Fitness Inheritance in the Bayesian Optimization Algorithm

Martin Pelikan[1] and Kumara Sastry[2]

[1] Dept. of Math. and Computer Science, 320 CCB
University of Missouri at St. Louis
8001 Natural Bridge Rd., St. Louis, MO 63121
pelikan@cs.umsl.edu
[2] Illinois Genetic Algorithms Laboratory, 107 TB
University of Illinois at Urbana-Champaign
104 S. Mathews Ave. Urbana, IL 61801
kumara@illigal.ge.uiuc.edu

Abstract. This paper describes how fitness inheritance can be used to estimate fitness for a proportion of newly sampled candidate solutions in the Bayesian optimization algorithm (BOA). The goal of estimating fitness for some candidate solutions is to reduce the number of fitness evaluations for problems where fitness evaluation is expensive. Bayesian networks used in BOA to model promising solutions and generate the new ones are extended to allow not only for modeling and sampling candidate solutions, but also for estimating their fitness. The results indicate that fitness inheritance is a promising concept in BOA, because population-sizing requirements for building appropriate models of promising solutions lead to good fitness estimates even if only a small proportion of candidate solutions is evaluated using the actual fitness function. This can lead to a reduction of the number of actual fitness evaluations by a factor of 30 or more.

1 Introduction

To ensure reliable convergence to a global optimum, genetic and evolutionary algorithms (GEAs) must often maintain a large population of candidate solutions for a number of iterations. However, in many real-world problems, fitness evaluation is computationally expensive and evaluating even moderately sized populations of candidate solutions is intractable. For example, fitness evaluation may include a large finite element analysis, it may consist of a complex traffic simulation, or it may require interaction with a human expert.

This leads to an interesting question: Would it be possible to make GEAs evolve not only the population of candidate solutions, but also a model of fitness, which could be used to evaluate a certain proportion of newly generated candidate solutions (fitness inheritance)? Fortunately, the answer to the above question is positive, and a few studies have been made to support this argument. Methods were proposed for fitness inheritance in the simple genetic algorithm

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 48–59, 2004.

(GA) [1] and the univariate marginal distribution algorithm (UMDA) [2]. In both cases, the results were promising and suggested that fitness inheritance can significantly reduce the number of fitness evaluations.

The purpose of this paper is to propose a method that uses models of promising solutions developed by the Bayesian optimization algorithm (BOA) [3,4] to model the fitness landscape and estimate fitness of newly generated candidate solutions. Two types of models are considered: (1) traditional Bayesian networks with full conditional probability tables (CPTs) used in BOA and (2) Bayesian networks with local structures used in BOA with decision graphs (dBOA) [5] and the hierarchical BOA (hBOA) [6,7]. Since the model in BOA captures significant nonlinearities in the fitness landscape, using this model as the basis for developing a model of the fitness landscape seems to be a promising approach. Of course, other methods, such as neural networks or various regression models, could be used instead. The proposed method is examined on BOA with decision trees on three example problems: onemax, concatenated traps of order 4, and concatenated traps of order 5. The results indicate that fitness inheritance is beneficial in BOA even if only less than 1% of candidate solutions are evaluated using the actual fitness function. It turns out that due to the population sizing requirements for creating a correct model of promising solutions, the more fitness inheritance, the better.

The paper starts by discussing BOA and previous fitness inheritance studies. Section 4 presents the proposed method for fitness inheritance in BOA. Section 5 presents and discusses experimental results. Section 6 summarizes and concludes the paper.

2 Bayesian Optimization Algorithm

Probabilistic model-building genetic algorithms (PMBGAs) [8] replace traditional variation operators of genetic and evolutionary algorithms [9,10] by building a probabilistic model of promising solutions and sampling the model to generate new candidate solutions. The Bayesian optimization algorithm (BOA) [3] uses Bayesian networks to model candidate solutions.

BOA evolves a population of candidate solutions to the given problem. The first population of candidate solutions is usually generated randomly according to a uniform distribution over all solutions. The population is updated for a number of iterations using two basic operators: (1) selection, and (2) variation. The selection operator selects better solutions at the expense of the worse ones from the current population, yielding a population of promising candidates. The variation operator starts by learning a probabilistic model of the selected solutions that encodes features of these promising solutions and the inherent regularities. Bayesian networks are used to model promising solutions because Bayesian networks are among the most powerful tools for capturing and representing decomposition [11], which is an inherent feature of most complex real-world systems [12]. The variation operator then proceeds by sampling the probabilistic model to generate new solutions, which are incorporated into the

original population. Here, a simple replacement scheme is used where new solutions fully replace the original population. A more detailed description of BOA can be found in [4].

The remainder of this section discusses Bayesian networks, which are going to serve as the basis for developing the model of fitness in BOA.

2.1 Bayesian Networks

Bayesian networks (BNs) [13,14,15] are among the most popular graphical models, where statistics, modularity, and graph theory are combined in a practical tool for estimating probability distributions and inference. A Bayesian network is defined by two components: (1) a structure, and (2) parameters. The structure is encoded by a directed acyclic graph with the nodes corresponding to the variables in the modeled data set (in this case, to the positions in solution strings) and the edges corresponding to conditional dependencies. The parameters are represented by a set of conditional probability tables (CPTs) specifying a conditional probability for each variable given any instance of the variables that the variable depends on.

A Bayesian network encodes a joint probability distribution given by

$$p(X) = \prod_{i=1}^{n} p(X_i|\Pi_i), \tag{1}$$

where $X = (X_1, \ldots, X_n)$ is a vector of all the variables in the problem; Π_i is the set of parents of X_i (the set of nodes from which there exists an edge to X_i); and $p(X_i|\Pi_i)$ is the conditional probability of X_i given its parents Π_i.

A directed edge relates the variables so that in the encoded distribution, the variable corresponding to the terminal node is conditioned on the variable corresponding to the initial node. More incoming edges into a node result in a conditional probability of the variable with a condition containing all its parents. In addition to encoding dependencies, each Bayesian network encodes a set of independence assumptions. Independence assumptions state that each variable is independent of any of its antecedents in the ancestral ordering, given the values of the variable's parents.

To learn Bayesian networks, a greedy algorithm is usually used for its efficiency and robustness. The greedy algorithm starts with an empty Bayesian network. Each iteration then adds an edge into the network that improves quality of the network the most. Network quality can be measured by any popular scoring metric for Bayesian networks, such as the Bayesian Dirichlet metric with likelihood equivalence (BDe) [16,17] or the Bayesian information criterion (BIC) [18]. The learning is terminated when no more improvement is possible.

2.2 Conditional Probability Tables (CPTs)

Conditional probability tables (CPTs) store conditional probabilities $p(X_i|\Pi_i)$ for each variable X_i. The number of conditional probabilities for a variable that

is conditioned on k parents grows exponentially with k. For binary variables, for instance, the number of conditional probabilities is 2^k, because there are 2^k instances of k parents and it is sufficient to store the probability of the variable being 1 for each such instance. Figure 1 shows an example CPT for $p(X_1|X_2, X_3, X_4)$.

Nonetheless, the dependencies sometimes also contain regularities. Furthermore, the exponential growth of full CPTs often obstructs the creation of models that are both accurate and efficient. That is why Bayesian networks are often extended with local structures that allow more efficient representation of local conditional probability distributions than full CPTs [19,20].

2.3 Decision Trees and Graphs for Conditional Probabilities

Decision trees are among the most flexible and efficient local structures, where conditional probabilities of each variable are stored in one decision tree. Each internal (non-leaf) node in the decision tree for $p(X_i|\Pi_i)$ has a variable from Π_i associated with it and the edges connecting the node to its children stand for different values of the variable. For binary variables, there are two edges coming out of each internal node; one edge corresponds to 0, whereas the other edge corresponds to 1. For more than two values, either one edge can be used for each value, or the values may be classified into several categories and each category would create an edge.

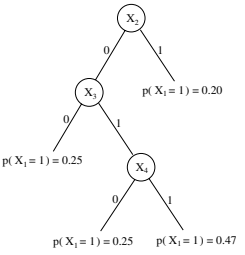
Each path in the decision tree for $p(X_i|\Pi_i)$ that starts in the root of the tree and ends in a leaf encodes a set of constraints on the values of variables in Π_i. Each leaf stores the value of a conditional probability of $X_i = 1$ given the condition specified by the path from the root of the tree to the leaf. A decision tree can encode the full conditional probability table for a variable with k parents if it splits to 2^k leaves, each corresponding to a unique condition. However, a decision tree enables more efficient and flexible representation of local conditional distributions. See Figure 1b for an example decision tree for the conditional probability table presented earlier.

A decision graph allows more edges to terminate in a single node. In other words, internal nodes in the decision tree are allowed to share children and, as a result, each node can have more than one parent. That makes this representation even more flexible. However, our experience indicates that, in BOA, decision graphs usually do not provide better performance than decision trees. See Figure 1c for an example decision graph.

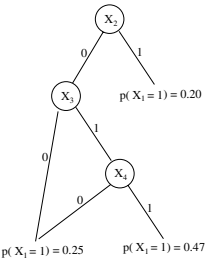
To learn Bayesian networks with decision trees, a decision tree for each variable X_i is initialized to an empty tree with a univariate probability of $X_i = 1$. In each iteration, each leaf of each decision tree is split to determine how quality of the current network improves by executing the split, and the best split is performed. The learning is finished when no splits improve the current network anymore. Quality of each model can be estimated using any popular scoring metric. Here we use a combination of the BDe [16,17] and BIC [18] metrics, where the BDe score is penalized with the number of bits required to encode parameters [4]. For decision graphs, a merge operation is introduced to allow for merging two leaves of any (but always the same) decision graph.

Π_1			$p(X_1 = 1 \mid \Pi_1)$
X_2	X_3	X_4	
0	0	0	0.25
0	0	1	0.25
0	1	0	0.25
0	1	1	0.47
1	0	0	0.20
1	0	1	0.20
1	1	0	0.20
1	1	1	0.20

(a) Conditional probability table.



(b) Decision tree.



(c) Decision graph.

Fig. 1. A conditional probability table for $p(X_1|X_2, X_3, X_4)$ using traditional representation (a) as well as local structures (b and c).

3 Previous Fitness Inheritance Studies

Despite the importance of fitness inheritance in robust population-based search, surprisingly few studies of fitness inheritance can be found. This section reviews the most important studies.

3.1 Fitness Inheritance in the Simple GA

Smith, Dike, and Stegmann [1] proposed two approaches to fitness inheritance in the simple GA [10]. The first approach is to compute the fitness of an offspring as the average fitness of its parents. The second approach is to consider a weighted average based on how similar the offspring is to each parent. The results indicated that GAs with fitness inheritance outperformed those without inheritance. However, the above study of fitness inheritance did not consider the effects of fitness inheritance on crucial GA parameters such as the population size and the number of generations. As a result, the speed-up achieved by using fitness inheritance could not be estimated properly.

Zhang, Julstrom, and Chen [21] used the aforementioned fitness inheritance model in the simple GA for design of vector quantization codebooks.

3.2 Fitness Inheritance in PMBGAs

Sastry, Goldberg, and Pelikan [2] considered the univariate marginal distribution algorithm (UMDA), which is one of the simplest PMBGAs. Using fitness inheritance in UMDA introduces new challenges, because UMDA does not use two-parent recombination and therefore it is difficult to find direct correspondence between parents and their offspring. Instead, Sastry et al. extend the probabilistic model to allow for estimating fitness of newly sampled candidate solutions.

UMDA models the population of promising solutions after selection using the probability vector, which stores the probability of a 1 at each position. These

probabilities are then used to sample new candidate solutions. To incorporate fitness inheritance, the probability vector $p = (p_1, p_2, \ldots, p_n)$ is extended to include additional two statistics $\bar{f}(X_i = 0)$ and $\bar{f}(X_i = 1)$ for each string position i. The term $\bar{f}(X_i = 0)$ denotes the average fitness of all solutions where the ith bit is 0; analogously, the term $\bar{f}(X_i = 1)$ denotes the average fitness of solutions with the ith bit equal to 1. The fitness of each new solution can then estimated as

$$f_{est}(X_1, X_2, \ldots, X_n) = \bar{f} + \sum_{i=1}^{n} \left(\bar{f}(X_i) - \bar{f} \right), \tag{2}$$

where $\bar{f}$ is the average fitness of all solutions used to estimate the fitness.

Sastry et al. [2] developed theory for UMDA on onemax that estimates the number of actual fitness evaluations when a given proportion of candidate solutions inherits fitness, whereas the remaining candidate solutions are evaluated using the actual fitness. The basic idea is to start by adapting the population sizing and time to convergence models to UMDA with fitness inheritance, and relate these quantities to their counterparts in standard UMDA. If optimal population size is used in both cases, Sastry et al. showed that only about 20% evaluations can be saved. However, if the same population size is used in both cases, the number of evaluations decreases by a factor of more than three.

4 Modeling Fitness in BOA

This section describes how the fitness model is built and updated using Bayesian networks, and how new candidate solutions can be evaluated using the model. Both Bayesian networks with full CPTs as well as the ones with local structures are discussed. The section also discusses where the statistics can be acquired from to build an accurate fitness model.

4.1 Modeling Fitness Using Bayesian Networks

In UMDA, probabilities of a 1 at each position that form the probability vector are each coupled with an average fitness of a 0 and a 1 at that position. Analogically, Bayesian networks can be extended to incorporate an average fitness of a 0 and a 1 for each statistic stored by the model.

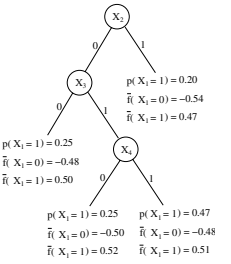
In BOA, for every variable X_i and each possible value x_i of X_i, an average fitness of solutions with $X_i = x_i$ must be stored for each instance π_i of X_i's parents Π_i. In the binary case, each row in the conditional probability table is thus extended by two additional entries. Figure 2a shows an example conditional probability table extended with fitness information based on the conditional probability table presented in Figure 1a. The fitness can then be estimated as

$$f_{est}(X_1, X_2, \ldots, X_n) = \bar{f} + \sum_{i=1}^{n} \left(\bar{f}(X_i|\Pi_i) - \bar{f}(\Pi_i) \right), \tag{3}$$

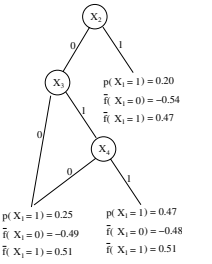
where $\bar{f}(X_i|\Pi_i)$ denotes the average fitness of solutions with X_i and Π_i, and $\bar{f}(\Pi_i)$ is the average fitness of all solutions with Π_i. Clearly,

Π_i			$p(X_i = 1 \mid \Pi_i)$	$\bar{f}(X_i = 0 \mid \Pi_i)$	$\bar{f}(X_i = 1 \mid \Pi_i)$
X_2	X_3	X_4			
0	0	0	0.25	−0.47	0.51
0	0	1	0.25	−0.52	0.49
0	1	0	0.25	−0.50	0.53
0	1	1	0.47	−0.37	0.47
1	0	0	0.20	−0.63	0.58
1	0	1	0.20	−0.45	0.46
1	1	0	0.20	−0.49	0.73
1	1	1	0.20	−0.51	0.62

(a) Conditional probability table.



(b) Decision tree.



(c) Decision graph.

Fig. 2. Fitness inheritance in a conditional probability table for $p(X_1|X_2, X_3, X_4)$ (a) and its representation using local structures (b and c).

$$\bar{f}(\Pi_i) = \sum_{X_i} p(X_i|\Pi_i)\bar{f}(X_i|\Pi_i). \tag{4}$$

4.2 Modeling Fitness Using Bayesian Networks with Decision Graphs

A similar method as for full CPTs can be used to incorporate fitness information into Bayesian networks with decision trees or graphs. The average fitness of each instance of each variable must be stored in every leaf of a decision tree or graph. Figure 2 shows an example decision tree and graph extended with fitness information based on the decision tree and graph presented earlier in Figure 1. The fitness averages in each leaf are restricted to solutions that satisfy the condition specified by the path from the root of the tree to the leaf.

Since decision graphs enable better encoding of statistical dependencies in the selected population of selected solutions, where the statistical dependencies originate in nonlinearities in the fitness function [22], using decision graphs for fitness inheritance should also improve fitness inheritance.

4.3 Where to Inherit Fitness from?

We still have not faced the following question: Where to obtain information to compute statistics used for fitness inheritance? More specifically, for each instance x_i of X_i and each instance π_i of X_i's parents Π_i, we must compute the average fitness of all solutions with $X_i = x_i$ and $\Pi_i = \pi_i$. Here we use two sources for computing the fitness-inheritance statistics:

1. Selected parents that were evaluated using the actual fitness function, and
2. the offspring that were evaluated using the actual fitness function.

The reason for restricting computation of fitness-inheritance statistics to selected parents and offspring is that the probabilistic model used as the basis for

selecting relevant statistics represents nonlinearities in the population of parents and the population of offspring. Since it is best to maximize learning data available, it seems natural to use these two populations to compute the fitness-inheritance statistics. The reason for restricting input for computing these statistics to solutions that were evaluated using the actual fitness function is that the fitness of other solutions was estimated only and it involves errors that could mislead fitness inheritance and propagate through generations. Both using only those solutions that were evaluated using the actual fitness function and incorporating the offspring in estimating inheritance statistics differs from previous fitness inheritance studies [1,2].

5 Experiments

This section describes experiments and provides experimental results. Test problems are described first. Next, experimental results are presented and discussed.

5.1 Onemax

Onemax is a simple linear function that computes the sum of bits in the input binary string:

$$f_{onemax}(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} X_i, \tag{5}$$

where $(X_1, X_2, \ldots, X_n)$ denotes the input binary string of n bits. In onemax, the fitness contribution of each bit is independent of its context. That is why a simple model used in UMDA that considers each variable independently of other variables suffices and yields convergence to the optimum in about $O(n \log n)$ evaluations. However, any other models of bounded complexity should work well, and practically any crossover operator used in standard GAs should also suffice.

In the model of fitness developed by BOA, the average fitness of a 1 in any leaf should be approximately 0.5, whereas the average fitness of a 0 in any leaf should be approximately -0.5. As a result, solutions will get penalized for 0s, while they would be rewarded for 1s. The average fitness will vary throughout the run. This paper considers onemax of $n = 50$ bits.

5.2 Concatenated 4-Bit Trap

In concatenated 4-bit traps [23,24], the input string is first partitioned into independent groups of 4 bits each. This partitioning should be unknown to the algorithm, but it should not change during the run. A 4-bit trap function is applied to each group of 4 bits and the contributions of all traps are added together to form the fitness. Each 4-bit trap is defined as follows:

$$trap_4(u) = \begin{cases} 4 & \text{if } u = 4 \\ 3 - u & \text{otherwise} \end{cases}, \tag{6}$$

where u is the number of 1s in the input string of 4 bits. An important feature of traps is that in each of the 4-bit traps, all 4 bits must be treated together, because all statistics of lower order lead the algorithm away from the optimum [24]. That is why most crossover operators as well as the model in UMDA will fail at solving this problem faster than in exponential number of evaluations, which is just as bad as blind search.

Unlike in onemax, $\bar{f}(X_i = 0)$ and $\bar{f}(X_i = 1)$ depend on the state of the search because the distribution of contexts of each bit changes over time and bits in a trap are not independent. The context of each leaf also determines whether $\bar{f}(X_i = 0) < \bar{f}(X_i = 1)$ or $\bar{f}(X_i = 0) > \bar{f}(X_i = 1)$ in the leaf. This paper considers a trap consisting of 10 copies of the 4-bit trap, where the total number of bits is $n = 40$.

5.3 Concatenated 5-Bit Trap

Concatenated traps of order 5 can be defined analogically to traps of order 4, but instead of dealing with groups of 4 bits, groups of 5 bits are considered. The contribution of each group of 5 bits is computed as

$$trap_5(u) = \begin{cases} 5 & \text{if } u = 5 \\ 4 - u & \text{otherwise} \end{cases}, \tag{7}$$

where u is the number of 1s in the input string of 5 bits. Traps of order 5 also necessitate that all bits in each group are treated together, because statistics of lower order are misleading.

Average fitness values $\bar{f}(X_i)$ depend on context similarly as for traps of order 4, and they thus follow similar dynamics. This paper considers a trap consisting of 10 copies of the 5-bit trap, where the total number of bits is $n = 50$.

5.4 Experimental Results

On each test problem, the following fitness inheritance proportions were considered: 0 to 0.9 with step 0.1, 0.91 to 0.99 with step 0.01, and 0.991 to 0.999 with step 0.001. For each test problem and fitness inheritance proportion, 30 independent experiments were performed. Each experiment consisted of 10 independent runs with the minimum population size to ensure convergence to a solution within 10% of the optimum (i.e., with at least 90% correct bits) in all 10 runs. For each experiment, bisection method was used to determine the minimum population size, and the number of evaluations (excl. the evaluations done using the model of fitness) was recorded. The average of 10 runs in all experiments was then computed and displayed as a function of the proportion of candidate solutions for which fitness was estimated using the fitness model.

The results on onemax, traps of order 4, and traps of order 5, are shown in figures 3 and 4. In all experiments, the number of actual fitness evaluations decreases with the inheritance proportion and it reaches the optimum when the proportion of candidate solutions for fitness inheritance is more than 99%. That means that, considering only the actual fitness evaluations, evaluating less than

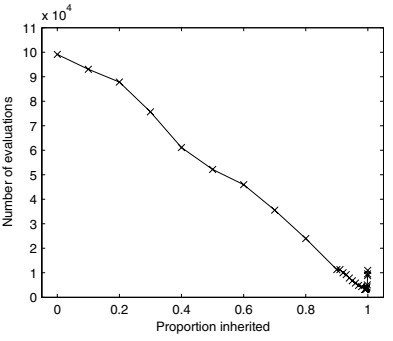
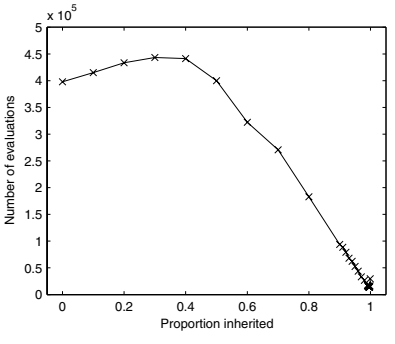
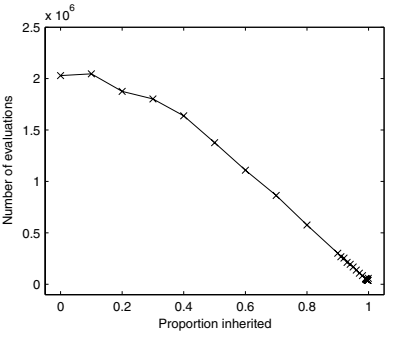


Fig. 3. Results on a 50-bit onemax.



(a) A 40-bit trap of order 4.



(b) A 50-bit trap of order 5.

Fig. 4. Results on concatenated traps of order 4 and 5.

1% of candidate solutions with the actual fitness seems to be beneficial. As a result, the number of evaluations of the actual fitness can be decreased by a factor of more than 31 for onemax, 32 for the trap of order 4, and 53 for the trap of order 5. Although the actual savings depend on the problem considered, it can be expected that fitness inheritance enables significant reduction of fitness evaluations on many problems because deceptive problems of bounded difficulty test BOA on the boundary of its design envelope to determine if BOA can solve a class of nearly decomposable problems [25].

Considering only the actual fitness evaluations ignores time complexity of selection, model construction, generation of new candidate solutions, and fitness estimation. Combining these factors with the complexity estimate for the actual fitness evaluation can be used to compute the optimal proportion of candidate solutions to evaluate using fitness inheritance. Nonetheless, the results presented in this paper clearly indicate that using fitness inheritance in BOA can reduce the number of solutions that must be evaluated using the actual fitness function by a factor of 30 or more. Consequently, if fitness evaluation is a bottleneck, there is a lot of space for improvement using fitness inheritance in BOA.

6 Summary and Conclusions

Fitness inheritance enables genetic and evolutionary algorithms to evaluate only a certain proportion of candidate solutions using the actual fitness function, while the fitness of remaining solutions is computed using a model of the fitness landscape updated on the fly. Using fitness models that can be updated and used efficiently can significantly speed up solution to problems where fitness evaluation is computationally expensive.

This paper showed that while fitness inheritance yields only moderate speed-ups of about 20% in simple GAs and UMDA, in BOA the benefits of using fitness inheritance become more significant. Due to rather large population-sizing requirements for creating an adequate probabilistic model of promising solutions in BOA, the number of actual function evaluations decreases even if less than 1% of candidate solutions are evaluated using the actual fitness function, while the fitness of the remaining solutions is estimated using only its model. That is an important result, because BOA and other advanced PMBGAs often require large populations, and evaluating large populations can become intractable for problems with computationally expensive fitness evaluation.

An important topic for future work is to incorporate fitness inheritance in presence of niching, which can lead to accumulation of candidate solutions whose fitness is overestimated. Resolving this problem would enable the use of fitness inheritance in the hierarchical BOA (hBOA) [6,7], which combines BOA with local structures and niching. Another important topic is to develop theory that extends theoretical work on fitness inheritance in UMDA to BOA and other competent GAs. Finally, it is important to apply the proposed fitness inheritance model to solve challenging real-world problems with expensive fitness evaluation.

Acknowledgments. The authors would like to thank David E. Goldberg for discussions and comments. A part of this work was supported by the Research Award at the University of Missouri at St. Louis and the Research Board at the University of Missouri. Most experiments were done on Asgard cluster at the Institute of Theoretical Physics at the Swiss Federal Institute of Technology (ETH) Zürich. The hBOA software, used by Pelikan, was developed by Martin Pelikan and David E. Goldberg at the University of Illinois at Urbana-Champaign.

References

1. Smith, R.E., Dike, B.A., Stegmann, S.A.: Fitness inheritance in genetic algorithms. Proceedings of the ACM Symposium on Applied Computing (1995) 345–350
2. Sastry, K., Goldberg, D.E., Pelikan, M.: Don't evaluate, inherit. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 551–558 Also IlliGAL Report No. 2001013.
3. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: BOA: The Bayesian optimization algorithm. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99) **I** (1999) 525–532 Also IlliGAL Report No. 99003.
4. Pelikan, M.: Bayesian optimization algorithm: From single level to hierarchy. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL (2002)

5. Pelikan, M., Goldberg, D.E., Sastry, K.: Bayesian optimization algorithm, decision graphs, and Occam's razor. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 519–526
6. Pelikan, M., Goldberg, D.E.: Escaping hierarchical traps with competent genetic algorithms. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 511–518 Also IlliGAL Report No. 2000020.
7. Pelikan, M., Goldberg, D.E.: A hierarchy machine: Learning to optimize from nature and humans. Complexity **8** (2003)
8. Pelikan, M., Goldberg, D.E., Lobo, F.: A survey of optimization by building and using probabilistic models. Computational Optimization and Applications **21** (2002) 5–20
9. Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, MI (1975)
10. Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, MA (1989)
11. Jordan, M.I., ed.: Learning in Graphical Models. MIT Press (1998)
12. Simon, H.A.: The Sciences of the Artificial. The MIT Press, Cambridge, MA (1968)
13. Howard, R.A., Matheson, J.E.: Influence diagrams. In Howard, R.A., Matheson, J.E., eds.: Readings on the principles and applications of decision analysis. Volume II. Strategic Decisions Group, Menlo Park, CA (1981) 721–762
14. Pearl, J.: Probabilistic reasoning in intelligent systems: Networks of plausible inference. Morgan Kaufmann, San Mateo, CA (1988)
15. Buntine, W.L.: Theory refinement of Bayesian networks. Proceedings of the Uncertainty in Artificial Intelligence (UAI-91) (1991) 52–60
16. Cooper, G.F., Herskovits, E.H.: A Bayesian method for the induction of probabilistic networks from data. Machine Learning **9** (1992) 309–347
17. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research, Redmond, WA (1994)
18. Schwarz, G.: Estimating the dimension of a model. The Annals of Statistics **6** (1978) 461–464
19. Chickering, D.M., Heckerman, D., Meek, C.: A Bayesian approach to learning Bayesian networks with local structure. Technical Report MSR-TR-97-07, Microsoft Research, Redmond, WA (1997)
20. Friedman, N., Goldszmidt, M.: Learning Bayesian networks with local structure. In Jordan, M.I., ed.: Graphical models. MIT Press, Cambridge, MA (1999) 421–459
21. Zheng, X., Julstrom, B., Cheng, W.: Design of vector quantization codebooks using a genetic algorithm. In: Proceedings of the International Conference on Evolutionary Computation (ICEC-97), Picataway, NJ, IEEE Press (1997) 525–529
22. Pelikan, M., Sastry, K., Goldberg, D.E.: Scalability of the Bayesian optimization algorithm. International Journal of Approximate Reasoning **31** (2002) 221–258
23. Ackley, D.H.: An empirical study of bit vector function optimization. Genetic Algorithms and Simulated Annealing (1987) 170–204
24. Deb, K., Goldberg, D.E.: Sufficient conditions for deceptive and easy binary functions. Annals of Mathematics and Artificial Intelligence **10** (1994) 385–408
25. Goldberg, D.E.: The design of innovation: Lessons from and for competent genetic algorithms. Volume 7 of Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers (2002)