Swarm and Evolutionary Computation 1 (2011) 111–128

Contents lists available at SciVerse ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Survey paper

An introduction and survey of estimation of distribution algorithms

Mark Hauschild *, Martin Pelikan

Missouri Estimation of Distribution Algorithms Laboratory (MEDAL), Department of Mathematics and Computer Science, University of Missouri in St. Louis, One University Blvd, St. Louis, MO 63121-4400, United States

ARTICLE INFO

Article history:
Received 4 March 2011
Received in revised form
29 August 2011
Accepted 30 August 2011
Available online 14 September 2011

Keywords:
Stochastic optimization
Estimation of distribution algorithms
Probabilistic models
Model building
Decomposable problems
Evolutionary computation

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 probabilistic 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 and some of the key theoretical results relevant to EDAs are outlined.

1. Introduction

Estimation of distribution algorithms [1–4] 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 [5], military antenna design [6], multi-objective knapsack [7], groundwater remediation design [8,9], amino-acid alphabet reduction for protein structure prediction [10], identification of clusters of genes with similar expression profiles [11], economic dispatch [12], forest management [13], portfolio management [14], cancer chemotherapy optimization [15], environmental monitoring network design [16], 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.

This survey aims to provide the reader with enough information to not only understand what an EDA is, but also to implement a basic EDA. Compared to the past surveys on this topic [4,17], this survey targets a broader audience, including readers who are not familiar with genetic algorithms or evolutionary computation. In addition, this survey covers many algorithms and methods that have not been covered in previously published surveys on this topic [4,17].

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 discusses similarities of EDAs and some of the most closely related stochastic optimization techniques. Section 5 reviews advantages and disadvantages of EDAs compared to other metaheuristics. Section 6 discusses the most common efficiency enhancements that may be incorporated into EDAs to speed up their operation. Section 7 gives a broad overview of some of the most important theoretical results in the field of EDAs. Section 8 contains pointers to additional information on EDAs for the interested reader, such as the important journals and conferences in the field, as well free software implementations. Lastly, Section 9 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

* Corresponding author.
E-mail addresses: mwh308@umsl.edu, markhauschild@gmail.com (M. Hauschild), pelikan@cs.umsl.edu (M. Pelikan).

2210-6502/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.swevo.2011.08.003

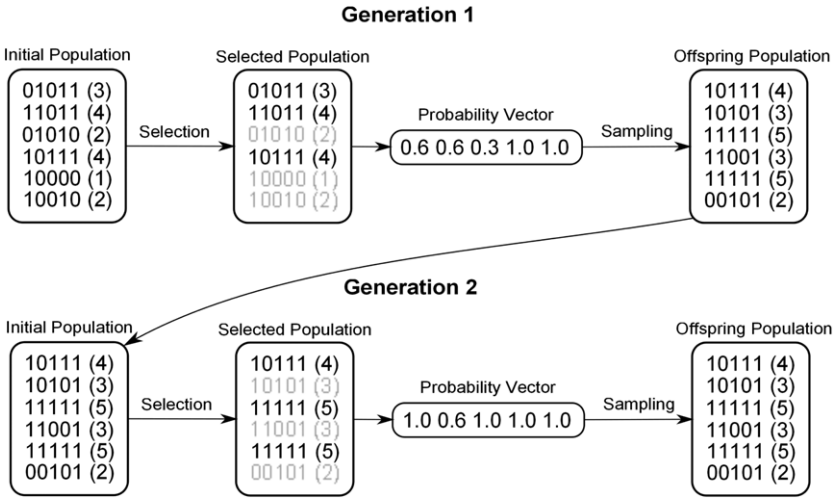


Fig. 1. Two generations of a simple EDA using a probability vector to solve onemax.

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) [1–4] 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, possibly 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

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

where n is the number of variables and X_i is the ith variable in the problem (ith 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 Fig. 1.

It is clear from the first generation that the procedure is having positive effects. The offspring population already contains

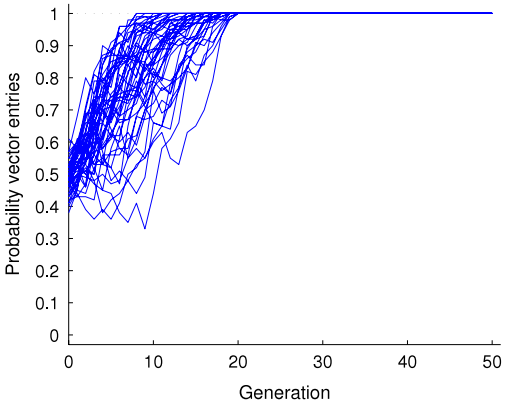


Fig. 2. Proportions of 1s in a probability vector of a simple EDA on the onemax problem of $n = 50$ using a population of $N = 100$ candidate solutions.

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 probabilities of ones in each in different positions of solution strings from an example run of an EDA on a bigger onemax problem are shown in Fig. 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 even in this experiment. While the probabilities of 1s in some positions do fluctuate in value in the initial iterations, eventually all the probability vector entries become 1.

Assuming that the population size is large enough to ensure reliable convergence [18,19], 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) [20,21]. 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

$$\text{trap}_5(u) = \begin{cases} 5 & \text{if } u = 5 \\ 4 - u & \text{otherwise,} \end{cases} \tag{2}$$

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 [22]; 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 Fig. 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 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 $N = 100$. Fig. 4(a) 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

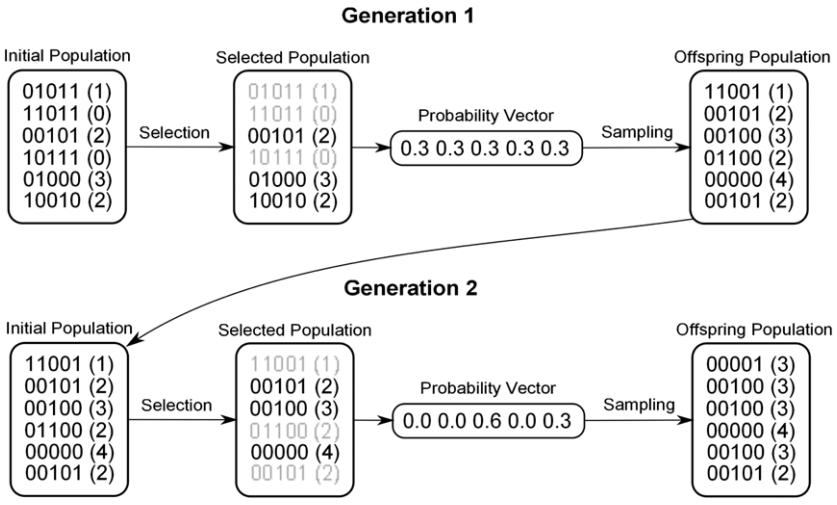
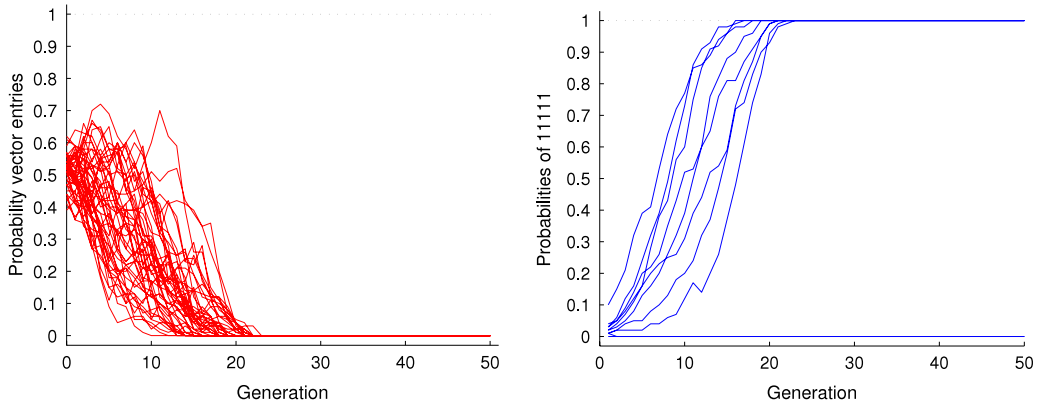


Fig. 3. Two generations of a simple EDA using a probability vector to solve trap-5.



(a) EDA based on the probability vector.

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

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

with a 0 in any position is greater than the average fitness of solutions with a 1 in that position [21], 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 (ECGA) [23] 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 [23].

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. Fig. 4(b) shows the proportions of blocks of five 1s 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 [24]. 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 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. 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.

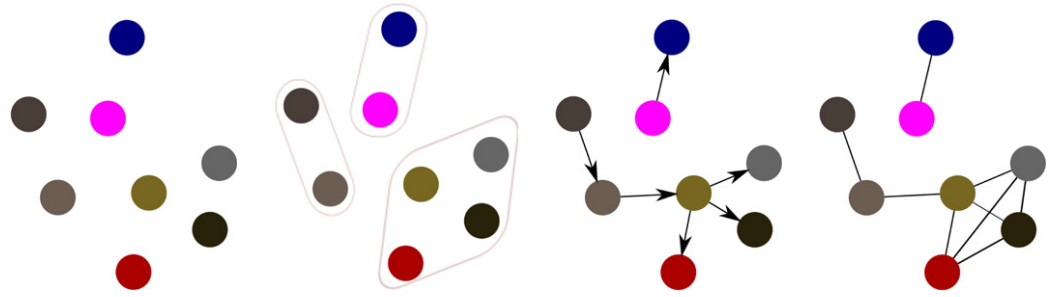
3.1.1. Univariate models

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. Fig. 5(a) shows an illustration of this type of model.

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

$$p(X_1, X_2, \ldots, X_n) = p(X_1)p(X_2), \ldots, p(X_n)$$

where $p(X_i)$ is the probability of variable X_i, and $p(X_1, X_2, \ldots, X_n)$ is the probability of the candidate solution $(X_1, X_2, \ldots, X_n)$. 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.



(a) Independent. (b) ECGA. (c) Bayesian. (d) Markov.

Fig. 5. Examples of graphical models produced by different EDAs.

One example of a univariate EDA is the univariate marginal distribution algorithm (UMDA) [25], which is the algorithm we used to solve the onemax problem in Section 2. UMDA works on binary strings and uses the probability vector $p = (p_1, p_2, \ldots, p_n)$ 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.

While most EDAs work by keeping a population of candidate solutions, *incremental* EDAs fully replace the population with the probabilistic model. The model is first set to give a uniform distribution over the solution space. Each generation, several solutions are created and the most fit solution or several most fit solutions are selected. The model is then slightly biased towards solutions of this type. In this way the model is incrementally improved over time, without the expense of storing large populations each generation.

One incremental univariate EDA is the population-based incremental learning (PBIL) [1] algorithm, which works on binary strings. 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 probability of a 1 in each position to 0.5. In each generation of PBIL, the probability vector is sampled to generate a small set of solutions using the same sampling procedure as in UMDA. The best solutions in this set are selected and for each variable in each selected solution, the corresponding entry in the probability vector is shifted by

$$p_i = (p_i * (1.0 - LR)) + (LR * v_i)$$

where p_i is the probability of generating a 1 in bit position i, v_i is the ith position in the solution string and LR is the learning rate 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) [26] 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 to determine the best of the two solutions (winner). If at any particular position the winning solution's bit is different from the losing solution's bit, the corresponding 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 have a much smaller memory footprint, which can be useful when trying to solve extremely large problems [27]; we return to this topic in Section 5.

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 (for example, when solving the trap-5 problem discussed in Section 2.3). 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) [28] 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 maximum mutual information with the variable that was added last. This process is repeated until all the variables are selected. The resulting tree model consists of a single chain of dependences, with each parent having exactly one child. Once the structure of the model has been completed, the conditional probability of each variable based on its parent is calculated from the promising solutions. Given a permutation of the n variables in a problem, $\pi = i_1, i_2, \ldots, i_n$, MIMIC decomposes the probability distribution of $p(X_1, X_2, \ldots, X_n)$ as

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

where $p(X_{i_j}|X_{i_{j+1}})$ 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}.

Baluja and Davies [29] 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 with respect to the true distribution [30]. 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) [31] 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 [32] 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 [33,31]. 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) [23] 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 [34]. 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 Fig. 5(b).

Many problems contain highly overlapping subproblems that cannot be accurately modeled by dividing the problem into independent clusters. The Bayesian optimization algorithm (BOA) [35] 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. Fig. 5(c) 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. A greedy algorithm is then used to add edges to the network, adding the edge that gives the most improvement according to the Bayesian–Dirichlet (BD) metric [36]. Since the BD metric has a tendency to favor overly complex models, usually an upper bound on the number of allowable parents is set or a prior bias on the network structure is introduced to prefer simpler models [37]. New candidate solutions are generated by sampling the probability distribution encoded by the built network using probabilistic logic sampling [38].

The estimation of Bayesian network algorithm (EBNA) [39] and the learning factorized distribution algorithm (LFDA) [24] also use Bayesian networks to model the promising solutions. EBNA and LFDA use the Bayesian information criterion (BIC) [40] 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 incorporate prior information into problem solving, as discussed in Section 6.6.

Many complex problems in the real world are hierarchical in nature [41]. A hierarchical problem is a problem composed of subproblems, with each subproblem being a hierarchical problem itself until the bottom level is reached [41]. 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 hierarchical problems presents new challenges for EDAs. First, on each level of problem solving, the hierarchical problem solver must be capable of decomposing the problem. Second, 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) [42] is able to solve many difficult hierarchically decomposable problems by extending BOA in two 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 used. Specifically, hBOA uses Bayesian networks with local structures [43,44] to allow feasible learning and sampling of more complex networks than would be possible with conventional Bayesian networks. In addition, the preservation of alternative solutions over time is ensured by using a niching technique called restricted tournament replacement (RTR) [45] 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 [46].

Another type of model that can encode multivariate interactions 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 [47]. Nonetheless, sampling Markov networks is 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 [48] uses Gibbs sampling to generate new solutions from its model in the Markovianity based optimization 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 Fig. 5(d).

The affinity propagation EDA (AffEDA) designed by Santana et al. [49] starts by first generating a mutual information matrix between all variables. An affinity propagation algorithm is then used to obtain a partitioning of the variables into independent clusters. Affinity propagation is a clustering algorithm that, given a set of points and a measure of their similarity, finds clusters of similar points and for each cluster gives a representative example. Unlike many clustering algorithms, affinity propagation does not require specifying the total number of clusters beforehand. Once this model of independent clusters is generated by affinity propagation, it can be sampled similarly to ECGA. The resulting algorithm was shown to be much faster than ECGA on simplified protein folding problems and on deceptive non-binary problems [49].

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 [50] 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 is certainly possible to use non-permutation based EDAs using specific encodings to solve permutation problems. For example, one may use the *random key encoding* [51] to solve permutation-based problems using EDAs for optimization of real-valued vectors [52,53]. Random key encoding represents a permutation as a vector of real numbers. The permutation is defined by the reordering of the values in the vector that sorts the values in ascending order. The main advantage of using random keys is that any real-valued vector defines a valid permutation and any EDA capable of solving problems defined on vectors of real numbers can thus be used to solve permutation problems. However, since EDAs do not process the aforementioned types of regularities in permutation problems directly their performance can often be poor [52]. The following EDAs attempt to encode both of these types of features or constraints for permutation problems explicitly.

To solve problems where candidate solutions are permutations of a string, Bengoetxea et al. [54] starts with a Bayesian network model built using the same approach as in 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 et al. [55] 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 [55]. Both Bayesian networks as well as tree models are capable of encoding both the absolute position and the relative ordering constraints.

Bosman and Thierens [56] extended the real-valued EDA to the permutation domain by storing the dependences between different positions in a permutation in the induced chromosome element 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 the random keys encoding. 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 [56].

The edge histogram based sampling algorithm (EHBSA) [57] 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 [57]. 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 et al. [58] 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 be impractical 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 [59–63]. We discuss some of these next.

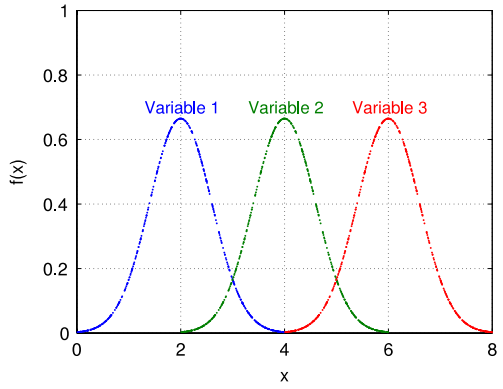


Fig. 6. Example model generated by SHCLVND.

Tsutsui et al. [59] 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 et al. [60]. 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 and Chen [63]. 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 [62] designed to test real-valued optimization techniques. In addition, rECGA was able to obtain better solutions than the previously best known solutions obtained by other algorithms on a 40-unit economic dispatch problem.

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) [64] 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. Fig. 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, SHCLVND assumes that all the variables are independent. The following algorithms all attempt to alleviate one or more of these problems.

Sebag and Ducoulombier [65] extended 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) [66] 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 [67], with a penalty term to prefer simpler models.

In the IDEA framework, Bosman and Thierens [68] proposed models capable of capturing multiple basins of attraction or clusters of points by storing the joint normal and kernel distributions. 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.

Gallagher et al. [69] extended PBIL to real-valued spaces by using an Adaptive Gaussian mixture model [70]. The Adaptive Gaussian mixture model used a mixture of Gaussian distributions that are gradually modified to improve the model as each new solution is sampled. One of the strengths of this method is that the complexity of the model can change over time, with additional components added if the current model does not correspond closely enough to the data [71].

The mixed iterated density estimation evolutionary algorithm (mIDEA) [72] also used 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 dependences between variables, the Bayesian information criterion (BIC) [73] 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) [74] 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 used for continuous variables.

The real-coded Bayesian optimization algorithm (rBOA) [75] 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.

Recently a new approach to developing EDAs to solve real-valued optimization problem has been developed that is based on copula theory [76]. Copulas are a way to describe the dependence between random variables, and according to copula theory a joint probability distribution can be decomposed into *n*

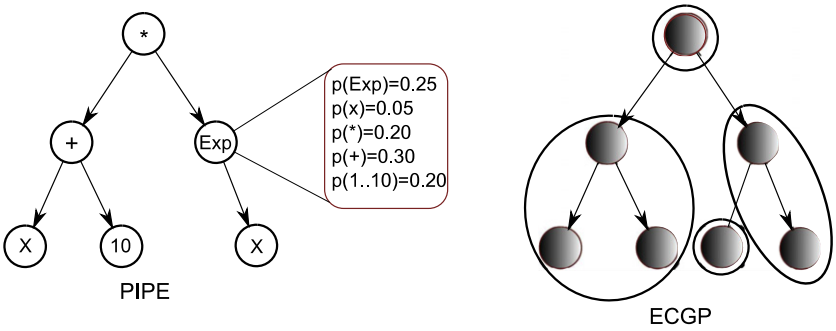


Fig. 7. Example of two different models used in EDA-GP.

marginal probability distributions and a copula function. Copula-based EDAs [77–79] use this to their advantage, as the marginal distributions and the dependences between variables can be studied separately. These EDAs start by calculating the marginal distribution of each variable separately. The algorithms then pick a particular copula to construct the joint distribution. Given the marginal distributions and a copula, it is then possible to generate new candidate solutions.

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) [80]. 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) [81] 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. Fig. 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 sampling 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) [82]. 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. Fig. 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) [83] 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) [84], 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) [85] 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) [86] 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) [87] metric is used to compare grammars. This algorithm is very adaptable, being able to generate a broad variety of possible grammars. However, comparing competing grammars is computationally expensive.

3.5. Multi-objective EDAs

Most EDAs discussed so far were designed to solve problems with one objective. However, many real-world problems contain competing objectives. For example, when optimizing a car engine, one may want to maximize power and, at the same time, minimize environmental impact and fuel consumption. One way to solve multi-objective problems is to transform the multiple objectives into a single objective by weighing the objectives in some way. However, it is often more desirable to find an optimal tradeoff between the objectives in the form of a diverse set of Pareto optimal solutions [88,89]. In short, a solution is Pareto optimal if it outperforms any other solution in at least one objective. In

this section, we review several EDAs that aim to find diverse sets of Pareto optimal solutions for multi-objective optimization problems.

For these types of problems, it is no longer possible to find one solution that maximizes all the goals simultaneously. Instead, the ultimate goal is to find a broad distribution of Pareto-optimal solutions.

The Bayesian multi-objective optimization algorithm (BMOA) [90] uses a special selection operator, ϵ-archive [91], to both ensure that Pareto-optimal solutions are maintained over time and that diversity is maintained so that an approximation of the entire Pareto set is preserved. The selection operator works by maintaining a minimal set of solutions that ϵ-dominates all other solutions generated so far, with ϵ being a problem specific parameter that stands for the relative tolerance allowed for different objective values. Results showed that the resulting algorithm was able to find a good model of the Pareto set for smaller instances of the 0/1 multi-objective knapsack problem. Essentially, BMOA combines the mixed BOA [74] and the improved strength Pareto evolutionary algorithm (SPEA2) [92].

The naive mixture-based multi-objective iterated density-estimation evolutionary algorithm (MIDEA) [93] extended the IDEA framework to multi-objective optimization. A special selection operator was used to ensure preservation of diversity along the Pareto front, guided by a single parameter δ. The population is then clustered using the leader algorithm [94], with the leader algorithm selected due its speed and the lack of any requirement to specify the number of clusters beforehand. A univariate model is built for each cluster and sampled to generate new candidate solutions.

The multi-objective Bayesian optimization algorithm (mBOA) [95] uses the selection operator from the non-dominated sorting algorithm-II (NSGA-II) [96] to maintain a diverse set of solutions along the Pareto-optimal front. This selection operator gives each solution in the population a rank and a crowding distance. The rank provides pressure to maximize all objectives whereas the crowding distance provides pressure to maintain diversity and broad coverage of the Pareto front. Solutions compete in binary tournaments. If the ranks of the two competing solutions differ, the winner of a tournament is the solution with a better rank; if the ranks are equal, the crowding distance is used to determine the winner. A Bayesian network model is then built for the selected solutions and the resulting network is sampled to generate new candidate solutions for the next generation. The multi-objective BOA outperformed the NSGA-II on several interleaved deceptive problems.

The multi-objective hierarchical BOA (mohBOA) [97] extended hBOA to the multi-objective domain by combining hBOA, NSGA-II [96] and clustering. mohBOA uses the non-dominated crowding of NSGA-II to rank candidate solutions and assign crowding distances. This information is then used to select promising solutions using the same procedure as in NSGA-II and mBOA. After selecting the promising candidate solutions, k-means clustering is used to obtain a clustering of the promising solutions. hBOA creates a separate Bayesian network model for each cluster and each created Bayesian network is used to generate the same number of new candidate solutions. New candidate solutions are incorporated into the old population using RTR to form the next generation's population. The results showed that mohBOA was able to solve the onemax–zeromax and trap5–invtrap5 in low-order polynomial time. Similar modifications were also tested in combination with marginal product models of ECGA in Ref. [98].

Shah and Reed [99] compared three different multi-objective evolutionary algorithms when solving the multi-objective d-dimensional knapsack problem. Their results showed that the ϵ-hBOA [16] was able to outperform both the strength Pareto evolutionary algorithm (SPEA2) [92] and the ϵ-NSGA-II [100]. The ϵ-hBOA uses the selection operator in NSGA-II to select promising candidate solutions each generation. hBOA is then used to generate a Bayesian network model and sampled to generate new candidate solutions. ϵ-nondominated archiving [91] is then used to form a new population from the candidate solutions, with this archive used as the population for subsequent generations. Essentially, ϵ-hBOA combines the operators of hBOA, NSGA-II and SPEA2

4. Related algorithms

All stochastic optimization algorithms guide their search for the optimum using probabilistic models. In most optimization algorithms, the models are defined *implicitly* by the current state of the search and the set of operators. On the other hand, in EDAs, explicit probabilistic models such as probability vectors and Bayesian networks are built from samples of high quality solutions and these models are then sampled to generate new candidate solutions. Nonetheless, EDAs and most other stochastic optimization techniques share many similarities, and this section will review some optimization methods that are most closely related to EDAs.

Evolutionary algorithms use operators of selection and variation to update a population of candidate solutions or a single candidate solution. For example, in genetic algorithms [101], binary tournament selection may be used to select promising solutions from the current population and the new population may be created by applying one-point crossover and bit-flip mutation to the selected parents. Selection and variation operators together with the current population of candidate solutions define the probability distribution over the populations of candidate solutions, and the new population of candidate solutions can be seen as a sample from that distribution. The main difference between most evolutionary algorithms and EDAs is that in EDAs the probability distribution used to generate new candidate solutions is defined *explicitly* whereas in most evolutionary algorithms the distribution is defined *implicitly*.

In some evolutionary algorithms, the distribution used to generate new candidate solutions is in fact defined explicitly, just like in EDAs. For example, in evolution strategies [102], new candidate solutions are often generated from a single normal distribution centered around the best-so-far candidate solution or from a mixture of normal distributions centered around a population of high-quality solutions found so far. Similarly, many ant colony optimization (ACO) methods [103] and particle swarm optimization methods (PSO) [104] use models that explicitly define a probability distribution over candidate solutions. From this perspective, many evolutionary algorithms may be viewed as "true" EDAs.

Developed independently of EDAs, the cross-entropy method (CE) [105] is probably the most closely related approach to EDAs and sees search much in the same way as EDAs. Similarly to EDAs, CE generates a random data set and then creates a model based on this random data set. Each iteration CE updates the model to increase model quality and ensure that it will generate better candidate solutions over time. Ideally, after a reasonable number of iterations, the model should generate the global optimum with high probability.

5. Advantages and disadvantages 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. Section 5.1 covers some of the important advantages that EDAs have over other metaheuristics. Section 5.2 then discusses some of the disadvantages that EDAs have when compared to other metaheuristics.

5.1. Advantages of EDAs

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 [5,7,19,106,107]. However, adaptation in EDAs is usually limited by the initial choice of the probabilistic model. As a consequence, adaptive EDAs [108,109] have been proposed that dynamically change the type of model used while solving a problem.

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 [110–113].

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 6.6.

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 et al. [27] shows that solving a 2^{25} (over 33 million) bit onemax problem with a simple genetic algorithm takes about 700 GB but the cGA described in 3.1.1 requires only a little over 128 MB.

5.2. Disadvantages of EDAs

Building explicit probabilistic models is often more time consuming than using implicit models defined with simple search operators, such as tournament selection and two-point crossover. That is why it may sometimes be advantageous to use implicit models of conventional evolutionary algorithms instead of explicit ones of EDAs. However, doing this is only practical when search operators are available that allow scalable solutions of the target problem class; otherwise, the time complexity of learning a model is a small price to pay. Furthermore, the discovery of such operators may not be straightforward and it also comes at a cost.

It is important to note that sometimes it is difficult to learn an adequate probabilistic model and in some cases it is possible to create problems that render some model building algorithms ineffective. For example, many EDAs (such as BOA and ECGA) use a greedy algorithm to build probabilistic models and it has been shown that there exist problems for which the greedy algorithm often leads to an inadequate model. Specifically, Coffin and Smith [114] proposed the concatenated parity function (CPF) for which pairwise correlations between variables cannot be easily detected using only marginal statistics. One can envision several ways to alleviate this problem, such as limited probing [115] and linkage identification by non-monotonicity detection [116]. However, it is questionable whether the ability to solve such a special class of problems will outweigh the disadvantages of giving up the use of Bayesian statistics in learning the probabilistic models. Furthermore, it was shown [117] that the difficulties of EDAs when solving CPF are mainly due to spurious linkages; therefore, using methods to reduce spurious linkages may provide yet another solution to this problem.

6. 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) [19,17,42] 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 [42]:

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.

6.1. 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 [118]. 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 [27]. Several approaches to parallelizing model building in advanced EDAs with multivariate models have also been proposed [119–121].

6.2. 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 [19,122–125].

There are two basic approaches to evaluation relaxation: (1) endogenous models [122–125] and (2) exogenous models [126,127]. 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.

Sastry et al. [128] incorporated endogenous models in 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 [125] and BOA [124] can accurately approximate even complex fitness functions due to the additional information about the problem structure encoded in the EDA model, yielding speedups of several orders of magnitude even for only moderately sized problems [124]. This type of information is not available at all to other types of metaheuristics.

6.3. 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 [129].

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) [130], which can be incorporated into hBOA [5] 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 [131] 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 et al. [132]. In this work, substructural neighborhoods were defined as a variable and all its parents in the Bayesian network model were 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 et al. [132] also discussed other possible neighborhood structures that could be extracted from Bayesian networks. In a similar approach, Handa [133] 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.

To perform advanced local search based on Bayesian networks, Mendiburu et al. [134] proposed the use of loopy belief propagation [135] to generate the most likely instance from the Bayesian network learned in each iteration of EBNA. A similar approach has later been studied by [136], who also used loopy belief propagation but instead of conditional probabilities, the loopy belief propagation was driven by fitness statistics incorporated into the model in hBOA. Related approaches have also been studied in the context of other probabilistic models [137–141].

6.4. 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 [142,143,19]. 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 [144] 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.

6.5. 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 [145,42]. 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 in a while [146].

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* [39].

6.6. 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 [113,147,148] and (2) bias or restrict the model building procedure [113,149,112]. For both these approaches, we may either (1) hard code the modifications based on prior problem-specific knowledge [111–113] or (2) develop automated procedures to improve EDA performance by learning from previous EDA runs on problems of similar type (learning from experience) [110].

One technique used to bias the initial population towards good solutions (and, consequently, to also improve model quality) is called *seeding* [113,148,147]. 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 [113,148], or created in some way from high-quality solutions of smaller instances of the same problem [147].

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 [113], 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 [149] also considered graph bipartitioning but in this case only allowed edges in the underlying graph. Baluja [112] also only allowed edges in the underlying graph in his work on graph coloring.

To develop a method that was more broadly applicable, Hauschild et al. [111] 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 [110] to bias the BD metric itself based on the probability that an edge was used in the previous runs.

7. EDA theory

While most of this paper has focused on the practical applications of EDAs in solving optimization problems, it is of equal importance that these algorithms have a strong theoretical background. The stronger our theoretical understanding about how these algorithms work, the easier it should be to develop new algorithms and efficiency enhancements, and to successfully apply these algorithms to new problems. In this section we will review some of the most important theoretical results in the field of EDAs. We divide these theoretical results into five main categories: (1) convergence proofs, (2) population sizing models, (3) diversity loss, (4) memory complexity, and (5) model accuracy.

7.1. Convergence proofs

Some of the most important theoretical results in EDA theory focus on the conditions that allow EDAs to provably converge to a global optimum. The convergence of the factorized distribution algorithm (FDA) on separable additively decomposable functions (ADFs) was explored by Muhlenbein and Mahnig [150], who developed an exact formula for convergence time when using proportional selection. Since in practice proportional selection is rarely used, truncation selection was also examined and an equation was derived giving the approximate time to convergence from an analysis of the onemax function.

In the aforementioned study [150], no overlap between the subfunctions in an additive decomposition of the objective function was assumed. In [151], the authors studied the convergence of FDA on ADFs where subproblems were allowed to interact. In this work, an infinite population was used and the effects of three different selection schemes on convergence were examined. The results showed that EDAs converged under all selection schemes examined as long as the probability distribution represented by the population after sampling the model was identical to the probability distribution of the population after selection (which was used to build the model). The authors also defined sufficient conditions for provable convergence of multivariate EDAs on ADFs, although these conditions are often not practical.

Zhang [152] examined two different EDAs, the FDA which uses higher-order statistics, and the UMDA using only first-order statistics. After developing limit models of UMDA and FDA, the authors show that in the case of a general objective function, the limit model of FDA has a better chance of obtaining the global optimum than UMDA. The authors then proved this on an additively decomposable objective function, showing that at least for some problems the chance of converging to the global optimum is indeed increased by using higher order statistics.

7.2. Population sizing

The convergence proofs mentioned in the previous subsection assumed infinite populations in order to simplify calculations. However, in practice using an infinite population is not possible. The population size in EDAs is closely related to the reliability and complexity of the search, similarly as for other population-based evolutionary algorithms [153,154,18]. Using a population that is too small can lead to convergence to solutions of low quality and inability to reliably find the global optimum. On the other hand, using a population that is too large can lead to an increased complexity of building and sampling probabilistic models, evaluating populations, and executing other EDA components. That is why choosing an adequate population size is crucial. Similar to GAs, EDAs must have a sufficient population size such that the initial supply of raw building blocks [153] is sufficient and that good decisions are made between competing partial solutions [153]. However, the population must be large enough for EDAs to make good decisions on variable interactions.

To examine this topic, Pelikan et al. [155] explored the population size required for BOA to solve decomposable problems of bounded difficulty with uniformly and nonuniformly scaled subproblems. The results showed that the population sizes required grew nearly linearly. The results also showed that the approximate number of evaluations grew sub-quadratically for uniformly scaled subproblems but was quadratic on some nonuniformly scaled subproblems.

Yu et al. [156] refined the model of Pelikan et al. [155] to provide more accurate bounds for the adequate population size in multivariate EDAs and also examined the effects of selection pressure on population size. The work focused on entropy-based EDAs and started by assuming an infinite population. Then, for a finite population size, the distributions of the sampled entropy was investigated.

7.3. Diversity loss

It is possible for stochastic errors in population sampling to lead to a loss of diversity. If this loss of diversity continues over time (by producing simplified models), it is possible for the population to no longer contain enough information to solve the problem. Shapiro [157] examined the susceptibility of UMDA to diversity loss and discussed how it is necessary to set the learning parameters in such a way that this does not happen.

Bosman et al. [158] examined diversity loss in EDAs for solving real-valued problems and the approaches to alleviating this difficulty. The results showed that due to diversity loss some of the state-of-the-art EDAs for real-valued problems could still fail on slope-like regions in the search space. The authors proposed using anticipated mean shift (AMS) to shift the mean of new solutions each generation in order to effectively maintain diversity.

7.4. Memory complexity

Another factor of importance in EDA problem solving is the memory required to solve the problem. Gao and Culberson [159] examined the space complexity of the FDA and BOA on additively decomposable functions where overlap was allowed between subfunctions. Gao and Culberson [159] proved that the space complexity of FDA and BOA is exponential in the problem size even with very sparse interaction between variables. While these results are somewhat negative, the authors point out that this only shows that EDAs have limitations and work best when the interaction structure is of bounded size.

One way to reduce the memory complexity required in multivariate EDAs is to use incremental EDAs. The incremental BOA (iBOA) [160] starts with a simple univariate model that is gradually increased in complexity. Each generation, several solutions are generated and the best and worst fit individuals are used to determine whether to increase the complexity of the model by adding an edge between nodes, with the parameters of the model also slowly updated over time. To limit the required memory complexity, iBOA stores only the marginal probabilities for the current model and any that would be required after adding an edge between nodes. The resulting multivariate EDA has less memory complexity than BOA but was still able to scalably solve deceptive trap problems.

7.5. Model accuracy

Model accuracy studies examine the accuracy of the models in EDAs that successfully solve a problem. By understanding the structure and complexity of the models and how they relate to the underlying problem structure, researchers should be able to develop new theoretical models and design new efficiency enhancement techniques.

Hauschild et al. [161] analyzed the models generated by hBOA when solving concatenated traps, random additively decomposable problems, hierarchical traps and 2D Ising spin glasses. The models generated were then compared to the underlying problem structure by comparing the number of spurious and correct edges added (given a perfect model). The results showed that the models did closely correspond to the structure of the underlying problems and that the models did not change significantly between consequent iterations of hBOA.

The relationship between the probabilistic models learned by BOA and the underlying problem structure was also explored by Lima et al. [162]. The accuracy of the models was defined as the ratio of the edges found in a perfect model over the total number of edges in the network. One of the most important contributions of this study was to demonstrate the dramatic effect that selection has on spurious dependences. The results showed that model accuracy was significantly improved when using truncation selection compared to tournament selection. Motivated by these results, the authors modified the complexity penalty of BOA model building to take into account tournament sizes when using binary tournament selection. This new s-penalty was found to significantly improve the model structural accuracy of BOA even when tournament selection is used.

Echegoyen et al. [163] also analyzed the structural accuracy of the models, this time using EBNA on concatenated traps, two variants of Ising spin glass and MAX-SAT. In this work two variations of EBNA were compared, one that was given the complete model structure based on the underlying problem and another that learned the approximate structure. The authors then examined the probability at any generation that the models would generate the optimal solution. The results showed that it was not strictly necessary to have all the interactions that were in the complete model in order to solve the problems. It was also discovered that in order for the algorithm to reach a solution, the probability of an optimal solution must always exceed a certain threshold.

Finally, the effects of spurious linkages on EDA performance were examined by Radetic and Pelikan [164]. The authors started by proposing a theoretical model to describe the effects of spurious (unnecessary) dependences on the population sizing of EDAs. This model was then tested empirically on onemax and the results showed that while it would be expected that spurious dependences would have little effect on population size, when niching was included the effects were substantial.

8. Additional information

This section provides pointers to additional sources of information on EDAs.

8.1. Software

The following list contains some free EDA implementations available online:

- *Implementations of BOA, hBOA and dtEDA*: http://medal.cs.umsl.edu/software.php
- *Implementation of ECGA in Matlab and C++*: http://illigal.org/category/source-code/
- *Implementations of sequential and parallel mixed BOA and adaptive mixed BOA*: http://jiri.ocenasek.com/#Download
- *Matlab toolbox for EDAs and several implementations*: http://www.sc.ehu.es/ccwbayes/members/rsantana/software/Software.html
- *Implementation of the Adapted Maximum-Likelihood Gaussian Model Iterated Density Estimation Evolutionary Algorithm (AMaLGaM-IDEA)*: http://homepages.cwi.nl/~bosman/source_code.php
- *Source code for RM-MEDA and several other EDAs and test problems*: http://cswww.essex.ac.uk/staff/qzhang/mypublication.htm
- *Implementations of Real-coded BOA and multi-objective Real-coded BOA*: http://www.evolution.re.kr/

8.2. Journals

The following journals are key venues for papers in EDAs and often contain papers on the cutting edge of research in EDAs:

- *Evolutionary Computation (MIT Press)*: http://www.mitpressjournals.org/loi/evco
- *IEEE Transactions on Evolutionary Computation (IEEE Press)*: http://www.ieee-cis.org/pubs/tec/
- *Swarm and Evolutionary Computation*: http://www.elsevier.com/wps/find/journaldescription.cws_home/724666/description#description
- *Genetic Programming and Evolvable Machines*: http://www.springer.com/computer/ai/journal/10710

8.3. Conferences

The following conferences are the primary ones that publish papers in the EDA field:

- *ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO)*:
- *Parallel Problem Solving in Nature (PPSN)*:
- *IEEE Congress on Evolutionary Computation (CEC)*:

9. 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 practitioners 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

[1] S. Baluja, Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning, Tech. Rep. No. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.
[2] H. Mühlenbein, G. Paaß, From recombination of genes to the estimation of distributions I. Binary parameters, in: Parallel Problem Solving from Nature, 1996, pp. 178–187.
[3] P. Larrañaga, J.A. Lozano (Eds.), Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Kluwer, Boston, MA, 2002.
[4] M. Pelikan, D.E. Goldberg, F. Lobo, A survey of optimization by building and using probabilistic models, Computational Optimization and Applications 21 (2002) 5–20.
[5] M. Pelikan, A.K. Hartmann, Searching for ground states of Ising spin glasses with hierarchical BOA and cluster exact approximation, in: M. Pelikan, K. Sastry, E. Cantú-Paz (Eds.), Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, Springer, 2006, pp. 333–349.
[6] T.-L. Yu, S. Santarelli, D.E. Goldberg, Military antenna design using a simple genetic algorithm and hBOA, in: M. Pelikan, K. Sastry, E. Cantú-Paz (Eds.), Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, Springer, 2006, pp. 275–289.
[7] R. Shah, P. Reed, Comparative analysis of multiobjective evolutionary algorithms for random and correlated instances of multiobjective d-dimensional knapsack problems, European Journal of Operational Research 211 (3) (2011) 466–479.
[8] R. Arst, B.S. Minsker, D.E. Goldberg, Comparing advanced genetic algorithms and simple genetic algorithms for groundwater management, in: Proceedings of the American Society of Civil Engineers, ASCE, Environmental & Water Resources Institute, EWRI, 2002 Water Resources Planning & Management Conference, Roanoke, VA, 2002.
[9] M.S. Hayes, B.S. Minsker, Evaluation of advanced genetic algorithms applied to groundwater remediation design, in: Proceedings of the American Society of Civil Engineers, ASCE, Environmental & Water Resources Institute, EWRI, World Water & Environmental Resources Congress 2005 & Related Symposia, Anchorage, AK, 2005.
[10] J. Bacardit, M. Stout, J.D. Hirst, K. Sastry, X. Llorà, N. Krasnogor, Automated alphabet reduction method with evolutionary algorithms for protein structure prediction, in: Genetic and Evolutionary Computation Conference, GECCO-2007, 2007, pp. 346–353.
[11] J. Peña, J. Lozano, P. Larrañaga, 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 (2004) 63–82.
[12] C.-H. Chen, Y.p. Chen, Real-coded ECGA for economic dispatch, in: Genetic and Evolutionary Computation Conference, GECCO-2007, 2007, pp. 1920–1927.
[13] E. Ducheyne, B. De Baets, R. De Wulf, Probabilistic models for linkage learning in forest management, in: Y. Jin (Ed.), Knowledge Incorporation in Evolutionary Computation, Springer, 2004, pp. 177–194.
[14] P. Lipinski, ECGA vs. BOA in discovering stock market trading experts, in: Genetic and Evolutionary Computation Conference, GECCO-2007, 2007, pp. 531–538.
[15] A. Petrovski, S. Shakya, J. Mccall, Optimising cancer chemotherapy using an estimation of distribution algorithm and genetic algorithms, in: Genetic and Evolutionary Computation Conference, GECCO-2006, 2006, pp. 413–418.
[16] J.B. Kollat, P.M. Reed, J.R. Kasprzyk, A new epsilon-dominance hierarchical Bayesian optimization algorithm for large multi-objective monitoring network design problems, Advances in Water Resources 31 (2008) 828–845.
[17] K. Sastry, M. Pelikan, D.E. Goldberg, Efficiency enhancement of estimation of distribution algorithms, in: M. Pelikan, K. Sastry, E. Cantú-Paz (Eds.), Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, Springer, 2006, pp. 161–185.
[18] G.R. Harik, E. Cantú-Paz, D.E. Goldberg, B.L. Miller, The gambler's ruin problem, genetic algorithms, and the sizing of populations, in: International Conference on Evolutionary Computation, ICEC-97, IEEE Press, Piscataway, NJ, 1997, pp. 7–12.
[19] D.E. Goldberg, The Design of Innovation: Lessons From and for Competent Genetic Algorithms, Kluwer, 2002.
[20] D.H. Ackley, An empirical study of bit vector function optimization, in: Genetic Algorithms and Simulated Annealing, 1987, pp. 170–204.
[21] K. Deb, D.E. Goldberg, Analyzing deception in trap functions, IlliGAL, Report No. 91009, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1991.
[22] D. Thierens, D.E. Goldberg, Mixing in genetic algorithms, in: International Conference on Genetic Algorithms, ICGA-93, 1993, pp. 38–45.
[23] G. Harik, Linkage learning via probabilistic modeling in the ECGA, IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1999.
[24] H. Mühlenbein, T. Mahnig, FDA—a scalable evolutionary algorithm for the optimization of additively decomposed functions, Evolutionary Computation 7 (1999) 353–376.
[25] H. Mühlenbein, G. Paaß, From recombination of genes to the estimation of distributions I. Binary parameters, in: A. Eiben, T. Bäck, M. Shoenauer, H. Schwefel (Eds.), Parallel Problem Solving from Nature, Springer Verlag, Berlin, 1996, pp. 178–187.
[26] G.R. Harik, F.G. Lobo, D.E. Goldberg, The compact genetic algorithm, IlliGAL Report No. 97006, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1997.
[27] K. Sastry, D.E. Goldberg, X. Llorà, Towards billion bit optimization via parallel estimation of distribution algorithm, in: Genetic and Evolutionary Computation Conference, GECCO-2007, 2007, pp. 577–584.
[28] J.S. De Bonet, C.L. Isbell, P. Viola, MIMIC: finding optima by estimating probability densities, Advances in Neural Information Processing Systems 9 (1997) 424–431. (NIPS-97).
[29] S. Baluja, S. Davies, Using optimal dependency-trees for combinatorial optimization: learning the structure of the search space, in: Proceedings of the International Conference on Machine Learning, 1997, pp. 30–38.
[30] C. Chow, C. Liu, Approximating discrete probability distributions with dependence trees, IEEE Transactions on Information Theory 14 (1968) 462–467.

[31] M. Pelikan, H. Mühlenbein, The bivariate marginal distribution algorithm, in: Advances in Soft Computing—Engineering Design and Manufacturing, 1999, pp. 521–535.
[32] L.A. Marascuilo, M. McSweeney, Nonparametric and Distribution-Free Methods for the Social Sciences, Brooks/Cole Publishing Company, CA, 1977.
[33] P.A.N. Bosman, D. Thierens, Linkage information processing in distribution estimation algorithms, in: Genetic and Evolutionary Computation Conference, GECCO-99, vol. I, 1999, pp. 60–67.
[34] T.M. Mitchell, Machine Learning, McGraw-Hill, 1997.
[35] M. Pelikan, D.E. Goldberg, E. Cantú-Paz, Linkage problem, distribution estimation, and Bayesian networks, Evolutionary Computation 8 (2000) 311–341. Also IlliGAL Report No. 98013.
[36] D. Heckerman, D. Geiger, D.M. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, Machine Learning 20 (1995) 197–243. doi:10.1007/BF00994016.
[37] D. Heckerman, D. Geiger, D.M. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, Technical Report MSR-TR-94-09, Microsoft Research, Redmond, WA, 1994.
[38] M. Henrion, Propagating uncertainty in Bayesian networks by probabilistic logic sampling, in: J.F. Lemmer, L.N. Kanal (Eds.), Uncertainty in Artificial Intelligence, Elsevier, Amsterdam, London, New York, 1988, pp. 149–163.
[39] R. Etxeberria, P. Larrañaga, Global optimization using Bayesian networks, in: A. Ochoa, M.R. Soto, R. Santana (Eds.), Proceedings of the Second Symposium on Artificial Intelligence, CIMAF-99, Editorial Academia, Havana, Cuba, 1999, pp. 151–173.
[40] G. Schwarz, Estimating the dimension of a model, The Annals of Statistics 6 (1978) 461–464.
[41] H.A. Simon, The Sciences of the Artificial, The MIT Press, Cambridge, MA, 1968.
[42] M. Pelikan, Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms, Springer-Verlag, 2005.
[43] D.M. Chickering, D. Heckerman, C. Meek, A Bayesian approach to learning Bayesian networks with local structure, Technical Report MSR-TR-97-07, Microsoft Research, Redmond, WA, 1997.
[44] N. Friedman, M. Goldszmidt, Learning Bayesian networks with local structure, in: M.I. Jordan (Ed.), Graphical Models, MIT Press, 1999, pp. 421–459.
[45] G.R. Harik, Finding multimodal solutions using restricted tournament selection, in: International Conference on Genetic Algorithms, ICGA-95, 1995, pp. 24–31.
[46] M. Pelikan, D.E. Goldberg, Hierarchical Bayesian optimization algorithm, in: M. Pelikan, K. Sastry, E. Cantú-Paz (Eds.), Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, Springer, 2006, pp. 63–90.
[47] H. Mühlenbein, Convergence of estimation of distribution algorithms for finite samples, Technical Report, Fraunhofer Institut Autonomous intelligent Systems, Sankt Augustin, Germany, 2008.
[48] S. Shakya, R. Santana, An EDA based on local Markov property and Gibbs sampling, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO'08, ACM, New York, NY, USA, 2008, pp. 475–476.
[49] R. Santana, P. Larrañaga, J.A. Lozano, Learning factorizations in estimation of distribution algorithms using affinity propagation, Evolutionary Computation 18 (2010) 515–546.
[50] T.C. Koopmans, M.J. Beckmann, Assignment problems and the location of economic activities, Econometrica 25 (1957) 53–76.
[51] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA Journal on Computing 6 (1994) 154–160.
[52] P.A. Bosman, D. Thierens, Crossing the road to efficient ideas for permutation problems, in: Genetic and Evolutionary Computation Conference, GECCO-2001, 2001, pp. 219–226.
[53] V. Robles, P. de Miguel, P. Larrañaga, Solving the traveling salesman problem with EDAs, in: P. Larrañaga, J.A. Lozano (Eds.), Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation, Kluwer Academic Publishers, Boston, Dordrecht, London, 2002, pp. 227–238.
[54] E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, C. Boeres, Inexact graph matching using learning and simulation of Bayesian networks. An empirical comparison between different approaches with synthetic data, in: Proceedings of CaNew Workshop, ECAI 2000 Conference, Berlin.
[55] M. Pelikan, S. Tsutsui, R. Kalapala, Dependency trees, permutations, and quadratic assignment problem, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO'07, ACM, New York, NY, USA, 2007, 629–629.
[56] P.A. Bosman, D. Thierens, New IDEAs and more ICE by learning and using unconditional permutation factorizations, in: Late-Breaking Papers of the Genetic and Evolutionary Computation Conference, GECCO-2001, 2001, pp. 16–23.
[57] S. Tsutsui, 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, Springer-Velag, 2002, pp. 224–233.
[58] S. Tsutsui, M. Pelikan, D.E. Goldberg, Node histogram vs. edge fistogram: 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, 2006.
[59] S. Tsutsui, M. Pelikan, D.E. Goldberg, Evolutionary algorithm using marginal histogram models in continuous domain, in: Workshop Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001, 2001, pp. 230–233.
[60] M. Pelikan, D.E. Goldberg, S. Tsutsui, Getting the best of both worlds: discrete and continuous genetic and evolutionary algorithms in concert, Information Sciences 156 (2003) 147–171. Evolutionary Computation.
[61] C. Chen, W. Liu, Y. Chen, Adaptive discretization for probabilistic model building genetic algorithms, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO'06, ACM, New York, NY, USA, 2006, pp. 1103–1110.
[62] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real parameter optimization, Technical Report, Nanyang Technological University, 2005.
[63] Y. Chen, C. Chen, Enabling the extended compact genetic algorithm for real-parameter optimization by using adaptive discretization, Evolutionary Computation 18 (2010) 199–228.
[64] S. Rudlof, M. Köppen, Stochastic hill climbing with learning by vectors of normal distributions, in: First on-line Workshop on Soft Computing, Nagoya, Japan, pp. 60–70.
[65] M. Sebag, A. Ducoulombier, Extending population-based incremental learning to continuous search spaces, in: Parallel Problem Solving from Nature, Springer Verlag, Berlin, Heidelberg, 1998, pp. 418–427.
[66] P. Larrañaga, R. Etxeberria, J.A. Lozano, J.M. Peña, Optimization by learning and simulation of Bayesian and Gaussian networks, Technical Report, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 1999.
[67] D. Geiger, D. Heckerman, Learning Gaussian networks, in: Proceedings of UAI'1994, pp. 235–243.
[68] P.A.N. Bosman, D. Thierens, Continuous iterated density estimation evolutionary algorithms within the IDEA framework, in: Workshop Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2000, 2000, pp. 197–200.
[69] M. Gallagher, M. Frean, T. Downs, Real-valued evolutionary optimization using a flexible probability density estimator, in: Proceedings of the Genetic and Evolutionary Computation Conference, Citeseer, pp. 840–846.
[70] C.E. Priebe, Adaptive mixtures, Journal of the American Statistical Association 89 (1994) 796–806.
[71] M. Gallagher, Multi-layer perceptron error surfaces: visualization, structure and modelling, Ph.D. Thesis, 2000.
[72] P.A. Bosman, D. Thierens, 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 GECCO-2001, Morgan Kaufmann, 2001, pp. 208–212.
[73] G. Schwarz, Estimating the dimension of a model, Annals of Statistics 6 (1978) 461–464.
[74] J. Ocenasek, S. Kern, N. Hansen, P. Koumoutsakos, A mixed Bayesian optimization algorithm with variance adaptation, in: X. Yao, E. Burke, J.A. Lozano, J. Smith, J.J. Merelo-Guervós, J.A. Bullinaria, J. Rowe, P. Tino, A. Kabán, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature—PPSN VIII, in: Lecture Notes in Computer Science, vol. 3242, Springer, Berlin, Heidelberg, 2004, pp. 352–361.
[75] C.W. Ahn, R.S. Ramakrishna, D.E. Goldberg, Real-coded Bayesian optimization algorithm: bringing the strength of BOA into the continuous world, in: K. Deb (Ed.), Proceedings of the Genetic and Evolutionary Computation Conference—GECCO 2004, Springer-Verlag, Berlin, 2004, pp. 840–851.
[76] R.B. Nelsen, An Introduction to Copulas, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.
[77] R. Salinas-Gutiérrez, A. Hernández-Aguirre, E.R. Villa-Diharce, Using copulas in estimation of distribution algorithms, in: MICAI 2009: Advances in Artificial Intelligence, in: Lecture Notes in Computer Science, vol. 5845, Springer, Guanajato, Mexico, 2009, pp. 658–668.
[78] L.-F. Wang, J.-C. Zeng, Y. Hong, Estimation of distribution algorithm based on Archimedean copulas, in: Proceedings of the International Conference GEC'2009, ACM, Shangai, China, 2009, pp. 993–996.
[79] L.-F. Wang, J.-C. Zeng, Y. Hong, Estimation of distribution algorithm based on copula theory, in: Proceedings of the 2009 Congress on Evolutionary Computation, CEC-2009, IEEE Press, Norway, 2009, pp. 1057–1063.
[80] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, The MIT Press, Cambridge, MA, 1992.
[81] R.P. Salustowicz, J. Schmidhuber, Probabilistic incremental program evolution: stochastic search through program space, in: Proceedings of the European Conference of Machine Learning, ECML-97, vol. 1224, 1997, pp. 213–220.
[82] K. Sastry, D.E. Goldberg, On extended compact genetic algorithm, IlliGAL Report No. 2000026, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2000.
[83] M. Looks, Competent program evolution, Doctor of Science, Washington University, St. Louis, USA, 2006.
[84] A. Ratle, M. Sebag, Avoiding the bloat with stochastic grammar-based genetic programming, 2006, CoRR..
[85] Y. Shan, R.I. Mckay, H.A. Abbass, D. Essam, Program evolution with explicit learning: a new framework for program automatic synthesis, in: University College, University of New South, IEEE Press, 2003, pp. 1639–1646.

[86] Y. Shan, R.I. Mckay, R. Baxter, Grammar model-based program evolution, in: In Proceedings of the 2004 IEEE Congress on Evolutionary Computation, IEEE Press, 2004, pp. 478–485.
[87] C.S. Wallace, D.M. Boulton, An information measure for classification, Computer Journal 11 (1968) 185–194.
[88] K. Deb, Multi-Objective Optimization Using Evolutionary Algorithms, John Wiley & Sons, Chichester, UK, 2001.
[89] C. Coello, G. Lamont, Applications of Multi-Objective Evolutionary Algorithms, in: Advances in Natural Computation, World Scientific, 2004.
[90] M. Laumanns, J. Ocenasek, Bayesian optimization algorithms for multi-objective optimization, in: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, in: PPSN, vol. VII, Springer-Verlag, London, UK, UK, 2002, pp. 298–307.
[91] M. Laumanns, L. Thiele, K. Deb, E. Zitzler, Combining convergence and diversity in evolutionary multi-objective optimization, Evolutionary Computation 10 (2002) 263–282.
[92] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: improving the strength Pareto evolutionary algorithm for multiobjective optimization, in: K. Giannakoglou, et al. (Eds.), Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems, EUROGEN 2001, International Center for Numerical Methods in Engineering, CIMNE, 2002, pp. 95–100.
[93] P. Bosman, D. Thierens, Multi-objective optimization with the naive MIDEA, in: J. Lozano, P. Larrañaga, I. Inza, E. Bengoetxea (Eds.), Towards a New Evolutionary Computation, in: Studies in Fuzziness and Soft Computing, vol. 192, Springer, Berlin, Heidelberg, 2006, pp. 123–157.
[94] J.A. Hartigan, Clustering Algorithms, John Wiley and Sons, Inc., 1975.
[95] N. Khan, D.E. Goldberg, M. Pelikan, Multi-objective Bayesian optimization algorithm, IlliGAL Report No. 2002009, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2002.
[96] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2002) 182–197.
[97] M. Pelikan, K. Sastry, D.E. Goldberg, Multiobjective hBOA, clustering, and scalability, in: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, GECCO'05, ACM, New York, NY, USA, 2005, pp. 663–670.
[98] M. Pelikan, K. Sastry, D.E. Goldberg, Multiobjective estimation of distribution algorithms, in: Scalable Optimization via Probabilistic Modeling, 2006, pp. 223–248.
[99] R. Shah, P. Reed, Comparative analysis of multiobjective evolutionary algorithms for random and correlated instances of multiobjective d-dimensional knapsack problems, European Journal of Operational Research 211 (2011) 466–479.
[100] J.B. Kollat, P.M. Reed, The value of online adaptive search: a performance comparison of NSGA-II, ϵ-NSGAII, and ϵ-MOEA, in: The Third International Conference on Evolutionary Multi-Criterion Optimization, EMO 2005, in: Lecture Notes in Computer Science, Springer-Verlag, Guanajuato, Mexico, 2005, pp. 386–398.
[101] J. Holland, Genetic algorithms and the optimal allocation of trials, SIAM Journal of Computing 2 (1973) 88–105.
[102] I. Rechenberg, Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution, Frommann-Holzboog, Stuttgart, 1973.
[103] M. Dorigo, Optimization, learning and natural algorithms, Ph.D. Thesis, Politecnico di Milano, Italie, 1992.
[104] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Neural Networks, Proceedings, IEEE International Conference on, vol. 4, 1995, pp. 1942–1948.
[105] R.Y. Rubinstein, Optimization of computer simulation models with rare events, European Journal of Operational Research 99 (1996) 89–112.
[106] Q. Zhang, J. Sun, E. Tsang, Evolutionary algorithm with the guided mutation for the maximum clique problem, IEEE Transactions on Evolutionary Computation 9 (2005) 192–200.
[107] Q. Zhang, A. Zhou, Y. Jin, RM-MEDA: a regularity model-based multiobjective estimation of distribution algorithm, IEEE Transactions on Evolutionary Computation 12 (2008) 41–63.
[108] R. Santana, P. Larrañaga, J. Lozano, Adaptive estimation of distribution algorithms, in: C. Cotta, M. Sevaux, K. Sõrensen (Eds.), Adaptive and Multilevel Metaheuristics, in: Studies in Computational Intelligence, vol. 136, Springer, Berlin, Heidelberg, 2008, pp. 177–197.
[109] F.G. Lobo, C.F. Lima, Towards automated selection of estimation of distribution algorithms, in: Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO-2010, ACM, 2010, pp. 1945–1952.
[110] M.W. Hauschild, M. Pelikan, Intelligent bias of network structures in the hierarchical BOA, in: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO'09, ACM, New York, NY, USA, 2009, pp. 413–420.
[111] M. Hauschild, M. Pelikan, K. Sastry, D.E. Goldberg, Using previous models to bias structural learning in the hierarchical BOA, in: Genetic and Evolutionary Computation Conference, GECCO-2008, 2008, pp. 415–422.
[112] S. Baluja, Incorporating a priori knowledge in probabilistic-model based optimization, in: M. Pelikan, K. Sastry, E. Cantú-Paz (Eds.), Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, Springer, 2006, pp. 205–219.
[113] J. Schwarz, J. Ocenasek, A problem-knowledge based evolutionary algorithm KBOA for hypergraph partitioning, Personal Communication, 2000.
[114] D.J. Coffin, R.E. Smith, Why is parity hard for estimation of distribution algorithms? in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO'07, ACM, New York, NY, USA, 2007, 624–624.
[115] R.B. Heckendorn, A.H. Wright, Efficient linkage discovery by limited probing, Evolution Computing 12 (2004) 517–545.
[116] M. Munetomo, D.E. Goldberg, Linkage identification by non-monotonicity detection for overlapping functions, Evolutionary Computation 7 (1999) 377–398.
[117] S.-C. Chen, T.-L. Yu, Difficulty of linkage learning in estimation of distribution algorithms, in: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO'09, ACM, New York, NY, USA, 2009, pp. 397–404.
[118] E. Cantú-Paz, Efficient and Accurate Parallel Genetic Algorithms, Kluwer, Boston, MA, 2000.
[119] J. Ocenasek, Parallel estimation of distribution algorithms, Ph.D. Thesis, Faculty of Information Technology, Brno University of Technology, Brno, 2002.
[120] J. Ocenasek, E. Cantú-Paz, M. Pelikan, J. Schwarz, Design of parallel estimation of distribution algorithms, in: M. Pelikan, K. Sastry, E. Cantú-Paz (Eds.), Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, Springer, 2006, pp. 187–203.
[121] A. Mendiburu, J. Lozano, J. Miguel-Alonso, Parallel implementation of EDAs based on probabilistic graphical models, IEEE Transactions on Evolutionary Computation 9 (2005) 406–423.
[122] R.E. Smith, B.A. Dike, S.A. Stegmann, Fitness inheritance in genetic algorithms, in: Proceedings of the ACM Symposium on Applied Computing, 1995, pp. 345–350.
[123] K. Sastry, D.E. Goldberg, M. Pelikan, Don't evaluate, inherit, in: Genetic and Evolutionary Computation Conference, GECCO-2001, 2001, pp. 551–558.
[124] M. Pelikan, K. Sastry, Fitness inheritance in the Bayesian optimization algorithm, in: Genetic and Evolutionary Computation Conference, GECCO-2004, vol. 2, 2004, pp. 48–59.
[125] K. Sastry, M. Pelikan, D.E. Goldberg, Efficiency enhancement of genetic algorithms via building-block-wise fitness estimation, in: Proceedings of the IEEE International Conference on Evolutionary Computation, 2004, pp. 720–727.
[126] K. Sastry, Evaluation-relaxation schemes for genetic and evolutionary algorithms, Master's Thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL, 2001.
[127] L.A. Albert, Efficient genetic algorithms using discretization scheduling, Master's Thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL, 2001.
[128] K. Sastry, D.E. Goldberg, M. Pelikan, Don't evaluate, inherit, in: L. Spector, E.D. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M.H. Garzon, E. Burke (Eds.), Genetic and Evolutionary Computation Conference, GECCO-2001, Morgan Kaufmann, San Francisco, CA, 2001, pp. 551–558.
[129] W.E. Hart, Adaptive global optimization with local search, Ph.D. Thesis, University of California, San Diego, San Diego, CA, 1994.
[130] A.K. Hartmann, Cluster-exact approximation of spin glass ground states, Physica A 224 (1996) 480.
[131] K. Sastry, D.E. Goldberg, Designing competent mutation operators via probabilistic model building of neighborhoods, in: Genetic and Evolutionary Computation Conference, GECCO-2004, 2004, pp. 114–125.
[132] C.F. Lima, M. Pelikan, K. Sastry, M.V. Butz, D.E. Goldberg, F.G. Lobo, Substructural neighborhoods for local search in the Bayesian optimization algorithm, in: Parallel Problem Solving from Nature, 2006, pp. 232–241.
[133] H. Handa, The effectiveness of mutation operation in the case of estimation of distribution algorithms, BioSystems 87 (2007) 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.
[134] A. Mendiburu, R. Santana, J.A. Lozano, E. Bengoetxea, A parallel framework for loopy belief propagation, in: Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation, GECCO'07, ACM, New York, NY, USA, 2007, pp. 2843–2850.
[135] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, San Mateo, CA, 1988.
[136] C.F. Lima, M. Pelikan, F.G. Lobo, D.E. Goldberg, Loopy substructural local search for the Bayesian optimization algorithm, in: T. Stützle, M. Birattari, H.H. Hoos (Eds.), SLS, in: Lecture Notes in Computer Science, vol. 5752, Springer, 2009, pp. 61–75.
[137] A. Ochoa, R. Höns, M. Soto, H. Muhlenbein, A Maximum Entropy Approach to Sampling in EDA—The Single Connected Case, Springer, 2003.
[138] R. Höns, Estimation of distribution algorithms and minimum relative entropy, Ph.D. Thesis, University of Bonn, Germany, 2006.
[139] R. Santana, Advances in graphical models for optimization and learning: applications in protein modeling, Ph.D. Thesis, University of the Basque Country, San Sebastian, Spain, 2006.
[140] H. Muhlenbein, R. Höns, The factorized distribution algorithm and the minimum relative entropy principle, in: Scalable Optimization via Probabilistic Modeling, 2006, pp. 11–37.
[141] R. Höns, R. Santana, P. Larrañaga, J.A. Lozano, Optimization by max-propagation using Kikuchi approximations, Tech. Rep. EHUKAT-IK-2-07, Department of Computer Science and Artificial Intelligence, University of the Basque Country, Urbana, IL, 2007.

[142] D.E. Goldberg, Using time efficiently: genetic-evolutionary algorithms and the continuation problem, in: Genetic and Evolutionary Computation Conference, GECCO-99, 1999, pp. 212–219. Also IlliGAL Report No. 99002.
[143] R. Srivastava, D.E. Goldberg, Verification of the theory of genetic and evolutionary continuation, in: Genetic and Evolutionary Computation Conference, GECCO-2001, 2001, pp. 551–558. Also IlliGAL Report No. 2001007.
[144] K. Sastry, D.E. Goldberg, Let's get ready to rumble: crossover versus mutation head to head, in: Genetic and Evolutionary Computation Conference, GECCO-2004, 2004, pp. 126–137.
[145] J. Ocenasek, J. Schwarz, The parallel Bayesian optimization algorithm, in: Proceedings of the European Symposium on Computational Inteligence, Physica-Verlag, 2000, pp. 61–67.
[146] M. Pelikan, K. Sastry, D.E. Goldberg, Sporadic model building for efficiency enhancement of the hierarchical BOA, Genetic Programming and Evolvable Machines 9 (2008) 53–84.
[147] K. Sastry, Efficient atomic cluster optimization using a hybrid extended compact genetic algorithm with seeded population, IlliGAL Report No. 2001018, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2001.
[148] M. Pelikan, D.E. Goldberg, Hierarchical BOA solves Ising spin glasses and MAXSAT, in: Genetic and Evolutionary Computation Conference, GECCO-2003, vol. II, 2003, pp. 1275–1286.
[149] H. Mühlenbein, T. Mahnig, Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning, International Journal of Approximate Reasoning 31 (2002) 157–192.
[150] H. Muhlenbein, T. Mahnig, Convergence theory and applications of the factorized distribution algorithm, Journal of Computing and Information Technology 7 (1999).
[151] Q. Zhang, H. Muhlenbein, On the convergence of a class of estimation of distribution algorithms, IEEE Transactions on Evolutionary Computation 8 (2004) 127–136.
[152] Q. Zhang, On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm, IEEE Transactions on Evolutionary Computation 8 (2004) 80–93.
[153] D.E. Goldberg, K. Deb, J.H. Clark, Genetic algorithms, noise, and the sizing of populations, Complex Systems 6 (1992) 333–362.
[154] D.E. Goldberg, K. Sastry, T. Latoza, On the supply of building blocks, in: L. Spector, et al. (Eds.), Genetic and Evolutionary Computation Conference, GECCO-2001, Morgan Kaufmann, San Francisco, CA, 2001, pp. 336–342. Also IlliGAL Report No. 2001015.
[155] M. Pelikan, K. Sastry, D.E. Goldberg, Scalability of the Bayesian optimization algorithm, International Journal of Approximate Reasoning 31 (2002) 221–258.
[156] T.-L. Yu, K. Sastry, D.E. Goldberg, M. Pelikan, Population sizing for entropy-based model building in discrete estimation of distribution algorithms, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO'07, ACM, New York, NY, USA, 2007, pp. 601–608.
[157] J.L. Shapiro, Drift and scaling in estimation of distribution algorithms, Evolution Computing 13 (2005) 99–123.
[158] P.A. Bosman, J. Grahl, D. Thierens, Enhancing the performance of maximum—likelihood Gaussian EDAs using anticipated mean shift, in: Proceedings of the 10th International Conference on Parallel Problem Solving from Nature: PPSN X, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 133–143.
[159] Y. Gao, J. Culberson, Space complexity of estimation of distribution algorithms, Evolutionary Computation 13 (2005) 125–143.
[160] M. Pelikan, K. Sastry, D.E. Goldberg, iBOA: the incremental Bayesian optimization algorithm, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO'08, ACM, New York, NY, USA, 2008, pp. 455–462.
[161] M. Hauschild, M. Pelikan, K. Sastry, C. Lima, Analyzing probabilistic models in hierarchical BOA, IEEE Transactions on Evolutionary Computation 13 (2009) 1199–1217.
[162] C.F. Lima, F.G. Lobo, M. Pelikan, D. Goldberg, Model accuracy in the Bayesian optimization algorithm, Soft Computing (2010).
[163] C. Echegoyen, A. Mendiburu, R. Santana, J.A. Lozano, Toward understanding EDAs based on Bayesian networks through a quantitative analysis, IEEE Transactions on Evolutionary Computation 15 (2011) 1–17.
[164] E. Radetic, M. Pelikan, Spurious dependencies and EDA scalability, in: GECCO, pp. 303–310.