Bacardit, J., Stout, M., Hirst, J. D., Sastry, K., Llorà, X., & Krasnogor, N. (2007). Automated alphabet reduction method with evolutionary algorithms for protein structure prediction. *Genetic and Evolutionary Computation Conference (GECCO-2007)*, 346–353.

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning* (Tech. Rep. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University.

Baluja, S. (2006). Incorporating a priori knowledge in probabilistic-model based optimization. In Pelikan, M., Sastry, K., & Cantú-Paz, E. (Eds.), *Scalable optimization via probabilistic modeling: From algorithms to applications* (pp. 205–219). Springer.

Baluja, S., & Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. *Proceedings of the International Conference on Machine Learning*, 30–38.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, *6*(2), 154–160.

Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., & Boeres, C. (2000). Inexact graph matching using learning and simulation of bayesian networks. an empirical comparison between different approaches with synthetic data.

Bosman, P. A., & Thierens, D. (2001a). Advancing continuous ideas with mixture distributions and factorization selection metrics. In *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO2001* (pp. 208–212). Morgan Kaufmann.

Bosman, P. A., & Thierens, D. (2001b). Crossing the road to efficient ideas for permutation problems.

Bosman, P. A., & Thierens, D. (2001c). New ideas and more ice by learning and using unconditional permutation factorizations. *Late-Breaking Papers of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 16–23.

Bosman, P. A. N., & Thierens, D. (1999). Linkage information processing in distribution estimation algorithms. *Genetic and Evolutionary Computation Conference (GECCO-99)*, *I*, 60–67.

Bosman, P. A. N., & Thierens, D. (2000). Continuous iterated density estimation evolutionary algorithms within the IDEA framework. *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 197–200.

Cantú-Paz, E. (2000). *Efficient and accurate parallel genetic algorithms*. Boston, MA: Kluwer.

Chen, C.-H., Liu, W.-N., & Chen, Y.-P. (2006). Adaptive discretization for probabilistic model building genetic algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06 (pp. 1103–1110). New York, NY, USA: ACM.

Chen, C.-H., & p. Chen, Y. (2007). Real-coded ECGA for economic dispatch. *Genetic and Evolutionary Computation Conference (GECCO-2007)*, 1920–1927.

Chickering, D. M., Heckerman, D., & Meek, C. (1997). *A Bayesian approach to learning Bayesian networks with local structure* (Technical Report MSR-TR-97-07). Redmond, WA: Microsoft Research.

Chow, C., & Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, *14*, 462–467.

De Bonet, J. S., Isbell, C. L., & Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. *Advances in neural information processing systems (NIPS-97)*, *9*, 424–431.

Deb, K., & Goldberg, D. E. (1991). *Analyzing deception in trap functions* (IlliGAL Report No. 91009). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Ducheyne, E., De Baets, B., & De Wulf, R. (2004). Probabilistic models for linkage learning in forest management. In Jin, Y. (Ed.), *Knowledge incorporation in evolutionary computation* (pp. 177–194). Springer.

Etxeberria, R., & Larrañaga, P. (1999a). Global optimization using Bayesian networks. *Second Symposium on Artificial Intelligence (CIMAF-99)*, 332–339.

Etxeberria, R., & Larrañaga, P. (1999b). Global optimization using Bayesian networks. In Rodriguez, A. A. O., Ortiz, M. R. S., & Hermida, R. S. (Eds.), *Second Symposium on Artificial Intelligence (CIMAF-99)* (pp. 332–339). Habana, Cuba: Institude of Cybernetics, Mathematics, and Physics and Ministry of Science, Technology and Environment.

Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical models* (pp. 421–459). MIT Press.

Geiger, D., & Heckerman, D. (1994). Learning gaussian networks. In *UAI* (pp. 235–243).

Goldberg, D. E. (1999). Using time efficiently: Genetic-evolutionary algorithms and the continuation problem. *Genetic and Evolutionary Computation Conference (GECCO-99)*, 212–219. Also IlliGAL Report No. 99002.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Kluwer.

Handa, H. (2007). The effectiveness of mutation operation in the case of estimation of distribution algorithms. *Biosystems*, *87*(2-3), 243 – 251. Papers presented at the Sixth International Workshop on Information Processing in Cells and Tissues, York, UK, 2005 - IPCAT 2005, Information Processing in Cells and Tissues.

Harik, G. (1999). *Linkage learning via probabilistic modeling in the ECGA* (IlliGAL Report No. 99010). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. *International Conference on Genetic Algorithms (ICGA-95)*, 24–31.

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *International Conference on Evolutionary Computation (ICEC-97)* (pp. 7–12). Piscataway, NJ: IEEE Press.

Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1997). *The compact genetic algorithm* (IlliGAL Report No. 97006). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Hart, W. E. (1994). *Adaptive global optimization with local search*. Doctoral dissertation, University of California, San Diego, San Diego, CA.

Hartmann, A. K. (1996). Cluster-exact approximation of spin glass ground states. *Physica A*, *224*, 480.

Hauschild, M., Pelikan, M., Sastry, K., & Goldberg, D. E. (2008). Using previous models to bias structural learning in the hierarchical BOA. *Genetic and Evolutionary Computation Conference (GECCO-2008)*, 415–422.

Hauschild, M. W., & Pelikan, M. (2009). Intelligent bias of network structures in the hierarchical boa. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09 (pp. 413–420). New York, NY, USA: ACM.

Hayes, M. S., & Minsker, B. S. (2005). Evaluation of advanced genetic algorithms applied to groundwater remediation design. *Proceedings of the American Society of Civil Engineers (ASCE) Environmental & Water Resources Institute (EWRI) World Water & Environmental Resources Congress 2005 & Related Symposia*, Anchorage, AK.

Heckerman, D., Geiger, D., & Chickering, D. M. (1994). *Learning Bayesian networks: The combination of knowledge and statistical data* (Technical Report MSR-TR-94-09). Redmond, WA: Microsoft Research.

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, *20*, 197–243. 10.1007/BF00994016.

Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J. F., & Kanal, L. N. (Eds.), *Uncertainty in Artificial Intelligence* (pp. 149–163). Amsterdam, London, New York: Elsevier.

Kollat, J. B., Reed, P. M., & Kasprzyk, J. R. (2008). A new epsilon-dominance hierarchical Bayesian optimization algorithm for large multi-objective monitoring network design problems. *Advances in Water Resources*, *31*(5), 828–845.

Koopmans, T. C., & Beckmann, M. J. (1957). Assignment problems and the location of economic activities. *Econometrica*, *25*, 53–76.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: The MIT Press.

Larrañaga, P., Etxeberria, R., Lozano, J. A., Pea, J., & na, J. M. P. (1999). Optimization by learning and simulation of bayesian and gaussian networks.

Larrañaga, P., & Lozano, J. A. (Eds.) (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation*. Boston, MA: Kluwer.

Lima, C. F., Pelikan, M., Lobo, F. G., & Goldberg, D. E. (2009). Loopy substructural local search for the bayesian optimization algorithm. In Stützle, T., Birattari, M., & Hoos, H. H. (Eds.), *SLS*, Volume 5752 of *Lecture Notes in Computer Science* (pp. 61–75). Springer.

Lima, C. F., Pelikan, M., Sastry, K., Butz, M. V., Goldberg, D. E., & Lobo, F. G. (2006). Substructural neighborhoods for local search in the Bayesian optimization algorithm. *Parallel Problem Solving from Nature*, 232–241.

Lipinski, P. (2007). ECGA vs. BOA in discovering stock market trading experts. *Genetic and Evolutionary Computation Conference (GECCO-2007)*, 531–538.

Looks, M. (2006, 11 December). *Competent program evolution*. Doctor of science, Washington University, St. Louis, USA.

Marascuilo, L. A., & McSweeney, M. (1977). *Nonparametric and distribution-free methods for the social sciences*. CA: Brooks/Cole Publishing Company.

Mendiburu, A., Miguel-Alonso, J., & Lozano, J. A. (2006). Implementation and performance evaluation of a parallelization of estimation of bayesian network algorithms. *Parallel Processing Letters*, *16*(1), 133–148.

Mendiburu, A., Santana, R., Lozano, J. A., & Bengoetxea, E. (2007). A parallel framework for loopy belief propagation. In *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, GECCO '07 (pp. 2843–2850). New York, NY, USA: ACM.

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Mühlenbein, H. (2008). *Convergence of estimation of distribution algorithms for finite samples* (Technical Report). Sankt Augustin, Germany: Fraunhofer Institut Autonomous intelligent Systems.

Mühlenbein, H., & Mahnig, T. (1999). FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, *7*(4), 353–376.

Mühlenbein, H., & Mahnig, T. (2002). Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning. *International Journal on Approximate Reasoning*, *31*(3), 157–192.

Mühlenbein, H., & Paaß, G. (1996a). From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature*, 178–187.

Mühlenbein, H., & Paaß, G. (1996b). From recombination of genes to the estimation of distributions I. Binary parameters. In Eiben, A., Bäck, T., Shoenauer, M., & Schwefel, H. (Eds.), *Parallel Problem Solving from Nature* (pp. 178–187). Berlin: Springer Verlag.

Ocenasek, J. (2002). *Parallel estimation of distribution algorithms*. Doctoral dissertation, Faculty of Information Technology, Brno University of Technology, Brno.

Ocenasek, J., Cantú-Paz, E., Pelikan, M., & Schwarz, J. (2006). Design of parallel estimation of distribution algorithms. In Pelikan, M., Sastry, K., & Cantú-Paz, E. (Eds.), *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications* (pp. 187–203). Springer.

Ocenasek, J., Kern, S., Hansen, N., & Koumoutsakos, P. (2004). A mixed bayesian optimization algorithm with variance adaptation. In Yao, X., Burke, E., Lozano, J. A., Smith, J., Merelo-Guervs, J. J., Bullinaria, J. A., Rowe, J., Tino, P., Kabn, A., & Schwefel, H.-P. (Eds.), *Parallel Problem Solving from Nature - PPSN VIII*, Volume 3242 of *Lecture Notes in Computer Science* (pp. 352–361). Springer Berlin / Heidelberg.

Ocenasek, J., & Schwarz, J. (2000). The parallel Bayesian optimization algorithm. In *Proceedings of the European Symposium on Computational Inteligence* (pp. 61–67). Physica-Verlag.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Pelikan, M. (2005). *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*. Springer-Verlag.

Pelikan, M., & Goldberg, D. E. (2003). Hierarchical BOA solves Ising spin glasses and MAXSAT. *Genetic and Evolutionary Computation Conference (GECCO-2003)*, *II*, 1275–1286.

Pelikan, M., & Goldberg, D. E. (2006). Hierarchical Bayesian optimization algorithm. In Pelikan, M., Sastry, K., & Cantú-Paz, E. (Eds.), *Scalable optimization via probabilistic modeling: From algorithms to applications* (pp. 63–90). Springer.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000). Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation*, *8*(3), 311–341. Also IlliGAL Report No. 98013.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*(1), 5–20.

Pelikan, M., Goldberg, D. E., & Tsutsui, S. (2003). Getting the best of both worlds: Discrete and continuous genetic and evolutionary algorithms in concert. *Information Sciences*, *156*(3-4), 147 – 171. Evolutionary Computation.

Pelikan, M., & Hartmann, A. K. (2006). Searching for ground states of Ising spin glasses with hierarchical BOA and cluster exact approximation. In Pelikan, M., Sastry, K., & Cantú-Paz, E. (Eds.), *Scalable optimization via probabilistic modeling: From algorithms to applications* (pp. 333–349). Springer.

Pelikan, M., & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. *Advances in Soft Computing—Engineering Design and Manufacturing*, 521–535.

Pelikan, M., & Sastry, K. (2004). Fitness inheritance in the Bayesian optimization algorithm. *Genetic and Evolutionary Computation Conference (GECCO-2004)*, *2*, 48–59.

Pelikan, M., Sastry, K., & Goldberg, D. E. (2008). Sporadic model building for efficiency enhancement of the hierarchical BOA. *Genetic Programming and Evolvable Machines*, *9*(1), 53–84.

Pelikan, M., Tsutsui, S., & Kalapala, R. (2007). Dependency trees, permutations, and quadratic assignment problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07 (pp. 629–629). New York, NY, USA: ACM.

Peña, J., Lozano, J., & Larrañaga, P. (2004). Unsupervised learning of bayesian networks via estimation of distribution algorithms: an application to gene expression data clustering. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *12*, 63–82.

Petrovski, A., Shakya, S., & Mccall, J. (2006). Optimising cancer chemotherapy using an estimation of distribution algorithm and genetic algorithms. *Genetic and Evolutionary Computation Conference (GECCO-2006)*, 413–418.

Ratle, A., & Sebag, M. (2006). Avoiding the bloat with stochastic grammar-based genetic programming. *CoRR*, *abs/cs/0602022*.

Robles, V., de Miguel, P., & Larrañaga, P. (2002). Solving the traveling salesman problem with EDAs. In Larrañaga, P., & Lozano, J. A. (Eds.), *Estimation of Distribution Algorithms. A*

New Tool for Evolutionary Computation (pp. 227–238). Boston/Dordrecht/London: Kluwer Academic Publishers.

Rudlof, S., & Köppen, M. (1996). Stochastic hill climbing with learning by vectors of normal distributions. Nagoya, Japan.

Salustowicz, R. P., & Schmidhuber, J. (1997). Probabilistic incremental program evolution: Stochastic search through program space. *Proceedings of the European Conference of Machine Learning (ECML-97)*, *1224*, 213–220.

Sastry, K. (2001a). *Efficient atomic cluster optimization using a hybrid extended compact genetic algorithm with seeded population* (IlliGAL Report No. 2001018). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Sastry, K. (2001b). *Evaluation-relaxation schemes for genetic and evolutionary algorithms*. Master's thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL.

Sastry, K., & Goldberg, D. E. (2000). *On extended compact genetic algorithm* (IlliGAL Report No. 2000026). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Sastry, K., & Goldberg, D. E. (2004a, 26-30). Designing competent mutation operators via probabilistic model building of neighborhoods. *Genetic and Evolutionary Computation Conference (GECCO-2004)*, 114–125.

Sastry, K., & Goldberg, D. E. (2004b). Let's get ready to rumble: Crossover versus mutation head to head. *Genetic and Evolutionary Computation Conference (GECCO-2004)*, 126–137.

Sastry, K., Goldberg, D. E., & Llorà, X. (2007). Towards billion bit optimization via parallel estimation of distribution algorithm. *Genetic and Evolutionary Computation Conference (GECCO-2007)*, 577–584.

Sastry, K., Goldberg, D. E., & Pelikan, M. (2001a). Don't evaluate, inherit. *Genetic and Evolutionary Computation Conference (GECCO-2001)*, 551–558.

Sastry, K., Goldberg, D. E., & Pelikan, M. (2001b). Don't evaluate, inherit. In Spector, L., Goodman, E. D., Wu, A., Langdon, W. B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M. H., & Burke, E. (Eds.), *Genetic and Evolutionary Computation Conference (GECCO-2001)* (pp. 551–558). San Francisco, CA: Morgan Kaufmann.

Sastry, K., Pelikan, M., & Goldberg, D. E. (2004). Efficiency enhancement of genetic algorithms via building-block-wise fitness estimation. *Proceedings of the IEEE International Conference on Evolutionary Computation*, 720–727.

Sastry, K., Pelikan, M., & Goldberg, D. E. (2006). Efficiency enhancement of estimation of distribution algorithms. In Pelikan, M., Sastry, K., & Cantú-Paz, E. (Eds.), *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications* (pp. 161–185). Springer.

Schwarz, G. (1978a). Estimating the dimension of a model. *The Annals of Statistics*, *6*, 461–464.

Schwarz, G. (1978b). Estimating the dimension of a model. *Annals of Statistics*, *6*, 461–464.

Schwarz, J., & Ocenasek, J. (2000). A problem-knowledge based evolutionary algorithm KBOA for hypergraph partitioning. Personal communication.

Sebag, M., & Ducoulombier, A. (1998). Extending population-based incremental learning to continuous search spaces. In *Parallel Problem Solving from Nature* (pp. 418–427). Berlin Heidelberg: Springer Verlag.

Shah, R., & Reed, P. (2010). Comparative analysis of multiobjective evolutionary algorithms for random and correlated instances of multiobjective d-dimensional knapsack problems. *European Journal of Operations Research*. In revision.

Shakya, S., & Santana, R. (2008). An eda based on local markov property and gibbs sampling. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08 (pp. 475–476). New York, NY, USA: ACM.

Shan, Y., Mckay, R. I., Abbass, H. A., & Essam, D. (2003). Program evolution with explicit learning: a new framework for program automatic synthesis. In *University College, University of New South* (pp. 1639–1646). IEEE Press.

Shan, Y., Mckay, R. I., & Baxter, R. (2004). Grammar model-based program evolution. In *In Proceedings of the 2004 IEEE Congress on Evolutionary Computation* (pp. 478–485). IEEE Press.

Simon, H. A. (1968). *The sciences of the artificial*. Cambridge, MA: The MIT Press.

Smith, R. E., Dike, B. A., & Stegmann, S. A. (1995). Fitness inheritance in genetic algorithms. *Proceedings of the ACM Symposium on Applied Computing*, 345–350.

Srivastava, R., & Goldberg, D. E. (2001). Verification of the theory of genetic and evolutionary continuation. *Genetic and Evolutionary Computation Conference (GECCO-2001)*, 551–558. Also IlliGAL Report No. 2001007.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., & Tiwari, S. (2005). *Problem definitions and evaluation criteria for the CEC 2005 Special Session on Real Parameter Optimization* (Technical Report). Nanyang Technological University.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. *International Conference on Genetic Algorithms (ICGA-93)*, 38–45.

Tsutsui, S. (2002). Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram. In *Proc. of the 7th Int. Conf. on Parallel Problem Solving from Nature (PPSN VII* (pp. 224–233). Springer-Velag.

Tsutsui, S., Pelikan, M., & Goldberg, D. E. (2001). Evolutionary algorithm using marginal histogram models in continuous domain. *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 230–233.

Tsutsui, S., Pelikan, M., & Goldberg, D. E. (2006). *Node histogram vs. edge histogram: A comparison of pmbgas in permutation domains* (MEDAL Report No. 2006009). Missouri Estimation of Distribution Algorithms Laboratory, University of Missour–St. Louis, St. Louis, MO.

Wallace, C. S., & Boulton, D. M. (1968). An information measure for classification. *Computer Journal*, *11*(2), 185–194.

Yu, T.-L., Santarelli, S., & Goldberg, D. E. (2006). Military antenna design using a simple genetic algorithm and hBOA. In Pelikan, M., Sastry, K., & Cantú-Paz, E. (Eds.), *Scalable optimization via probabilistic modeling: From algorithms to applications* (pp. 275–289). Springer.