
Computational Optimization and Applications, 21, 5–20, 2002

A Survey of Optimization by Building and Using Probabilistic Models

MARTIN PELIKAN pelikan@illigal.ge.uiuc.edu
DAVID E. GOLDBERG deg@illigal.ge.uiuc.edu
FERNANDO G. LOBO lobo@illigal.ge.uiuc.edu
Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Received September 22, 1999; Revised March 10, 2000

Abstract. This paper summarizes the research on population-based probabilistic search algorithms based on modeling promising solutions by estimating their probability distribution and using the constructed model to guide the exploration of the search space. It settles the algorithms in the field of genetic and evolutionary computation where they have been originated, and classifies them into a few classes according to the complexity of models they use. Algorithms within each class are briefly described and their strengths and weaknesses are discussed.

Keywords: genetic and evolutionary computation, genetic algorithms, model building, decomposable problems, stochastic optimization

1. Introduction

Recently, a number of evolutionary algorithms that guide the exploration of the search space by building probabilistic models of promising solutions found so far have been proposed. These algorithms have shown to perform very well on a wide variety of problems. However, in spite of a few attempts to do so, the field lacks a global overview of what has been done and where the research in this area is heading to.

The purpose of this paper is to review and describe basic principles of the recently proposed population-based search algorithms that use probabilistic modeling of promising solutions to guide their search. It settles the algorithms in the context of genetic and evolutionary computation, classifies the algorithms according to the complexity of models they use, and discusses the advantages and disadvantages of each of these classes.

The next section briefly introduces basic principles of genetic algorithms as our starting point. The paper continues by sequentially describing the classes of approaches classified according to complexity of a used class of models from the least to the most general one. Section 4 describes a few approaches that work with other than string representation of solutions. Section 5 summarizes and concludes the paper.

2. Genetic algorithms, problem decomposition, and building blocks

Simple genetic algorithms (GAs) [11, 18] are population-based search algorithms that guide the exploration of the search space by application of selection and genetic operators

of recombination/crossover and mutation. They are usually applied to problems where the solutions are represented or can be mapped onto fixed-length strings over a finite alphabet.

The user defines the problem that the GA will attempt to solve by choosing the length and base alphabet of strings representing the solutions and defining a function, usually called *fitness function*, that discriminates the string solutions according to their quality. For each string, the fitness function returns a real number quantifying its quality. The higher the fitness, the better the solution.

GAs start with a randomly generated population of solutions. From the current population of solutions the better solutions are selected by the *selection* operator. The selected solutions are processed by applying *recombination* and *mutation* operators. Recombination combines multiple (usually two) solutions that have been selected together by exchanging some of their parts. There are various strategies to do this, e.g. one-point and uniform crossover. Mutation performs a slight perturbation to the resulting solutions. Created solutions replace some of the old ones and the process is repeated until the termination criteria given by the user are met.

By selection, the search is biased to the high-quality solutions. New regions of the search space are explored by combining and mutating repeatedly selected promising solutions. By mutation, close neighborhood of the original solutions is explored like in a local hill-climbing. Recombination brings up innovation by combining pieces of multiple promising solutions together. GAs should therefore work very well for problems that can be somehow decomposed into subproblems of bounded difficulty by solving and combining the solutions of which a global solution can be constructed. Over-average solutions of these sub-problems are often called *building* blocks in GA literature.

Reproducing the building blocks by applying selection and preserving them from disruption, in combination with effective mixing, is a very powerful principle to solve decomposable problems [15, 28] which can be additively decomposed into terms of bounded order. An example of such decomposable function is a simple linear function called one-max which counts bits in the input string. A more complex example is the graph partitioning where each edge between two vertices from different partitions negatively contributes to the overall function by a constant penalty. By using the same principle, a much wider class of problems can be solved [12], including scheduling [19], telecommunication network optimization [38], and real-valued problems [3].

However, fixed, problem-independent recombination operators often either break the building blocks frequently or do not mix them effectively. GAs work very well only for problems where the building blocks are located tightly in strings representing the solutions [45]. On problems with the building blocks spread all over the solutions, the simple GAs experience very poor performance [45]. That is why there has been a growing interest in methods that learn the structure of a problem on the fly and use this information to ensure a proper mixing and growth of building blocks. One of the approaches is based on probabilistic modeling of promising solutions to guide the exploration of the search space instead of using crossover and mutation like in the simple GAs.

Probability distributions were recently used in various recombination schemes to generate new offspring, such as blend crossover [8], simulated binary crossover [6], fuzzy recombination [46] and UNDX [30]. However, in all these approaches, only two or three-parent

recombination is proposed. The methods discussed in this paper use macroscopic information about promising solutions as the marginal and conditional probabilities over large samples of high-quality solutions. Once the model of good solutions is constructed, this model is used to generate new points, regardless of the original population.

3. Evolutionary algorithms based on probabilistic modeling

From an abstract point of view, the selected set of promising solutions can be viewed as a sample drawn from an unknown probability distribution. Knowing that distribution would allow the optimization algorithm to generate new solutions that are somehow similar to the ones contained in the original selected set of solutions.

As pointed out, the true probability distribution is unknown. However, there are algorithms that are able to estimate that probability distribution by using the selected set of solutions itself and use this estimate to generate new solutions. These algorithms are called the probabilistic model-building genetic algorithms (PMBGAs), or the estimation of distribution algorithms (EDAs) [29]. In PMBGAs better solutions are selected from an initially randomly generated population of solutions like in the simple GA. Then, the true probability distribution of the selected set of solutions is estimated and new solutions are generated according to this estimate. The new solutions are then added into the original population, replacing some of the old ones. The process is repeated until the termination criteria are met.

The PMBGAs therefore do the same as the simple GAs except for that they replace genetic recombination and mutation operators by the following two steps:

1. A model (an estimate of the true distribution) of selected promising solutions is constructed.
2. New solutions are generated according to the constructed model.

Although PMBGAs process solutions in a different way than the simple GAs, it has been theoretically and empirically proven that the results of both can be very similar [16, 25]. For instance, the simple GA with uniform crossover which randomly picks a value on each position from either of the two parents works asymptotically the same as the so-called univariate marginal distribution algorithm [29] that assumes that the variables are independent [16, 25, 36]. Both the PMBGAs as well as the GAs are trying to put the same bias on the search. This bias prefers the solutions that can be obtained by combining partial solutions of promising solutions found so far. The difference is in a way this is achieved.

A distribution estimate can capture a building-block structure of a problem very accurately and ensure a very effective mixing and re-production of building blocks. This results in a linear or sub-quadratic performance of PMBGAs on these problems [26, 31, 33]. In fact, with an accurate distribution estimate that captures a structure of the solved problem the PMBGAs unlike the simple GAs perform the same as GA theory with mostly used assumptions claims. However, estimation of the true distribution is far from a trivial task. There is a trade-off between the accuracy and efficiency of the estimate.

The following sections describe three classes of PMBGAs that can be applied to problems with solutions represented by fixed-length strings over a finite alphabet. The algorithms

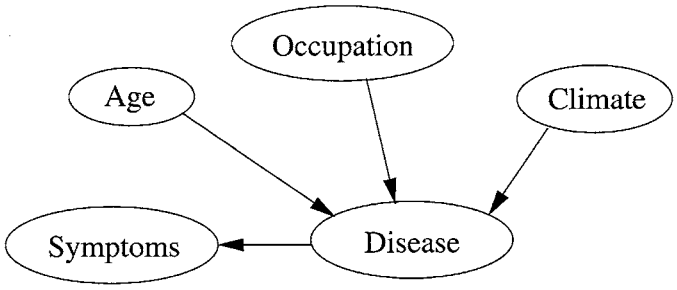


Figure 1. An example Bayesian network.

are classified according to the complexity of models they use. Starting with methods that assume that the variables in a problem (string positions) are independent, through the ones that take into account some pairwise interactions, to the methods that can accurately model even a very complex problem structure with highly overlapping multivariate building blocks.

An example model from each presented class of models will be shown. Models will be displayed as Bayesian networks, i.e. directed acyclic graphs with nodes corresponding to the variables in a problem (string positions) and edges corresponding to probabilistic relationships covered by the model. An edge between two nodes in a Bayesian network relates the two nodes so that the value of the variable corresponding to the terminal node of this edge depends on the value of the variable corresponding to the initial node of this edge. An example Bayesian network adapted from [44] is shown in figure 1. In this example, the variable *Disease* is conditioned on variables *Age, Occupation*, and *Climate*. *Symptoms* are conditioned on *Disease*. Other variables are assumed to be independent given their parents.

3.1. No interactions

The simplest way to estimate the distribution of promising solutions is to assume that the variables in a problem are independent and to look at the values of each variable regardless of the remaining solutions (see figure 2). The model of the selected promising solutions used to generate the new ones contains a set of frequencies of all values on all string positions in the selected set. These frequencies are used to guide further search by generating new string solutions position by position according to the frequency values. In this fashion, building blocks of order one are reproduced and mixed very efficiently. Algorithms based on this principle work very well on linear problems where the variables are not mutually interacting [15, 25].

In the population-based incremental learning (PBIL) algorithm [1] the solutions are represented by binary strings of fixed length. The population of solutions is replaced with the so-called probability vector which is initially set to assign each value on each position with the same probability 0.5. After generating a number of solutions the very best solutions are selected and the probability vector is shifted towards the selected solutions. The PBIL

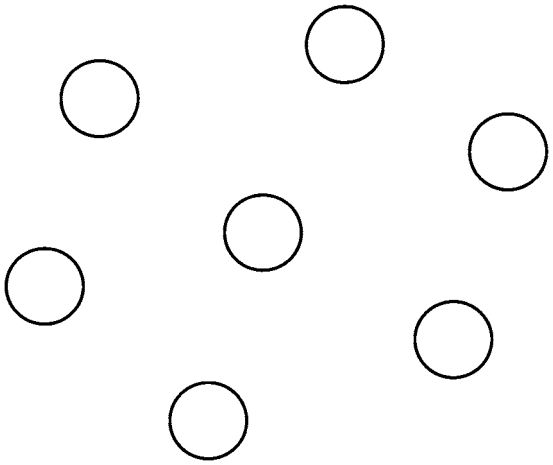


Figure 2. Graphical model with no interactions covered.

has been also referred to as the hill-climbing with learning (HCwL) [22] and the incremental univariate marginal distribution algorithm (IUMDA) [25] recently. A qualitative analysis of the PBIL was made by Kvasnicka et al. [22].

In the univariate marginal distribution algorithm (UMDA) [29] the population of solutions is processed. In each iteration the frequencies of values on each position in the selected set of promising solutions are computed and these are then used to generate new solutions which replace the old ones. The new solutions replace the old ones and the process is repeated until the termination criteria are met.

The compact genetic algorithm (cGA) [16] replaces the population with a single probability vector like the PBIL. However, unlike the PBIL, it modifies the probability vector so that there is direct correspondence between the population that is represented by the probability vector and the probability vector itself. Instead of shifting the vector components proportionally to the distance from either 0 or 1, each component of the vector is updated by shifting its value by the contribution of a single individual to the total frequency assuming a particular population size. By using this update rule, theory of simple genetic algorithms can be directly used in order to estimate the parameters and behavior of the cGA.

All algorithms described in this section perform similarly. They work very well for linear problems where they achieve linear or sub-quadratic performance, depending on the type of a problem, and they fail on problems with strong interactions among variables. For more information on the described algorithm as well as theoretical and empirical results, please see the cited papers.

Algorithms that do not take into account any interdependencies of various bits (variables) fail on problems where there are strong interactions among variables and where without taking into account these the algorithms are mislead. That is why a lot of effort has been put in extending methods that use a simple model that does not cover any interactions to methods that could solve a more general class of problems as efficiently as the simple PBIL, UMDA, or cGA can solve linear problems.

3.2. Pairwise interactions

The first algorithms that did not assume that the variables in a problem were independent could cover some pairwise interactions. An example of such algorithm is the mutual-information-maximizing input clustering (MIMIC) algorithm [5] which uses a simple chain distribution (see figure 3(a)) that maximizes the so-called mutual information of neighboring variables (string positions). In this fashion the Kullback-Liebler divergence [21] between the chain and the complete joint distribution is minimized. However, to construct a chain (which is equivalent to ordering the variables), MIMIC uses only a greedy search algorithm due to its efficiency, and therefore global optimality of the distribution is not guaranteed.

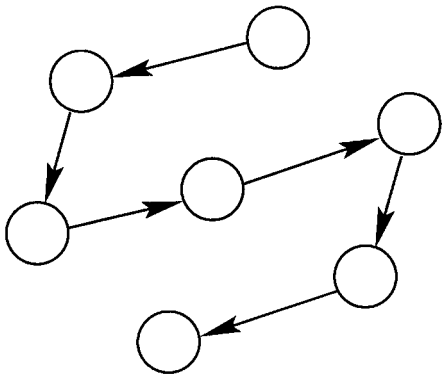
Baluja and Davies [2] use dependency trees (see figure 3(b)) to model promising solutions. Similarly as in the PBIL, the population is replaced by a probability vector which contains all pairwise probabilities. The probabilities are initialized to 0.25 and repeatedly adjusted according to new promising solutions acquired on the fly. There are two major advantages of using trees instead of chains. Trees are more general than chains because each chain is a tree. Moreover, by relaxing constraints of the model, in order to find the best model (according to a measure decomposable into terms of order two), a polynomial maximal branching algorithm [7] that guarantees global optimality of the solution can be used. On the other hand, MIMIC uses only a greedy search because in order to learn chain distributions, an NP-complete algorithm is needed.

In the bivariate marginal distribution algorithm (BMDA) [36] a forest (a set of mutually independent dependency trees, see figure 3(c) is used. This class of models is even more general than the class of dependency trees because a single tree is in fact a set of one tree. As a measure used to determine which variables should be connected and which should not, Pearson's chi-square test [23] is used. This measure is also used to discriminate the remaining dependencies in order to construct the final model.

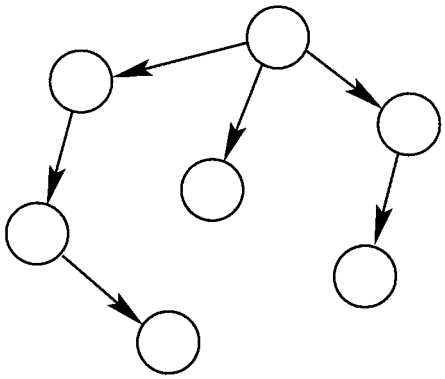
Pairwise models allow covering some interactions in a problem and are very easy to learn. The algorithms presented in this section reproduce and mix building blocks of order two very efficiently, and therefore they work very well on linear and quadratic problems [2, 4, 5, 25, 36]. The latter two approaches can also solve 2D spin-glass problems very efficiently [36]. Covering only some pairwise interactions has still shown to be insufficient to solve problems with multivariate or highly-overlapping building blocks [4, 36]. That is why the research in this area continued with more complex models.

3.3. Multivariate interactions

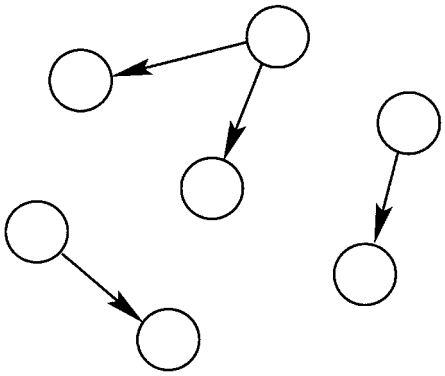
Using general models has brought powerful algorithms that are capable of solving many hard problems quickly, accurately, and reliably. However, it has also resulted in a necessity of using complex learning algorithms that require significant computational time and still do not guarantee global optimality of the resulting models. Nonetheless, in spite of increased computational time needed to learn the models, the number of evaluations of the optimized function is reduced significantly [33, 35, 38, 41]. That is why the overall time complexity is significantly reduced for large problems. Moreover, on many problems other algorithms simply do not work. Without learning the structure of a problem, algorithms must be either



(a) MIMIC



(b) Baluja & Davies

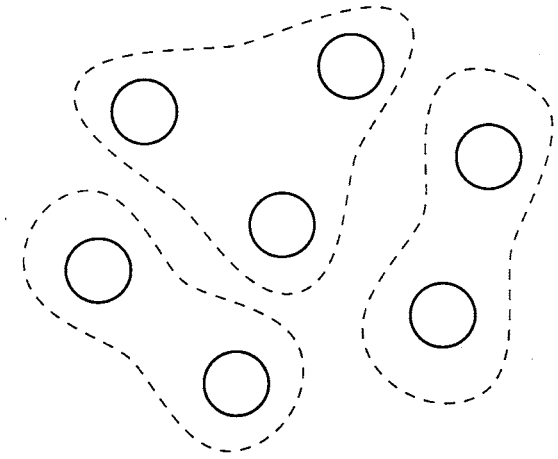


(c) BMDA

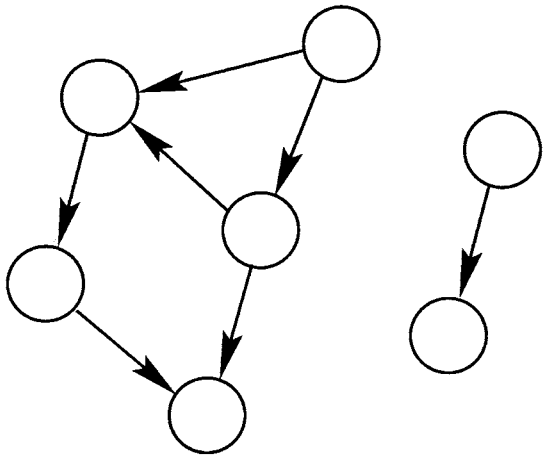
Figure 3. Graphical models with pairwise interactions covered.

given this information by an expert or they will simply be incapable of biasing the search in order to solve complex problems with a reasonable computational cost.

Algorithms presented in this section use models that can cover multivariate interactions. In the extended compact genetic algorithm (ECGA) [14], the variables are divided into a number of intact clusters which are manipulated as independent variables in the UMDA (see figure 4(a). Therefore, each cluster (building block) is taken as a whole and different clusters are considered to be mutually independent. To discriminate models, the ECGA uses



(a) ECGA



(b) BOA

Figure 4. Graphical models with multivariate interactions covered.

a minimum description length (MDL) metric [24] which prefers models that allow higher compression of data (selected set of promising solutions). The advantage of using the MDL metric is that it penalizes complex models when they are not needed and therefore the resulting models are not overly complex. To find a good model, a simple greedy algorithm is used. Starting with all variables separated, in each iteration current groups of variables are merged so that the metric increases the most. If no more improvement is possible, the current model is used.

Following from theory of the UMDA, for problems that are separable, i.e. decomposable into non-overlapping subproblems of a bounded order, the ECGA with a good model should perform in a sub-quadratic time. A question is whether the ECGA finds a good model and how much effort it takes. Moreover, many problems contain highly overlapping building blocks (e.g., 2D spin-glass systems) which can not be accurately modeled by simply dividing the variables into distinct classes. This results in a poor performance of the ECGA on these problems.

The factorized distribution algorithm (FDA) [28] uses a factorized distribution as a fixed model throughout the whole computation. The FDA is not capable of learning the structure of a problem on the fly. The distribution and its factorization are given by an expert. Distributions are allowed to contain marginal and conditional probabilities which are updated according to the currently selected set of solutions. It has been theoretically proven that when the model is correct, the FDA solves decomposable problems quickly, reliably, and accurately [28]. However, the FDA requires prior information about the problem in form of its decomposition and its factorization. Unfortunately, this is usually not available when solving real-world problems, and therefore the use of FDA is limited to problems where we can at least accurately approximate the structure of a problem.

The Bayesian optimization algorithm (BOA) [31] uses a more general class of distributions than the ECGA. It incorporates methods for learning Bayesian networks (see figure 4(b)) and uses these to model the promising solutions and generate the new ones. In the BOA, after selecting promising solutions, a Bayesian network that models these is constructed. The constructed network is then used to generate new solutions. As a measure of quality of networks, any metric can be used, e.g. Bayesian-Dirichlet (BD) metric [17], MDL metric, etc. In recently published experiments the BD scoring metric has been used. The BD metric does not prefer simpler models to the more complex ones. It uses accuracy of the encoded distribution as the only criterion. That is why the space of possible models has been reduced by specifying a maximal order of interactions in a problem that are to be taken into account. To construct the network with respect to a given metric, any algorithm that searches over the domain of possible Bayesian networks can be used. In recent experiments, a greedy algorithm has been used due to its efficiency.

The BOA is the first attempt to use general probabilistic models in optimization. It uses an equivalent class of models as the FDA; however, it does not require any information about the problem on input. It is able to discover this information itself. Nevertheless, prior information can be incorporated and the ratio of prior information and information contained in the set of high-quality solutions found so far can be controlled by the user. Not only does the BOA fill the gap between the FDA and uninformed search methods but also offers a method that is efficient even without any prior information [31, 32, 41] and still

does not prohibit further improvement by using this. Population sizing and convergence theory of the BOA was recently developed by [33]. An extension of the BOA to solve a very interesting class of hierarchically decomposable problems was proposed by [34].

Similar algorithms that use Bayesian networks to model promising solutions were later proposed by [9], who called the algorithm the estimation of Bayesian network algorithm (EBNA), and Mühlenbein and Mahnig [27], who called the algorithm the learning factorized distribution algorithm (LFDA). Both the EDNA as well as the LFDA proceed like the BOA, but they use an alternative to the metric used in the experiments presented by [31] to discriminate networks. This metric was previously used in the ECGA [14].

The algorithms that use models capable of covering multivariate interactions achieve a very good performance on a wide range of problems, e.g. 2D spin-glass systems [26, 31], graph partitioning [41], telecommunication network optimization [38], multidimensional real-valued problems [3], etc. However, even problems which are decomposable into terms of bounded order can still be very difficult to solve. Overlapping the subproblems can mislead the algorithm until the right solution to a particular subproblem is found and sequentially distributed across the solutions (e.g., see F_{0-peak} in [26]). Without generating the initial population with the use of problem-specific information, building blocks of size proportional to size of a problem have to be used which results in an exponential performance of the algorithms. This brings up a question on what are the problems we aim to solve by algorithms based on reproduction and mixing of building blocks that we have shortly discussed earlier in Section 2. We do not attempt to solve all problems that can be decomposed into terms of a bounded order, neither only these problems. The problems we approach to solve are decomposable in a sense that they can be solved by approaching the problem on a level of solutions of lower order by combining the best of which we can construct the optimal or a close-to-optimal solution. This is how we bias the search so that the total space explored by the algorithm substantially reduces by a couple orders of magnitude and computationally hard problems can be solved quickly, accurately, and reliably.

4. Beyond string representation of solutions

All algorithms described above work on problems defined on fixed-length strings over a finite alphabet. However, recently there have been a few attempts to go beyond this simple representation and directly tackle problems where the solutions are represented by vectors of real number or computer programs without mapping the solutions on strings. Most of these approaches use simple models that do not cover any interactions in a problem. However, there have been attempts to tackle more complex problems by using mixture models and continuous joint probabilistic models recently. Similar approaches are used in self-adaptive evolution strategies (ES) [37]. However, in evolution strategies the model of selected parents is not constructed and the individual solutions are not replaced by the constructed model. The selected parents are perturbed individually.

In the stochastic hill-climbing with learning by vectors of normal distributions (SHCLVND) [39] the solutions are represented by real-valued vectors. The population of solutions is replaced (and modeled) by a vector of mean values of Gaussian normal

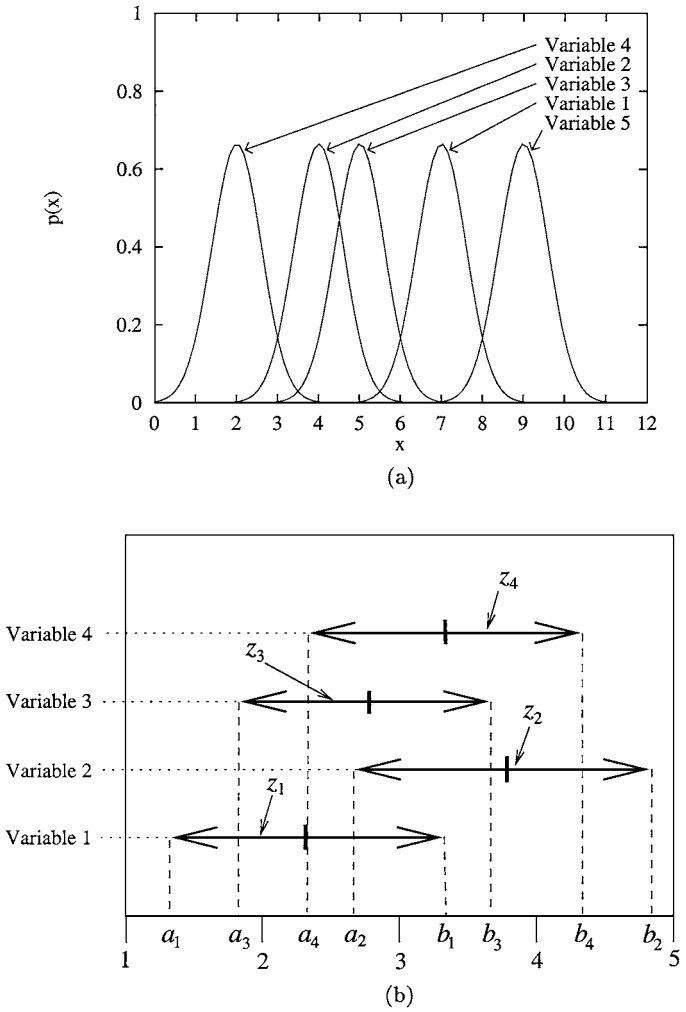


Figure 5. Probabilistic models of real vectors of independent variables. (a) SHCLVND (b) (Servet et al., 1998).

distribution μ_i for each optimized variable (see figure 5(a)). No interactions among the variables are covered. The standard deviation σ is stored globally and it is the same for all variables. After generating a number of new solutions, the mean values μ_i are shifted towards the best of the generated solutions and the standard deviation σ is reduced to make future exploration of the search space narrower. Various ways of modifying the σ parameter have been exploited in [42].

In another implementation of a real-coded PBIL [43], for each variable an interval (a_i, b_i) and a number z_i are stored (see figure 5(b)). The z_i stands for a probability of a solution to be in the right half of the interval. It is initialized to 0.5. Each time new solutions are generated using the corresponding intervals, the best solutions are selected and the numbers z_i are

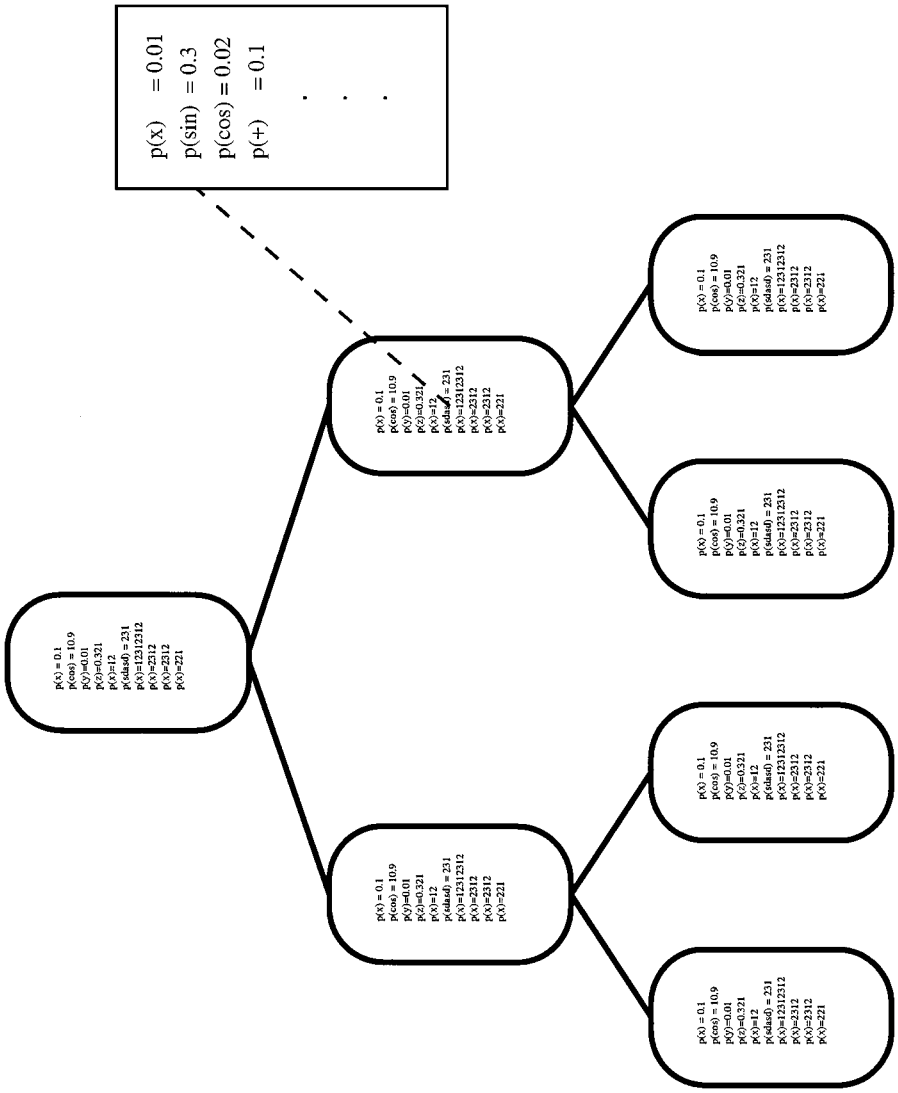


Figure 6. Graphical model of a program with no interactions covered used in PIPE.

shifted towards them. When z_i for a variable gets close to either 0 or 1, the interval is reduced to the corresponding half of it. In figure 5(b), each z_i is mapped to the corresponding interval (a_i, b_i). The used model also does not cover any interactions among the variables.

In [10], the PBIL is extended by using a finite adaptive Gaussian mixture model density estimator. This allows the algorithm to deal with multimodal distributions and explore different basins of attraction simultaneously.

Within the IDEA framework, Bosman [3] proposed the algorithm that uses the joint normal and the joint normal kernels distribution to model promising solutions represented by vectors of real numbers. These distributions are able to capture interactions of continuous variables. By generating new solutions according to this model, very good performance on a number of benchmark problems was achieved. The algorithm was compared with $(10 + 50)$-evolution strategies and other methods that use both binary as well as real-coded representations and it was shown to outperform other algorithms on all tested problems.

In the probabilistic incremental program evolution (PIPE) algorithm [40] computer programs or mathematical functions are evolved like in the genetic programming [20]. However, pair-wise crossover and mutation are replaced by probabilistic modeling of promising programs. Programs are represented by trees where each internal node represents a function or an instruction and leaves represent either input variable or a constant. In the PIPE algorithm, probabilistic representation of the program trees is used. Probabilities of each instruction in each node in a maximal possible tree are used to model promising programs and generate new ones (see figure 6). Unused portions of the tree are simply cut before the evaluation of the program by a fitness function. Initially, the model is set so that the trees are generated at random. From the current population of programs the ones that perform the best are selected. These are then used to update the probabilistic model. The process is repeated until the termination criteria are met.

Handley [13] used directed acyclic graphs to represent the population of programs (trees) in genetic programming. However, the method did not attempt to modify the recombination, but only to reduce the space to store the population and time to evaluate this population.

5. Summary and conclusions

Recently, the use of probabilistic modeling in genetic and evolutionary computation has become very popular. By combining various achievements of machine learning and genetic and evolutionary computation, efficient algorithms for solving a broad class of problems have been constructed. The most recent algorithms are continuously proving their wide-range applicability and efficiency, and offer a promising approach to solving the problems that can be resolved by combining high-quality pieces of information of a bounded order together.

To solve simple problems, algorithms that use a simple fixed or adaptive distribution estimate like the UMDA and BMDA can be used. To solve complex problems with strongly interacting decision variables, more sophisticated class of models must be considered and the BOA or ECGA should be used. In case of real-valued problems, the solution space can be either adequately discretized or the algorithms that evolve models of continuous solutions can be used.

In this paper, we have reviewed the algorithms that use probabilistic models of promising solutions found so far to guide further exploration of the search space. The algorithms have been classified in a few classes according to the complexity of models they use. Basic properties of each of these classes of algorithms have been shortly discussed and a thorough list of published papers and other references has been given.

Acknowledgments

The authors would like to thank Erick Cantú-Paz, Martin Butz, Dimitri Knjazew, and Jiri Pospichal for valuable discussions and useful comments that helped to shape the paper.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0050. Research funding for this project was also provided by a grant from the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies and endorsements, either expressed or implied, of the Air Force of Scientific Research or the U.S. Government.

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. S. Baluja and S. Davies, "Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space," in Proceedings of the 14th International Conference on Machine Learning, 1997, pp. 30–38.
3. P.A. Bosman, "Continuous iterated density estimation evolutionary algorithms within the IDEA framework," Personal communication, 2000.
4. P.A.N. Bosman and D. Thierens, "Linkage information processing in distribution estimation algorithms," in Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, Orlando, FL, W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith (Eds.), 1999, vol. I, pp. 60–67.
5. J.S. De Bonet, C.L. Isbell, and P. Viola, "MIMIC: Finding optima by estimating probability densities," in Advances in Neural Information Processing Systems, M.C. Mozer, M.I. Jordan, and T. Petsche (Eds.), 1997, vol. 9, p. 424.
6. K. Deb and R.B. Agrawal, "Simulated binary crossover for continuous search space," Complex Systems, vol. 9, pp. 115–148, 1995.
7. J. Edmonds, "Optimum branching," J. Res. NBS, vol. 71B, pp. 233–240, 1967.
8. L.J. Eshelman and J.D. Schaffer, "Real-coded genetic algorithms and interval-schemata," in Foundations of Genetic Algorithms Workshop (FOGA-92), D. Whitley (Ed.), Vail; Colorado, 1992.
9. R. Etxeberria and P. Larrañaga, "Global optimization using Bayesian networks," in Second Symposium on Artificial Intelligence (CIMAF-99), Habana, Cuba, 1999, pp. 332–339.
10. M. Gallagher, M. Frean, and T. Downs, "Real-valued evolutionary optimization using a flexible probability density estimator," in Proceedings of the Genetic and Evolutionary Computation Conference, W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith (Eds.), Orlando, Florida, USA, 1999, vol. 1, pp. 840–846.
11. D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley: Reading, MA, 1989.

12. D.E. Goldberg, "Genetic and evolutionary algorithms in the real world," University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IlliGAL Report No. 99013, 1999.
13. S. Handley, "On the use of a directed acyclic graph to represent a population of computer programs," in Proceedings of the First IEEE Conference on Evolutionary Computation, Piscataway, NJ, 1994, pp. 154–159.
14. G. Harik, "Linkage learning via probabilistic modeling in the ECGA," University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, IlliGAL Report No. 99010, 1999.
15. G. Harik, E. Cantú-Paz, D.E. Goldberg, and B.L. Miller, "The gambler's ruin problem, genetic algorithms, and the sizing of populations," in Proceedings of the International Conference on Evolutionary Computation (ICEC'97), Piscataway, NJ, 1997, pp. 7–12.
16. G.R. Harik, F.G. Lobo, and D.E. Goldberg, "The compact genetic algorithm," in Proceedings of the International Conference on Evolutionary Computation (ICEC'98), Piscataway, NJ, 1998, pp. 523–528.
17. D. Heckerman, D. Geiger, and M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," Microsoft Research, Redmond, WA, Technical Report MSR-TR-94-09, 1994.
18. J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press: Ann Arbor, MI, 1975.
19. D. Knjazew and D.E. Goldberg, "OMEGA—Ordering messy GA: Solving permutation problems with the fast messy genetic algorithm and random keys," University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, IlliGAL Report No. 2000004, 2000.
20. J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, The MIT Press: Cambridge, MA, 1992.
21. S. Kullback and R.A. Leibler, "On information and sufficiency," Annals of Math. Stats., vol. 22, pp. 79–86, 1951.
22. V. Kvasnicka, M. Pelikan, and J. Pospichal, "Hill climbing with learning (an abstraction of genetic algorithm)," Neural Network World, vol. 6, pp. 773–796, 1996.
23. L.A. Marascuilo and M. McSweeney, Nonparametric and Distribution-Free Methods for the Social Sciences, Brooks/Cole Publishing Company: CA, 1977.
24. T.M. Mitchell, Machine Learning, McGraw-Hill: New York, 1997.
25. H. Mühlenbein, "The equation for response to selection and its use for prediction," Evolutionary Computation, vol. 5, no. 3, pp. 303–346, 1997.
26. H. Mühlenbein and T. Mahnig, "Convergence theory and applications of the factorized distribution algorithm," Journal of Computing and Information Technology, vol. 7, no. 1, pp. 19–32, 1998.
27. H. Mühlenbein and T. Mahnig "FDA—A scalable evolutionary algorithm for the optimization of additively decomposed functions," Evolutionary Computation, vol. 7, no. 4, pp. 353–376, 1999.
28. H. Mühlenbein, T. Mahnig, and A.O. Rodriguez, "Schemata, distributions and graphical models in evolutionary optimization," Journal of Heuristics, vol. 5, pp. 215–247, 1999.
29. H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions I. Binary parameters," in Parallel Problem Solving from Nature—PPSN IV, Berlin, A. Eiben, T. Bäck, M. Shoenauer, and H. Schwefel (Eds.), 1996, pp. 178–187.
30. I. Ono and S. Kobayashi, "A real-coded genetic algorithm for function optimization using unimodal normal distribution crossovers," in Proceedings of the Seventh International Conference on Genetic Algorithms, San Francisco, T. Bäck (Ed.), 1997, pp. 246–253.
31. M. Pelikan, D.E. Goldberg, and E. Cantú-Paz, "Linkage problem, distribution estimation, and Bayesian networks," University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, IlliGAL Report No. 98013, 1998.
32. M. Pelikan, D.E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithms," in Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, Orlando, FL, W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith (Eds.), 1999, vol. I, pp. 525–532.
33. M. Pelikan, D.E. Goldberg, and E.Cantú-Paz, "Bayesian optimization algorithm, population sizing, and time to convergence," University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, IlliGAL Report No. 2000001, 2000.
34. M. Pelikan, D.E. Goldberg, and E. Cantú-Paz, "Hierarchical problem solving by the Bayesian optimization algorithms," University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, IlliGAL Report No. 2000002, 2000.

35. M. Pelikan, D.E. Goldberg, and E.Cantú-Paz, "Linkage problem, distribution estimation, and Bayesian networks," Evolutionary Computation, vol. 8, no. 3, pp. 311–341, 2000.
36. M. Pelikan and H. Mühlenbein, "The bivariate marginal distribution algorithm," in Advances in Soft Computing—Engineering Design and Manufacturing, London, R. Roy, T. Furuhashi, and P.K. Chawdhry (Eds.), 1999, pp. 521–535.
37. I. Rechenberg, Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Frommann-Holzboog: Stuttgart, 1973.
38. F. Rothlauf, D.E. Goldberg, and A. Heinzl, "Bad codings and the utility of well-designed genetic algorithms," University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, IlliGAL Report No. 200007, 2000.
39. S. Rudlof and M. Köppen, "Stochastic hill climbing with learning by vectors of normal distributions," in First On-line Workshop on Soft Computing, Nagoya, Japan, 1996.
40. R.P. Salustowicz and J. Schmidhuber, "Probabilistic incremental program evolution: Stochastic search through program space," in Machine Learning: ECML-97, M. van Someren and G. Widmer (Eds.), vol. 1224 of Lecture Notes in Artificial Intelligence, 1997, pp. 213–220.
41. J. Schwarz and J. Ocenasek, "Experimental study: Hypergraph partitioning based on the simple and advanced algorithms BMDA and BOA," in Proceedings of the Fifth International Conference on Soft Computing, Brno, Czech Republic, 1999, pp. 124–130.
42. M. Sebag and A. Ducoulombier, "Extending population-based incremental learning to continuous search spaces," in Parallel Problem Solving from Nature—PPSN V, Berlin Heidelberg, 1998, pp. 418–427.
43. I. Servet, L. Trave-Massuyes, and D. Stern, "Telephone network traffic overloading diagnosis and evolutionary computation techniques," in Proceedings of the Third European Conference on Artificial Evolution (AE'97), NY, G. Goos, J. Hartmanis, and J. Leeuwen (Eds.), 1997, pp. 137–144.
44. R. Shachter and D. Heckerman, "Thinking backwards for knowledge acquisition," AI Magazine, vol. 7, pp. 55–61, 1987.
45. D. Thierens, "Analysis and design of genetic algorithm," Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 1995.
46. H.-M. Voigt, H. Mühlenbein, and D. Cvetković, "Fuzzy recombination for the breeder genetic algorithm," in Proceedings of the Sixth International Conference on Genetic Algorithms, 1995, pp. 104–111.