Escaping Hierarchical Traps with Competent Genetic Algorithms

Martin Pelikan and David E. Goldberg

IlliGAL Report No. 2001003
January 2001

Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue Urbana, IL 61801
Office: (217) 333-2346
Fax: (217) 244-5705

Escaping Hierarchical Traps with Competent Genetic Algorithms

Martin Pelikan and David E. Goldberg
Illinois Genetic Algorithms Laboratory
104 S. Mathews Avenue, Urbana, IL 61801
University of Illinois at Urbana-Champaign
Phone/FAX: (217) 333-2346, (217) 244-5705
{pelikan,deg}@illigal.ge.uiuc.edu

Abstract

To solve hierarchical problems, one must be able to learn the linkage, represent partial solutions efficiently, and assure effective niching. Linkage learning results in a good problem solver on a single level. Niching and efficient representation of partial solutions ensure that the algorithm maintains enough alternative solutions on each level to compose the solutions on a higher level. We combine the Bayesian optimization algorithm, which has been shown to solve problems on a single level efficiently, with a powerful niching technique based on crowding and restricted tournament selection. Decision graphs are used as local structures to encode information about the relationships among the variables in a problem. The proposed algorithm is called the *hierarchical Bayesian optimization algorithm*. Additionally, we propose a new class of hierarchically decomposable problems that are deceptive on each level and show that the proposed algorithm scales up subquadratically on all test problems. The proposed class of problems is called *hierarchical traps*. Empirical results are in agreement with our recent convergence and population sizing theory.

1 Introduction

Genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989) combine short order partial solutions to form solutions of higher order. New solutions undergo selection and the process is repeated until the entire solution is formed. However, fixed, problem-independent, recombination operators have shown to perform quite poorly on problems with interactions among the variables spread across the solutions (Thierens & Goldberg, 1993; Pelikan, Goldberg, & Cantú-Paz, 1998; Bosman & Thierens, 1999). Moreover, the hierarchical nature of the optimization process has earned only little attention and it has been assumed that genetic algorithms do this automatically.

The purpose of this paper is to show that competent genetic algorithms which succeeded in solving problems of bounded difficulty on a single level quickly, accurately, and reliably, can be quite easily extended to solve problems that are hierarchical in their nature. We focus on the Bayesian optimization algorithm (Pelikan et al., 1998) using decision graphs to represent the conditional probabilities of the model used to represent populations of promising solutions. There are three major issues one must address to succeed in solving difficult hierarchical problems: linkage learning, niching, and efficient representation of the model. Linkage learning ensures powerful recombination. Niching and efficient representation of the model ensure preservation of alternative partial solutions that are assembled to form solutions of higher order. The proposed algorithm is called the *hierarchical Bayesian optimization algorithm* (BOA). Hierarchical BOA is able to solve problems that not only require that we approach them hierarchically, but that also deceive the algorithm to the local optimum on each level. Additionally, we design a hierarchical test problem which requires both efficient linkage learning and niching and perform a number of experiments to show that the proposed algorithm is able to solve the problem efficiently. Due to its deceptive nature, the proposed problem is called the *hierarchical trap*.

The paper starts by introducing probabilistic model building genetic algorithms (PMBGAs) (Pelikan, Goldberg, & Lobo, 2000) that evolve a probabilistic model of promising solutions to guide the search. Section 2.2 introduces the Bayesian optimization algorithm which uses Bayesian networks as a model. Section 3 introduces various niching methods that were used in the past as a diversity maintenance tool for genetic and evolutionary algorithms. The use of niching in PMBGAs is discussed. Test problems tackled in our experiments are described in Section 4. Section 5 provides and discusses the results of our experiments. Section 6 discusses interesting topics to be covered in future research in this area. Section 7 summarizes and concludes the paper.

2 Background

By applying recombination and mutation, GAs are manipulating a large number of promising partial solutions. However, fixed, problem independent, recombination and mutation operators often result in inferior performance even on simple problems. Without knowing where the important partial solutions are and designing problem specific operators that take this information into account, the required number of fitness evaluations and population size grow exponentially with the number of decision variables (Thierens & Goldberg, 1993).

That is why there has been a growing interest in *linkage learning* which studies methods that are able to learn where the important interactions in the problem are and use this information to combine solutions more effectively. This allows the algorithms to solve a large class of problems quickly, accurately, and reliably. One of the approaches to linkage learning is based on using probability distributions to model promising solutions found so far and generating new solutions according to the estimated distribution (Mühlenbein & Paaß, 1996; Pelikan, Goldberg, & Lobo, 2000). Probability distributions can capture variables which are correlated and the ones which are independent. This can subsequently be used to combine the solutions in more effective manner.

This section first reviews probabilistic model-building genetic algorithms. Subsequently, the Bayesian optimization algorithm is described. Finally, semantics, learning, and utilization of Bayesian networks are briefly discussed.

2.1 Probabilistic Model-Building Genetic Algorithms

The set of selected solutions can be seen as a sample from the space of solutions that we are interested in. Statistical information about the selected solutions can be used to estimate their distribution and this estimate can be used in order to generate new solutions. In other words, the distribution of "good" points can be estimated and the new points can be simply generated according to the same distribution. The algorithms based on this principle are called *probabilistic model-building genetic algorithms* (PMBGAs) (Pelikan, Goldberg, & Lobo, 2000), *estimation of distribution algorithms* (EDAs) (Mühlenbein & Paaß, 1996), or *iterated density estimation algorithms* (IDEAs) (Bosman, 2000).

However, estimating a multivariate distribution is not an easy task. There is a trade off between the accuracy of the estimation and its computational cost. To use computationally efficient methods, one must make a number of assumptions, all of which decrease the generality of the used method.

Moreover, it is very difficult to decide which model is the best one for our purpose. Too simple model may not cover all important interactions. Too complex model may not bring enough variation in the optimization process. Both too simple and too complex models may thus result in inferior performance. See Pelikan and Goldberg (2000a) and Pelikan, Goldberg, and Sastry (2000) for a discussion on this topic. Subsequent paragraphs describe basic principles of the algorithms that use probabilistic models of promising solutions to guide their search. For a more detailed overview of PMBGAs, see Pelikan et al. (2000).

Probably the simplest way to estimate the distribution of good solutions is to assume that the variables in a problem are independent. New solutions can be generated by only preserving the proportions of the values of all variables independently of the context. This is the basic principle of the population based incremental learning (PBIL) algorithm (Baluja, 1994), the compact genetic algorithm (cGA) (Harik et al., 1998), and the univariate marginal distribution algorithm (UMDA) (Mühlenbein, 1997). Since these algorithms take into account only contributions of the values of each variable without considering the contexts where these contributions take place, it is natural to expect that the algorithms should work very well on linear problems, but experience great difficulties on problems where the variables are correlated and therefore the context does matter.

The first attempts to resolve this problem were the incremental algorithm using so-called dependency trees in order to estimate the distribution of selected solutions (Baluja & Davies, 1997) and the population-based MIMIC algorithm using a simple chain distribution (De Bonet et al., 1997). Another population-based attempt to solve the problem of the disruption of building blocks of order two by using a slightly more general technique is the bivariate marginal distribution algorithm (BMDA) (Pelikan & Mühlenbein, 1999).

In the algorithms described above, to determine the contribution of a particular variable, the context of one other variable can be taken into account. Only acyclic models are allowed. Consequently, the algorithms can solve more complex problems efficiently. However, this has been shown to be still insufficient to solve problems with interactions of higher order efficiently (Pelikan & Mühlenbein, 1999; Bosman & Thierens, 1999). Covering pairwise interactions still does not preserve partial solutions of higher order. Moreover, interactions of higher order do not necessarily imply pairwise interactions that can be detected at the level of partial solutions of order two.

The factorized distribution algorithm (FDA) (Mühlenbein et al., 1998) uses a fixed factorization of the distribution to generate new candidate solutions. The FDA is capable of covering the interactions of higher order and combining important partial solutions effectively. It works very well on uniformly-scaled additively decomposable problems. However, the FDA requires prior information about the problem in the form of a problem decomposition and its factorization. As input, this algorithm gets complete or approximate information about the structure of a problem. Unfortunately, this information is mostly not available without computationally intensive problem analysis. We would like our algorithm to be able to learn the structure of the problem and utilize this information on the fly. Using an approximate distribution according to the current state of information represented by the set of promising solutions can be very effective even if it is not a valid factorization.

The above problem has been overcome with the Bayesian optimization algorithm (Pelikan, Goldberg, & Cantú-Paz, 1998), which uses Bayesian networks to model promising solutions and generate the new ones and the extended compact genetic algorithm (ECGA) (Harik, 1999) which uses the minimum description length metric to construct groups of correlated variables and generates new solutions accordingly. For another method using Bayesian networks, see also Etxeberria and Larrañaga (1999). The next subsection describes the Bayesian optimization algorithm.

2.2 Bayesian Optimization Algorithm

The Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1998) uses Bayesian networks to model promising solutions and subsequently guide the exploration of the search space. In the BOA, the first population of strings is generated randomly with a uniform distribution. The initial population can also be biased to the regions that we are interested in. From the current population, the better strings are selected. Any selection method can be used. A Bayesian network that fits the selected set of strings is constructed. Any metric as a measure of quality of networks and any search algorithm can be used to search over the networks in order to maximize/minimize the value of the used metric. Besides the set of good solutions, prior information about the problem can be used in order to enhance the estimation and subsequently improve convergence. New strings are generated according to the joint distribution encoded by the constructed network. The new strings are added into the old population, replacing some of the old ones. The pseudo-code of the BOA is shown in Figure 1.

The Bayesian Optimization Algorithm (BOA)

(1) set $t \leftarrow 0$
    randomly generate initial population $P(0)$
(2) select a set of promising strings $S(t)$ from $P(t)$
(3) construct the network B using a chosen metric and constraints
(4) generate a set of new strings $O(t)$ according to the joint distribution encoded by B
(5) create a new population $P(t+1)$ by replacing some strings from $P(t)$ with $O(t)$
    set $t \leftarrow t+1$
(6) if the termination criteria are not met, go to (2)

Figure 1: The pseudo-code of the Bayesian optimization algorithm.

The next subsection describes basic principles of learning and utilization of Bayesian networks. Subsequently, local structures that can be used to make the representation of the model more efficient are discussed and a simple greedy algorithm for network construction is briefly described.

2.3 Bayesian Networks

A Bayesian network (Pearl, 1988) is a directed acyclic graph with the nodes corresponding to the variables in the modeled data set (in our case, to the positions in solution strings). Mathematically, a Bayesian network encodes a joint probability distribution given by

$$p(X) = \prod_{i=0}^{n-1} p(X_i|\Pi_{X_i}), \tag{1}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of all the variables in the problem, Π_{X_i} is the set of parents of X_i in the network (the set of nodes from which there exists an edge to X_i) and $p(X_i|\Pi_{X_i})$ is the conditional probability of X_i given its parents Π_{X_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 corresponding variable with a conjunctional condition containing all its parents. The network encodes independence assumptions that each variable is independent of any of its antecedents in ancestral ordering given its parents.

To encode the conditional probabilities corresponding to the nodes of the network, one can use a simple probability table listing probabilities of all possible instances of a variable and its parents. The probabilities of one particular value of each variable can be eliminated and computed using the remaining ones because the probabilities sum to one. However, the size of such a table grows exponentially with the number of parents of the variable even though many probabilities of higher order may be the same. To solve hierarchical problems, it is essential to be able to represent conditional probabilities by structures that are polynomial in the order of interactions. While the order of interactions can be as high as the size of the problem, the number of corresponding alternative partial solutions must be polynomial in their order to allow efficient and reliable exploration. The next subsection presents alternative ways to represent conditional probabilities in the model which allow a more compact representation of the local densities in the model.

2.4 Local Structures in Bayesian Networks

One simple extension of the probability table is a *default table* (Friedman & Goldszmidt, 1999). In a default table, only some instances of the variable and its parents are listed together with the corresponding probabilities. The remaining probabilities are obtained from the default entry which is simply an average of the remaining (unlisted) probabilities.

One can use more complex local structures, such as *decision trees* (Friedman & Goldszmidt, 1999) or *decision graphs* (Chickering, Heckerman, & Meek, 1997). Each internal node of a decision tree or graph corresponds to some variable. Children (successors) of each internal node correspond to disjoint subsets of values the variable can obtain. For binary variables, each non-leaf node can have exactly two children where each child corresponds to one of the values zero and one. In case of bigger alphabets, there are more possibilities. A decision graph allows different parents to have the same child. This makes the structure both more general and expressive. In hierarchical BOA we use decision graphs. However, there is only little difference between the performance using decision graphs and trees. Since trees are simpler to interpret we used only decision trees in our experiments.

Using local structures can reduce the space we need to represent the model. Additionally, it can refine the model building by using smaller operators and make the model more general for some data sets. One can encode interactions of certain high order without having to consider exponentially many alternative instances and probabilities. Please, see Pelikan, Goldberg, and Sastry (2000) for more details on using decision graphs for model building in the BOA. See Figure 2 for an example of a decision tree and graph encoding the local probability density $p(z|x, y)$.

2.5 Learning Bayesian Networks

To construct the network, a simple greedy algorithm is usually used. This algorithm performs simple graph operations that improve the quality of the current network the most, starting from an empty network or a network from a different source. To determine the quality of each network, various scoring metrics can be used. Recently, we have used the Bayesian-Dirichlet metric, the minimum description length (MDL) metric, and a metric which is a combination of the Bayesian-Dirichlet and MDL metric. For more details on the network construction and scoring metrics for simple Bayesian networks of for the ones with local structures, please see Pelikan, Goldberg, and Cantú-Paz (2000b) and Pelikan, Goldberg, and Sastry (2000).

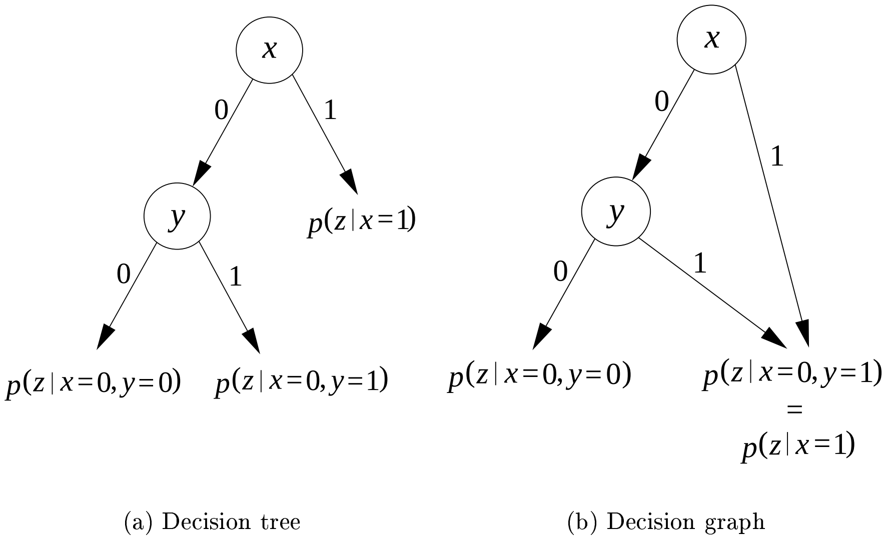


Figure 2: Example decision tree (a) and decision graph (b) for encoding a conditional probability distribution of $p(z|x, y)$.

3 Niching

The purpose of niching in genetic and evolutionary optimization is twofold: (1) discovery of multiple solutions of the problem and (2) preservation of alternative solutions until one can decide which solution is better. In some real-world applications it is important to find multiple solutions and let the expert or experiment decide which of the solutions is the best after all. This is usually the case when the fitness function does not fully determine which solution is the best in practice but only focuses on several aspects of solution quality, or when for the sake of efficiency instead of using a complete fitness function one uses only its approximation that is more computationally efficient. The reason for preserving multiple alternative solutions is that on some difficult problems one cannot clearly determine which alternative solutions are really on the right track until the optimization proceeds for a number of generations. Without niching the population is a subject to genetic drift which may destroy some alternatives before we find out whether or not they are the ones we are looking for.

There are three general approaches to niching. One approach modifies the fitness landscape before the selection is performed. The second approach modifies the selection itself to take into account the fitness as well as the genotype or the phenotype instead of using the fitness as the only criterion. Both approaches allow solutions that share many similarities to compete for common resources. Crowding, restricted mating, and fitness sharing are based on this idea. The third approach is to isolate several groups of individuals rather than to keep the entire population in one location. The individuals can migrate between different locations (islands or demes) at certain intervals and allow population at each location develop in isolation. In this paper we mainly focus on methods that modify only the selection and replacement mechanisms.

Some related work studies the preservation of diversity from a different point of view. The primary goal of these techniques is not the preservation of multiple solutions or alternative search regions, but the avoidance of premature convergence. One could, for instance, inject randomly generated individuals into the current population each now and then (Mauldin, 1984), or control the selection (Baker, 1985) somehow to prevent premature convergence. However, this is not a primary purpose of using niching in our work. Various techniques for niching were also proposed in the area of multiobjective optimization (Schaffer, 1984; Horn & Nafpliotis, 1993; Fonseca & Fleming, 1993). These methods are not applicable to single-criterion optimization and therefore we do not discuss them in this paper.

The next three sections briefly introduce history and existing methods in the three main classes of niching methods: selection-based, fitness-sharing, and island models. Subsequently, we describe the method used in hierarchical BOA.

3.1 Niching Based on Selection

All niching methods localize competition in some way. This section reviews methods that localize competition by modifying selection and replacement strategies. Cavicchio (1970) introduced the *preselection* where the offspring replaced the inferior parent. This encourages competition among similar individuals since the offspring and its parents usually share many similarities. This scheme was later generalized by De Jong (1975) who proposed the so-called *crowding*. In crowding, for each new individual a subset of the population is first selected. The new individual then replaces the most similar individual in this subset. Earlier in the run only little will change compared to a random replacement. However, as the run continues, the individuals will create groups of similar individuals who compete for space with other members of the same group.

The original purpose of crowding was not to preserve diversity but to speed up the convergence. De Jong (1975) showed that the simple genetic algorithm converged to the optimum faster when crowding was used on the tested multimodal function. Of course, crowding can be used also as a diversity preservation tool, having in mind premature convergence and preservation of alternative solutions, which is the primary goal of applying niching in our work. The crowding scheme was later used by Goldberg (1983) for a pipeline design by learning classifier systems.

Perry (1984) created multiple contexts in which the fitness function varied according to some externally defined schemata which define the species. Individuals could migrate between different contexts. The technique of Perry is mainly interesting for its biological background.

A number of techniques that *restrict mating* in some way to promote the niching behavior were proposed. Hollstein (1971) required close individuals to mate as long as their fitness improved. When the trend changes and the quality of the "family" decreases, crossbreeding across families is allowed. Booker (1982) discussed the need for restricted mating to prevent the formation of lethals. For the restricted mating, however, there is only little theory to support the presented ideas.

The *deterministic crowding* of Mahfoud (1992) pairs each offspring with the more similar parent and performs a competition among these pairs. The offspring replaces the parent only if it has a higher fitness. This idea was later studied by Mengshoel and Goldberg (1999) who proposed *probabilistic crowding* as a probabilistic extension of the original deterministic crowding method. In probabilistic crowding, the winner of the parent-offspring tournament is chosen by using the probability proportional to the fitness.

The *gene invariant genetic algorithm* (GIGA) (Culberson, 1992) maintains constant univariate frequencies of all values on all positions. For instance, in case of binary strings and uniformly generated initial population, at any point in the run there will be about half ones and half zeros on each string position. The GIGA is a static genetic algorithm, i.e. it selects and recombines only two individuals at each generation. Fitness proportionate selection is used to select the parents and the two children replace their parents. One-point, multiple-point, or uniform crossover can be used to recombine the parents. The GIGA never loses genetic material (particular alleles) and therefore mutation is eliminated. GIGA has been shown to perform the same or better than traditional (simple) GAs on De Jong's and deceptive test functions (Culberson, 1992). Preservation of univariate frequencies is a very interesting idea. However, it does not guarantee useful diversity. The success of the GIGA is determined by mixing (recombination).

Harik (1994) proposed the restricted tournament selection as an extension of De Jong's crowding. Restricted tournament selection (RTS) selects parents at random with a uniform distribution. After performing crossover, a subset of the population is selected for each parent, similarly as in crowding. However, instead of automatically replacing the closest individual, the two individuals compete and the one that has a higher fitness wins. In this fashion, the selection step is performed by elitist replacement with a flavor very similar to crowding. No extra selection operator is required. Harik showed that RTS performs very well on a number of multimodal problems and is able to locate all optima even on functions which are highly multimodal and very difficult to solve.

3.2 Niching by Fitness Sharing

To localize competition, fitness sharing modifies the fitness function. Both selection as well as replacement remain the same. The section first discusses the two-armed bandit problem which can be used as a simple intuitive example to introduce basic concepts of fitness sharing. Subsequently, recently used niching methods based on fitness sharing are presented and discussed.

The two-armed bandit problem, which is often used in machine learning to illustrate different facets of behavior of learning algorithms, was also used in one of the first studies of sharing. In the two-armed bandit problem, the goal is to pull one of the two handles of a slot machine to gain the most money at the end. The slot machine is a black box and therefore one can't predict the outcome based on anything but experience. The experience may be, of course, misleading.

There is a very interesting tradeoff in the two-armed bandit problem. If we pull the handle that seems to give a higher payoff at the moment, we may never realize that the other handle is better in the long run. However, if we pull the handle that seems to give lower payoff and this is in fact true, we are again losing by wasting our time with the wrong handle. Holland (1975) introduced a modification of the two-armed bandit problem where the number of times each handle is pulled is proportional to the average outcome we get after pulling the handle. The analogy between the two-armed bandit problem and genetic algorithms is apparent. The average outcome corresponds to the fitness in genetic algorithms. The number of samples we give to each handle corresponds to the number of copies in the current population. The basic idea is therefore to give each individual the number of copies proportional to its quality. In other words, the quality of an individual determines the amount of resources this individual gets.

The idea of sharing was developed further by Goldberg and Richardson (1987) who proposed a practical scheme that was able to preserve multiple niches by defining the neighborhood of each individual by the *sharing function*. In this scheme each individual shares a niche with each individual that is within a certain range from either its genotype or the phenotype. The effect may decrease with the distance and completely vanishes for distances greater than a certain threshold σ_{share}.

Deb and Goldberg (1989) calculated the value of the sharing threshold σ_{share} from the desired number of niches for dividing the space into a number of equally sized hyperspheres. Their study concluded that the performance of crowding is inferior to that of sharing due to that crowding was unable to maintain more than two different optima. In spite of quite pessimistic results with the original crowding, the deterministic crowding of Mahfoud (1992) has been later shown to be able to maintain multiple peaks and thus compete with fitness sharing.

One of the drawbacks of fitness sharing is that it experiences difficulty with preservation of optima that are close to each other. Moreover, it is quite difficult to estimate the number of niches. The estimation of the number of niches is also important for methods based on crowding. Another important issue is that the effects of fitness sharing are not well understood because sharing directly changes the fitness values used for selection.

On the other hand, fitness sharing seems to be very stable and capable of preserving all optima for long periods of time. The stability of sharing for the case with two niches was studied by Horn (1993). Fitness sharing is also more sensitive to the fitness function.

3.3 Niching by Spatial Separation

There are two reasons why spatial separation should be desirable in genetic and evolutionary computation. One reason is that in nature the populations are actually divided in a number of subpopulations that (genetically) interact only rarely or do not interact at all. Another reason is that separating a number of subpopulations allows an effective parallel implementation and is therefore interesting from the point of view of computational efficiency. This section reviews and discusses niching methods based on spatial separation. The use of spatial separation in combination with probabilistic model-building genetic algorithms is discussed.

Spatial separation localizes competition by introducing some sort of geographical location of each individual. Unlike in the fitness sharing, in the spatial separation the location of each individual does not depend on its genotype or phenotype. Amount of information exchange between groups of individuals from different locations is controlled by some strategy and may depend on the distance or the relationship between the locations.

Much work in spatial separation was inspired by the shifting balance theory (Wright, 1968) and the theory of punctuated equilibria (Eldredge & Gould, 1972). One approach is to divide the population into a number of subpopulations. Each subpopulation evolves on its own island and individuals migrate between the islands at certain rate. In this way, the genetic material is exchanged within each of the subpopulations often while its flow to other subpopulations is reduced. This approach was studied by Grosso (1985), inspired mainly by the theory of Wright, and by Cohoon, Hegde, Martin, and Richards (1987), whose work is primarily inspired by the theory of Eldredge. The second approach is to introduce some kind of distance metric in the population and force local competition and mating. This approach was studied by Gorges-Schleuter (1989), Collins and Jefferson (1991), Davidor (1991), Mühlenbein (1991), and others.

The use of spatial separation was also studied in context of the probabilistic model-building genetic algorithms (Pelikan & Goldberg, 2000b) as a niching and diversity preservation tool. Pelikan and Goldberg divided the population of selected parents in each generation into a number of clusters. A mixture of Gaussians was used to separate the solutions in the selected population by using the k-means clustering method. Recombination in each cluster was done separately and the offspring of each cluster was given a proportion of the new population of the size proportional to their average fitness.

When using probabilistic models, the separation can be directly encoded by using mixture probabilistic models. A special variable in the model is reserved for an identification of the group to which the individual belongs. The interactions or parameters of local densities for each variable then may depend on the value of this one variable. Various models such as Bayesian multinets (Geiger & Heckerman, 1996) and models with hidden variables can be used. In this paper, however, we must deal with a number of niches that can be exponential in the number of variables. Even though this implies exponentially sized populations, one can use the fact that the model itself preserves diversity quite well by that it makes many independence assumptions and uses these to generate new solutions. Only little extra pressure toward diversity preservation is then required.

3.4 BOA using Restricted Tournament Replacement

In hierarchical BOA we use a niching method that is very similar to crowding and restricted tournament selection. The replacement is localized by selecting a sub-set of the original population for each new offspring and letting the offspring compete with the most similar member of this subset. If the new offspring is better, it replaces the corresponding individual. The measure of similarity can be based on either the genotype or the phenotype. Since the generation of a probabilistic model in the BOA does not encourage using a steady state genetic algorithm, we incorporate niching in the replacement step of a traditional BOA. Because the RTS is in fact used as a replacement technique and not as a primary source of the selection pressure, we call the method the *restricted tournament replacement* (RTR).

In the BOA with RTR, promising solutions are first selected from the current population and a Bayesian network is constructed as their model. The built model is then used to create new solutions. However, the new solutions are not automatically added into the original population, replacing random or the worst solutions. Instead, for each new solution we first select a subset of the original population and then let the new individual compete with the closest individual in this selected subset. The winner takes the corresponding slot in the population. At the end, some of the new individuals will be included in the new population and some will be discarded. The process starting with the selection is repeated until the termination criteria are met. It is easy to see that this model fits our pseudo-code presented first in Pelikan, Goldberg, and Cantú-Paz (1998) which is also provided in Figure 1 in Section 2.2 of this paper.

One of the reasons for using RTR as a niching strategy for the BOA in this work is that it is easily incorporated into the replacement process and does not affect modeling. With fitness sharing, the input to the model would change and it would get harder to predict the behavior of the algorithm. Another reason for using RTR is that this method was originally intended to be a part of a competent genetic algorithm, in particular the messy genetic algorithm (Goldberg, Korb, & Deb, 1989).

It is important to set the size of the subsets that are selected to incorporate each new individual into the original population. The size of these subsets is called a *window size*. A window size should be proportional to the number of niches even though big populations can allow powerful niching even with smaller window sizes. We have tried a number of settings on various difficult problems presented in the next section. Even though for almost all problems, a constant window size $w = 20$ worked very well, for the most difficult problems, increasing the window size proportionally to the size of the problem has significantly improved the performance.

A window size proportional to the size of the problem can be supported by the following argument. For correct decision making on a single level, the population size must grow proportionally to the problem size (Pelikan, Goldberg, & Cantú-Paz, 2000a). To maintain a certain number of niches, one must lower-bound the size of each niche by a certain constant. Therefore, a population size proportional to the problem size allows for maintenance of the number of niches proportional to the problem size. The number of niches that RTR can maintain is proportional to the window size. Therefore, the window size growing linearly with the size of the problem is the strongest niching one can afford without increasing population sizing requirements.

The next section presents hierarchical problems used in this paper to test hierarchical BOA. Hierarchical trap problems are introduced. Subsequently, we describe our experiments and provide empirical results. Finally, we discuss the results, summarize and conclude the paper, and outline future work.

4 Test Problems

Since the primary purpose of this paper is to show that hierarchical BOA can solve problems that are hierarchical in nature, three test problems are hierarchical. To show that the method is capable of discovering multiple optima of highly multimodal problems, we also test the algorithm on a highly multimodal problem of 5 concatenated bipolar functions of order 6.

The following section discusses the design and framework for building hierarchical problems. Subsequently, three hierarchical problems are defined. The first problem is hierarchical if-and-only-if (HIFF) function proposed by Watson, Hornby, and Pollack (1998), where on each level the neighboring building blocks are required to be homogeneous and of the same value. This leads to two global optima, one is the string of all ones and one is the string of all zeroes. Section 4.3 introduces another class of hierarchical functions called hierarchical traps. Hierarchical traps are inspired by the *tobacco road functions* (Goldberg, 1997; Goldberg, 1998; Pelikan & Goldberg, 2000c). They use basic deceptive functions on each level to make the problem even harder. The algorithm is deceived to the local optimum on each level until the top level is reached. The top level finally determines the real global optimum. Hierarchical traps have only one global optimum, in the string of all ones. We construct two hierarchical trap functions that we use in our experiments. Section 4.4 introduces remaining test functions.

4.1 Hierarchically Decomposable Functions

Hierarchically decomposable functions (HDFs) (Watson, Hornby, & Pollack, 1998) are a subclass of general additively decomposable functions (Pelikan, Goldberg, & Cantú-Paz, 1998). HDFs are defined on multiple levels where the input to each level is based on the solutions found on lower levels. The *fitness contribution* of each building block is separated from its *interpretation* (meaning) when it is used as a building block for constructing the solutions on a higher level. The overall fitness is computed as the sum of fitness contributions of each building block.

Partial solutions on each level can be found by decomposing the problem into a number of subproblems of bounded difficulty. On each level, the problem is defined by using partial solutions from a lower level as basic building blocks. In spite of bounded difficulty of HDFs on each level, a hierarchical function can contain interactions of order equal to the size of the problem. Bounded difficulty on each level of the hierarchy makes HDFs solvable in polynomial time even though the problem is very difficult when viewed on a single level. It is important to note that hierarchical problems of bounded difficulty are a strictly more difficult class of problems than problems of bounded difficulty on a single level. This observation is very important for interpreting our results.

A hierarchically decomposable function is defined by its structure in the form of a tree with one-to-one mapping between the leaves and the variables in a problem, and two sets of functions: (1) the interpretation functions and (2) the contribution functions. The structure defines which blocks of interpretations to interpret to the next level and how, and which blocks contribute to the overall fitness on this level. The interpretation functions define how we interpret solutions from lower levels to become inputs of both the contribution and interpretation functions on a higher level. The contribution functions define how much certain blocks of interpretations on each level contribute to the overall fitness.

The difficulty of hierarchical functions depends on the underlying structure as well as the contribution and interpretation functions. The hierarchical if-and-only-if (HIFF) function (Watson, Hornby, & Pollack, 1998) uses the "if and only if" function on each level. More difficult functions have been proposed (Goldberg, 1997; Goldberg, 1998; Pelikan & Goldberg, 2000c), where functions deceive the algorithms to a local optimum on each level. Only at the top level it becomes clear which optimum is the global one. This feature makes the problem much more challenging.

In this paper, only simple structures such as balanced binary and ternary trees are used. The contribution of each subfunction on each level will be multiplied by a certain factor so that the contributions on all levels are of the same magnitude. We do not present the general definition of the hierarchically decomposable problems and only describe the test problems we used. For more detailed definition of what we mean by hierarchically decomposable functions, please see Pelikan and Goldberg (2000c).

4.2 Hierarchical If-and-Only-If (HIFF)

The structure of the HIFF is a balanced binary tree. By $height(x)$ we denote the distance from the node x in the tree to one of its antecedent leaves. Since the tree is balanced, the height is well-defined. Each leaf contributes to the fitness by 1. Each parent node x contributes to the overall fitness by $2^{height(x)}$ if and only if the interpretations of its children are both either 0 or 1. Otherwise, the contribution is 0. The two symbols are interpreted to their parent on the next level as 0 in case they are both 0's, 1 in case they are both 1's, and '-' otherwise. The leaves of the tree get as input the input string with no change.

Let us illustrate this on an example. To follow the example, see Figure 4.2. Let the input string be $X = 00001101$. Each leaf corresponds to the fitness by 1, which sums up to a total contribution of 8 at the bottom level. On the next level, there are 4 parents with children $(0,0)$, $(0,0)$, $(1,1)$, and $(0,1)$, respectively. Each of the first three parents contributes to the fitness by 2 (the height is 1 and thus the contribution is $2^1 = 2$), and the last parent does not contribute to the fitness at all, since the values of its children are neither $(0,0)$ nor $(1,1)$. The pairs of children interpret to values 0, 0, 1, and '-'. Therefore, on the next level, there are two parents with pairs of children $(0,0)$ and $(1,\text{-})$, respectively. The first parent contributes to the fitness by 4 and the second one has no contribution. The pair $(0,0)$ interprets to 0 and the pair $(1,\text{-})$ interprets to '-'. On the next level, there is only one parent with a pair of children $(0,\text{-})$. This parent does not contribute to the overall fitness. We do not have to interpret the symbols to the next level anymore, because this will not be used (we already considered the root). The overall fitness of $X = 00001101$ is thus $f(X) = 8 + 2 + 2 + 2 + 4 = 18$.

4.3 Hierarchical Trap Functions

Hierarchical trap functions use a balanced k-ary tree as the underlying structure, where $k \geq 3$. However, as we are interested in a scale-up of our algorithm and the problem sizes must grow as powers of k, it is reasonable to set k to only a small value. The interpretation functions interpret blocks of all 0's and 1's to 0 and 1, respectively, similarly to the HIFF. Everything else is interpreted into '-'.

Each contribution function is a function of unitation, i.e. its value depends only on the number of ones in the input string. If there is any '-' in the input to this function, it simply returns 0. If the input is composed of u ones and $k - u$ zeroes (k bits total), the output of the trap function is determined by

$$f(u) = \begin{cases} f_{high} & \text{if } u = k \\ f_{low} - u\frac{f_{low}}{k-1} & \text{otherwise} \end{cases} .$$

See Figure 4 for a graph of the trap function. The values of f_{high} and f_{low} define the heights of the two peaks. The function is fully deceptive whenever f_{high} is greater than f_{low} within some

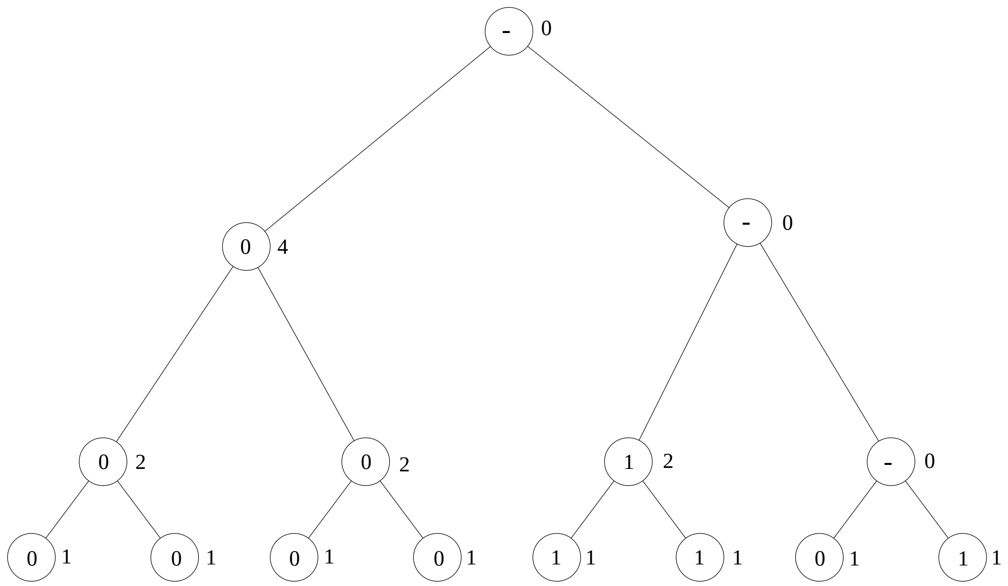


Figure 3: An example of HIFF for a string $X = 00001101$. The interpretations are displayed by circles. The contribution of each node is shown on the right side of the circle with the interpretation in the node. The fitness of this input string is 18.

proportion depending on the order k of the function. That means that any schemata of order lower than k bias the search to the local optimum of all zeroes. For sufficient conditions on deception, please see Deb and Goldberg (1994). The trap with equal peaks $f_{high} = f_{low}$ is not in fact deceptive because both optima are equal and therefore the schemata bias the search toward one of the global optima (the string of all zeroes). The trap attracts hillclimbers to strings with all zeroes unless they start in the string of all ones. Analogically as in the HIFF, contributions on each level are multiplied by k to the height of the level in order to make the overall contribution of all levels the same.

In both our functions, the underlying structure is a ternary tree (i.e., $k = 3$) and the leaves do not contribute to the overall fitness. For all non-leaf nodes x of the first hierarchical trap except for the root, the contribution is computed by a trap function with equal peaks $f_{high} = f_{low} = 1$ multiplied by $3^{height(x)}$. The contribution of the root node is given by the trap function with $f_{high} = 1$ and $f_{low} = 0.9$ multiplied by $3^{height(root)}$. Multiplication of the contribution of each node by 3 to the height of the node results in that the overall contribution of each level has the same magnitude. In this fashion, the function biases the search to the solution of all zeroes on each but the top level. However, the optimum is in the string of all ones. The top level is also deceptive which makes the problem even harder. See Figure 5 for a simple example of this function defined on two non-leaf levels. The above function is denoted by H-Trap1 in further text.

In the second function the bias toward solutions with many zeroes is made even stronger by making the peak f_{low} higher than the other peak everywhere except for the root. The contribution function in the root remains the same as before, with the peaks $f_{high} = 1$ and $f_{low} = 0.9$. To keep the global optimum in the string of all ones, the following inequality must be satisfied:

$$(k-1)(f_{low} - f_{high}) < f'_{high} - f'_{low},$$

where f_{low} and f_{high} denote the peaks for the levels below the root, f'_{low} and f'_{high} denote the peaks for the root, and k is the number of non-leaf levels. We set the $f_{high} = 1$, $f_{low} = 1 + 0.1/k$, $f'_{high} = 1$,

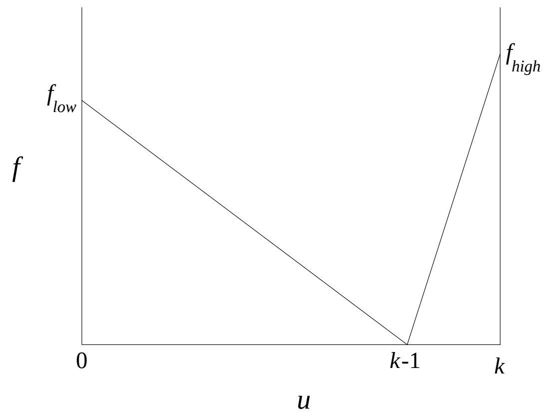


Figure 4: Trap function of order k.

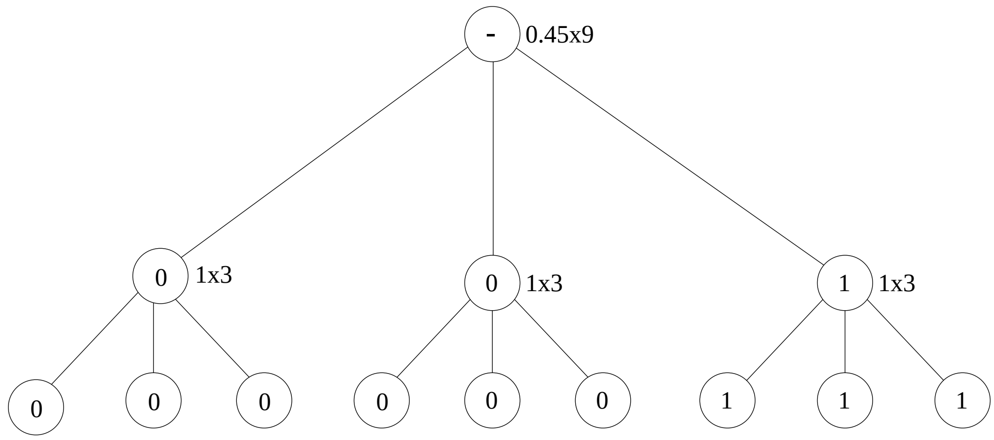


Figure 5: An example of the hierarchical trap function for a string $X = 000000111$. The interpretations are displayed by circles. The contribution of each node is shown on the right side of the circle with the interpretation in the node. The fitness of this input string is 13.05.

and $f'_{low} = 0.9$. This assignment satisfies the above inequality. The function is denoted by H-Trap2 in further text.

The HIFF function does not bias the search toward either global optimum. Unlike the HIFF, both hierarchical trap functions H-Trap1 and H-Trap2 bias the search toward solutions with all zeroes on all levels. However, the actual global optimum is in the string of all ones. Therefore, the functions are very difficult to solve and without effective linkage learning required to preserve the local optima on each level and niching required to preserve alternative partial solutions until solving the problem on the highest level, the algorithm cannot reach the global optimum. Of course, having information about the problem structure in advance would allow a problem-specific recombination operator in combination with niching to solve the problem, too. However, once we have the information about the problem structure of this problem, the problem becomes quite easy.

4.4 Other problems

The bipolar deceptive function of order 6 is constructed by concatenating a number of bipolar subfunctions of order 6 (Deb, Horn, & Goldberg, 1992). The bipolar function of order 6 is constructed from a deceptive function of order 3 defined on binary vectors $X = (X_0, X_1, X_2)$ of order 3 as

$$f_{3dec}(u) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise} \end{cases},$$

where u is the number of one's in the input vector (string) X. The bipolar function of order 6 is defined on binary vectors of length 6 as

$$f_{6bip}(u) = f_{3dec}(|3 - u|).$$

The bipolar fitness function is then defined as a concatenation of a number of bipolar subfunctions:

$$f_{bipolar}(X) = \sum_{i=0}^{n/6-1} f_{6bip}(X_i + X_{i+1} + X_{i+2} + X_{i+3} + X_{i+4} + X_{i+5}),$$

where $X = (X_0, \ldots, X_{n-1})$ is the input string.

5 Results

The primary goal of our experiments was to show that the BOA with an appropriate niching method (RTR) is capable of solving difficult hierarchical problems efficiently and it scales up subquadratically with the problem size. The second goal was to relate the empirical results on hierarchical problems to the population sizing and convergence theory (Pelikan et al., 2000a). Yet another goal was to show that the BOA with RTR is capable of discovering multiple optima in highly multimodal functions and maintain these for long periods of time. This section starts by describing our experiments. Subsequently, the experiments focusing on the scale-up behavior are presented. The results are discussed and related to theory. Finally, an example run of hierarchical BOA on a highly multimodal problem is analyzed to show how hierarchical BOA discovers all global optima of the problem.

To show how the algorithm scales up, we performed tests on each function with varying problem size. For each problem size, we required that the algorithm find the global optimum in all 30 independent runs. The performance was measured by an average number of fitness evaluations until the optimum was found. The population size was determined empirically to maximize the performance (i.e., to minimize the number of fitness evaluations until the optimum was found). This was usually achieved with the minimal population size required to converge in all 30 runs. Sometimes, the optimal population size was slightly bigger than the minimal one. Binary tournament selection was used in all experiments. Except for the experiments we have done to gain experience with the restricted tournament replacement, we always use the window size equal to the problem size, i.e. $w = n$. We used decision graphs to represent conditional probabilities in the model and construct the model. Prior distribution of models is biased toward simpler models (Pelikan et al., 2000).

The results of our experiments on the HIFF, H-Trap1, and H-Trap2 functions are shown in Figure 6. In all three cases the algorithm scales up subquadratically. On the left-hand side of the figure, the graphs in arithmetic scale display the growth of the number of fitness evaluations with respect to the size of the problem for all three problems.

Theory of population sizing and time to convergence for the BOA on separable problems of bounded difficulty (Pelikan, Goldberg, & Cantú-Paz, 2000a) can be used to estimate time to convergence of hierarchical BOA on hierarchical problems assuming that hierarchical BOA finds a correct model on each level. Theory suggests that a single level of the hierarchical problem can be solved in about $O(n^{1.5})$ fitness evaluations. The number of levels in all three hierarchical problems grows with a logarithm of the problems size, i.e. $O(\log n)$. Thus, the overall time to convergence should grow as $O(n^{1.5} \log n)$. In fact, we tried to approximate our experimental results with a function of the form $an^b \log n$, where a and b were determined by the least mean square error method. In all cases, the optimal exponent b was from about 1.55 to about 1.66 which is very close to what we would expect (1.5) according to the theory. The fit is very good and matches also the slopes in the log-log scaled graphs very accurately.

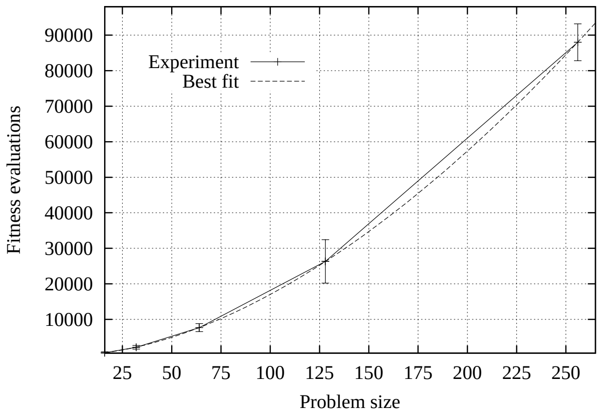
On the right-hand side, the log-log scaled graphs including the slopes between neighboring points are shown. A linear function in this scale is a polynomial of the degree equal to the slope of the curve. That means that, for instance, linear functions would have the slope of 1 and quadratic functions would have the slope of 2. To show that the number of fitness evaluations grows at most polynomially with the problem size, the curves in log-log scale must not grow faster than linearly. The points must lie on a straight line and therefore the slopes must be constant. In our experiments, we see that the slopes in fact *decrease* with the problem size. This is the effect of the logarithm in the expected number of fitness evaluations. The slopes are also in agreement with the approximations of the growth of fitness evaluations discussed in the above paragraph.

The simple genetic algorithm with fixed crossover is not able to optimize hierarchical functions without making sure that interacting genes are close to each other. Under the assumption of tight linkage, the simple genetic algorithm with good niching should work quite well. The algorithm presented in Watson (2000) is able to solve the HIFF problem even for interacting genes spread throughout the strings. However, Watson's algorithm requires $O(n^2 \log n)$ fitness evaluations for a problem of size n which is more than is required by our algorithm.

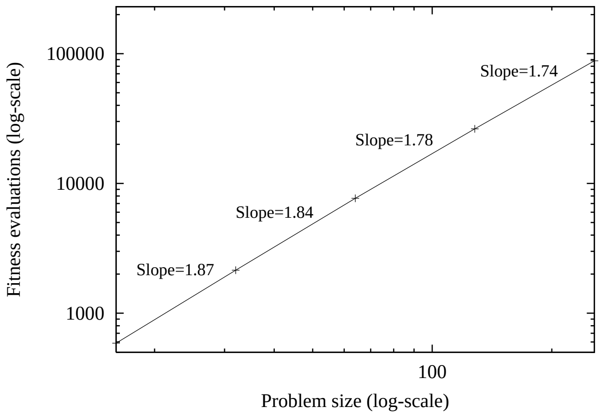
To show the ability of the algorithm to discover multiple optima, we performed a single run on a bipolar function of size $n = 30$ with a sufficiently big population size and recorded the number of copies of each global optimum in the population. The results of this experiment are shown in Figure 7. The population size was set to 1500. We have performed a number of experiments with varying parameters with a very similar result.

There are two important observations. First, the algorithm is able to discover all 32 global optima of the function quite fast and maintain these very stably. In order to find the optima, it is necessary to learn the linkage well and use the linkage information to recombine the solutions effectively. Once the optima are discovered, they gradually take over a part of the population. After only a couple of generations, almost every replacement results in replacing the same optimum, since this is the closest solution one can find and it occupies big enough portion of the population to be selected by the RTR with a high probability.

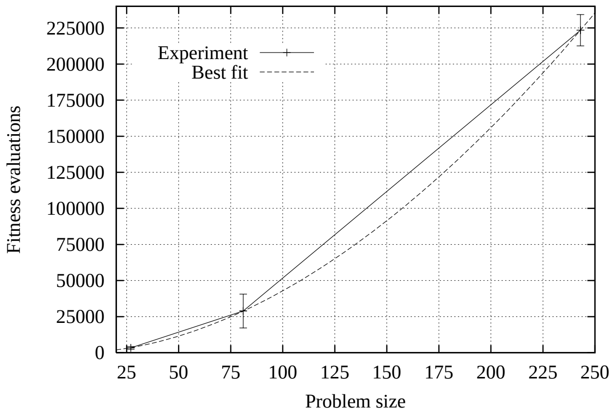
The second important observation is that even though all global optima have the same fitness value, they occupy quite different proportions of the population, from about 1.27% to about 5.53%. This confirms the intuition that, unlike fitness sharing, the methods based on crowding are not very sensitive to the fitness values. They are able to maintain a number of alternatives but the total space occupied by each alternative is not proportional to its fitness. We do not see this as a disadvantage, even though allocation of resources proportional to the fitness can become important on some problems.



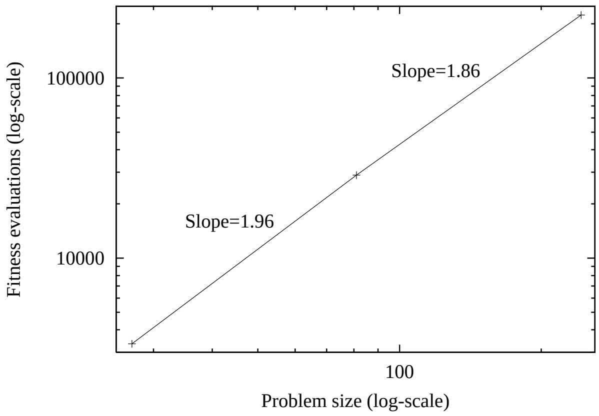
(a) HIFF: Arithmetic scale.



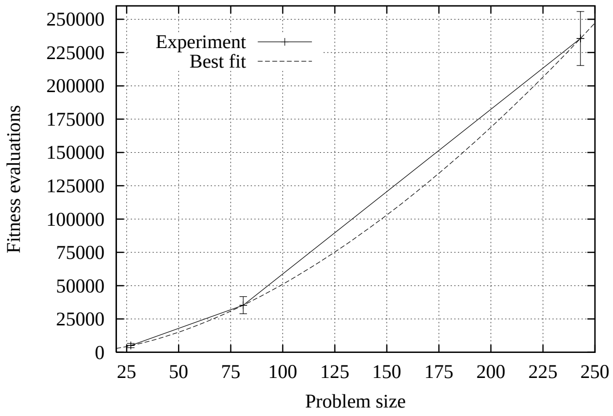
(b) HIFF: Log scale.



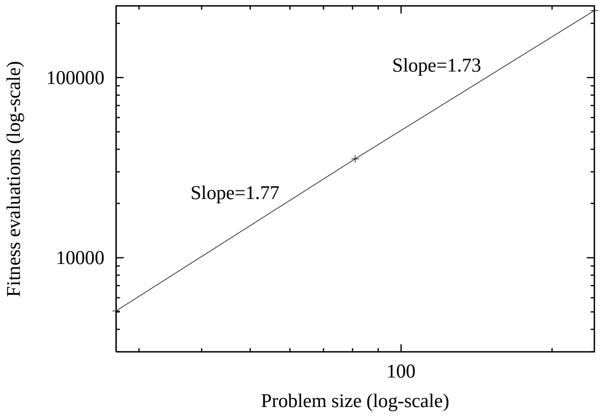
(c) H-Trap1: Arithmetic scale.



(d) H-Trap1: Log scale.



(e) H-Trap2: Arithmetic scale.



(f) H-Trap2: Log scale.

Figure 6: Results on the hierarchical functions.

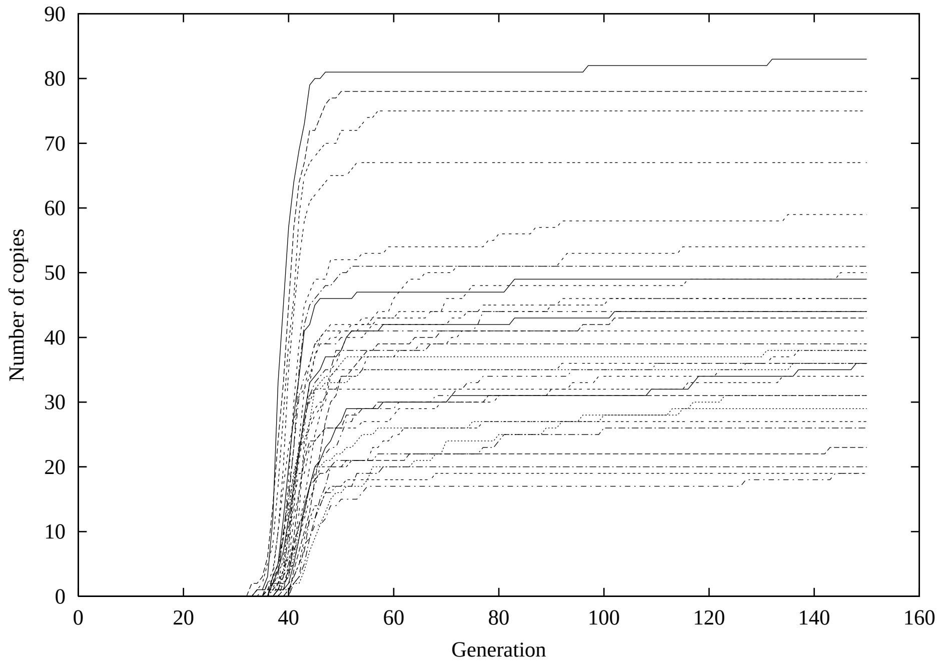


Figure 7: Number of copies of different global optima of the bipolar function. There are 32 optima in this function and all 32 are multiply represented at the end of the run.

6 What's Next?

Binary problems with a fixed number of decision variables are a very interesting, difficult, and general class of problems. However, many problems are defined in different domains such as vectors of real numbers, program codes, communication networks, geometric shapes, and rules. More work must be done in order to widen the applicability of the hierarchical BOA. There are two basic approaches to extend the algorithm to different problem domains. The first approach is to map potential solutions to binary strings of fixed length and use the original algorithm on a modified domain. To use the above approach on some problems, discretization or bounds on the solution size and accuracy may be necessary. For example, we would have to discretize vectors of real numbers before mapping them onto a finite domain of fixed-length binary strings.

Another approach is to adapt model building and utilization to a new problem domain. One must decide what class of distributions to use for each problem domain and how to learn and utilize distributions of this class. However, such a modification may not be a straightforward one and the applicability and efficiency of the algorithm may suffer.

Each approach has its advantages and disadvantages. Mapping solutions to fixed-length binary strings allows us to use the same method to solve the new problem. We can rely on theory and experience from using the BOA on fixed-length binary problems. On the other hand, mapping solutions from one space to another one may modify the problem difficulty (in either direction) and introduce additional work for the user to define and implement the problem. Modifying model building and utilization changes basic properties of the algorithm and may reduce the class of problems the algorithm can solve efficiently and reliably.

Solving artificial hierarchical problems is very important for binding the class of problems the algorithm is able to solve efficiently and reliably, but real-world problems must be tackled to show that the approach is also able to solve difficult real-world problems. Many real-world problems are simple and do not require such a powerful problem solver. On the other hand, some problems may not be solvable in this fashion.

7 Summary and Conclusions

The paper presented the *hierarchical Bayesian optimization algorithm* which combines linkage learning, efficient representation of partial solutions, and powerful niching. Additionally, *hierarchical trap problems* that are deceptive on each level of the hierarchy were designed. The hierarchical BOA was tested on a number problems and the results confirmed that the algorithm scales up subquadratically even on difficult hierarchical problems and is able to discover multiple optima of highly multimodal problems. The empirical results were in agreement with recent theory.

The paper takes another important step toward increasingly competent genetic algorithms by providing an algorithm that is able to solve problems on a single level as well as multiple levels. It emphasizes the importance of solving separable problems on a single level by showing that we need not modify much to successfully move from a single level to hierarchies. To solve hierarchically decomposable problems quickly, accurately, and reliably, a combination of niching, linkage learning, and efficient representation of partial solutions is necessary.

To learn the linkage, hierarchical BOA uses Bayesian networks to model promising solutions and to generate the new ones. To efficiently represent partial solutions, decision graphs are used to represent local densities in a model. To assure powerful niching, the restricted tournament replacement is used.

Separable deceptive problems of bounded difficulty are extended to multiple levels. The designed hierarchical trap problems that are deceptive on each level are intractable by local search methods and can be used as a benchmark for other optimization algorithms. Despite hierarchical problems that are more difficult and challenging than separable problems, hierarchical BOA can solve these problems very efficiently and reliably and it scales up subquadratically with the size of a problem. Population sizing and convergence theory can be used to approximate the behavior of the algorithm both on single-level and hierarchical problems.

Hierarchical BOA should be applicable to real-world problems without problem specific knowledge ahead of time. This takes us closer to the promised land of robustness, that has long been associated with GAs but rarely delivered.

8 Acknowledgments

The authors would like to thank Martin Butz, Erick Cantú-Paz, Clarissa Van Hoyweghen, Fernando Lobo, Franz Rothlauf, and Kumara Sastry for many useful discussions and valuable comments.

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-1-0163. Research funding for this work was also provided by the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. Martin Pelikan was partially supported by grant VEGA 1/7654/20 of the Slovak Grant Agency. The U. S. Government is authorized to reproduce and distribute reprints for Government 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 or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U. S. Government.

References

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In Eschelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 101–111). San Francisco, CA: Morgan Kaufmann.

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

Baluja, S., & Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In Fisher, D. (Ed.), *Proceedings of the 14th International Conference on Machine Learning* (pp. 30–38). San Francisco: Morgan Kaufmann.

Booker, L. B. (1982). *Intelligent behavior as an adaptation to the task environment.* Doctoral dissertation, The University of Michigan. (University Microfilms No. 8214966).

Bosman, P. A. (2000). Continuous iterated density estimation evolutionary algorithms within the idea framework. Personal communication.

Bosman, P. A. N., & Thierens, D. (1999). Linkage information processing in distribution estimation algorithms. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Volume I (pp. 60–67). Orlando, FL: Morgan Kaufmann Publishers, San Fransisco, CA.

Cavicchio, Jr., D. J. (1970). *Adaptive search using simulated evolution.* Doctoral dissertation, University of Michigan, Ann Arbor, MI. (University Microfilms No. 25-0199).

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

Cohoon, J. P., Hegde, S. U., Martin, W. N., & Richards, D. (1987). Punctuated equilibria: A parallel genetic algorithm. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 148–154). Hillsdale, NJ: Lawrence Erlbaum Associates.

Collins, R. J., & Jefferson, D. R. (1991). Selection in massively parallel genetic algorithms. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 249–256). San Mateo: CA: Morgan Kaufmann.

Culberson, J. C. (1992). *Genetic invariance: A new paradigm for genetic algorithm design.* Unpublished manuscript.

Davidor, Y. (1991). A naturally occuring niche and species phenomenon: The model and first results. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 257–263). San Mateo, CA: Morgan Kaufmann.

De Bonet, J. S., Isbell, C. L., & Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. In Mozer, M. C., Jordan, M. I., & Petsche, T. (Eds.), *Advances in neural information processing systems*, Volume 9 (pp. 424). The MIT Press, Cambridge.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation, University of Michigan, Ann Arbor. (University Microfilms No. 76-9381).

Deb, K., & Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 42–50). San Francisco, CA: Morgan Kaufmann.

Deb, K., & Goldberg, D. E. (1994). Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, *10*, 385–408.

Deb, K., Horn, J., & Goldberg, D. E. (1992). *Multimodal deceptive functions* (IlliGAL Report No. 92003). Urbana, IL: University of Illinois at Urbana-Champaign.

Eldredge, N., & Gould, S. (1972). Punctuated equilibria: an alternative to phyletic gradualism. San Francisco, CA: Freeman & Company.

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

Fonseca, C. M., & Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 416–423). San Mateo, CA: Morgan Kaufmann.

Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical models* (1 ed.). (pp. 421–459). Cambridge, MA: MIT Press.

Geiger, D., & Heckerman, D. (1996). Beyond Bayesian networks: Similarity networks and Bayesian multinets. *Artificial Intelligence*, *82*, 45–74.

Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning. *Dissertation Abstracts International*, *44* (10), 3174B. Doctoral dissertation, University of Michigan.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Goldberg, D. E. (1997, November). *The design of innovation: Lessons from genetic algorithms.* Unpublished manuscript.

Goldberg, D. E. (1998, June 15). Four keys to understanding building-block difficulty. Presented in Projet FRACTALES Seminar at I.N.R.I.A. Rocquencourt, Le Chesnay, Cedex.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3* (5), 493–530.

Goldberg, D. E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 41–49). Hillsdale, NJ: Lawrence Erlbaum Associates.

Gorges-Schleuter, M. (1989). ASPARAGOS: An asynchronous parallel genetic optimization strategy. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 422–428). San Mateo, CA: Morgan Kaufmann.

Grosso, P. B. (1985). *Computer simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model.* Unpublished doctoral dissertation, The University of Michigan. (University Microfilms No. 8520908).

Harik, G. (1994, May). *Finding multiple solutions in problems of bounded difficulty* (IlliGAL Report No. 94002). Urbana, IL: University of Illinois at Urbana-Champaign.

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

Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1998). The compact genetic algorithm. In *Proceedings of the International Conference on Evolutionary Computation 1998 (ICEC '98)* (pp. 523–528). Piscataway, NJ: IEEE Service Center.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Hollstein, R. B. (1971). *Artificial genetic adaptation in computer control systems.* Doctoral dissertation, University of Michigan. (University Microfilms No. 71-23,773).

Horn, J. (1993). Finite Markov chain analysis of genetic algorithms with niching. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 110–117). San Mateo, CA: Morgan Kaufmann. Also TCGA No. 04447, IlliGAL Report Number 93002.

Horn, J., & Nafpliotis, N. (1993, July). *Multiobjective optimization using the niched pareto genetic algorithm* (IlliGAL Report No. 93005). Urbana, IL: University of Illinois at Urbana-Champaign.

Mahfoud, S. W. (1992). Crowding and preselection revisited. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature, 2* (pp. 27–36). Elsevier Science.

Mauldin, M. L. (1984). Maintaining diversity in genetic search. In Brachman, R. J. (Ed.), *Proceedings of the National Conference on Artificial Intelligence* (pp. 247–250). Austin, TX: William Kaufmann.

Mengshoel, O. J., & Goldberg, D. E. (1999). Probabilistic crowding: Deterministic crowding with probabilisitic replacement. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Volume I (pp. 409–416). Orlando, FL: Morgan Kaufmann Publishers, San Fransisco, CA.

Mühlenbein, H. (1991). Evolution in time and space-The parallel genetic algorithm. In Rawlins, G. J. E. (Ed.), *Foundations of Genetic Algorithms* (pp. 316–337). San Mateo, CA: Morgan Kaufmann.

Mühlenbein, H. (1997). The equation for response to selection and its use for prediction. *Evolutionary Computation*, *5*(3), 303–346.

Mühlenbein, H., Mahnig, T., & Rodriguez, A. O. (1998). *Schemata, distributions and graphical models in evolutionary optimization.* Submitted for publication.

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

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

Pelikan, M., & Goldberg, D. E. (2000a). *A comparative study of scoring metrics in the Bayesian optimization algorithm: Minimum description length and Bayesian-Dirichlet.* Unpublished technical report.

Pelikan, M., & Goldberg, D. E. (2000b). *Genetic algorithms, clustering, and the breaking of symmetry* (IlliGAL Report No. 2000013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., & Goldberg, D. E. (2000c). *Hierarchical problem solving by the Bayesian optimization algorithm* (IlliGAL Report No. 2000002). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1998). *Linkage problem, distribution estimation, and Bayesian networks* (IlliGAL Report No. 98013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000a). *Bayesian optimization algorithm, population sizing, and time to convergence* (IlliGAL Report No. 2000001). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000b). Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation*, *8*(3), 311–341.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2000). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*. In press.

Pelikan, M., Goldberg, D. E., & Sastry, K. (2000). *Bayesian optimization algorithm, decision graphs, and Occam's razor* (IlliGAL Report No. 2000020). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In Roy, R., Furuhashi, T., & Chawdhry, P. K. (Eds.), *Advances in Soft Computing - Engineering Design and Manufacturing* (pp. 521–535). London: Springer-Verlag.

Perry, Z. A. (1984). Experimental study of speciation in ecological niche theory using genetic algorithms. *Dissertation Abstracts International*, *45*(12), 3870B. (University Microfilms No. 8502912).

Schaffer, J. D. (1984). *Some experiments in machine learning using vector evaluated genetic algorithms.* Doctoral dissertation, Vanderbilt University, Nashville, Tennessee. (University Microfilms No. 85-22492).

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). San Mateo, CA: Morgan Kaufmann.

Watson, R. A. (2000). Analysis of recombinative algorithms on a non-separable building-block problem. *Foundations of Genetic Algorithms*. In printing.

Watson, R. A., Hornby, G. S., & Pollack, J. B. (1998). Modeling building-block interdependency. In Eiben, A. E., Bäck, T., Schoenauer, M., & Schwefel, H.-P. (Eds.), *Parallel Problem Solving from Nature, PPSN V* (pp. 97–106). Berlin: Springer Verlag.

Wright, S. (1968). *Evolution and the genetics of populations: a treatise.* University of Chicago Press.