

---

# Robust and Scalable Black-Box Optimization, Hierarchy, and Ising Spin Glasses

**Martin Pelikan**

Computational Laboratory (CoLab), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland

**David E. Goldberg**

Illinois Genetic Algorithms Laboratory (IlliGAL), University of Illinois at Urbana-Champaign, Urbana, IL

**Jiri Ocenasek**

Computational Laboratory (CoLab), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland

**Simon Trebst**

Computational Laboratory (CoLab) and the Institute of Theoretical Physics, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland

---

## Abstract

One of the most important challenges in computational optimization is the design of advanced black-box optimization techniques that would enable automated, robust, and scalable solution to challenging optimization problems. This paper describes an advanced black-box optimizer—the hierarchical Bayesian optimization algorithm (hBOA)—that combines techniques of genetic and evolutionary computation, machine learning, and statistics to create a widely applicable tool for solving real-world optimization problems. The paper motivates hBOA, describes its basic procedure, and provides an in-depth empirical analysis of hBOA on the class of random 2D and 3D Ising spin glass problems. The results on Ising spin glasses indicate that even without much problem-specific knowledge, hBOA can provide competitive or better results than techniques specialized in solving the particular problem or class of problems. Furthermore, hBOA can solve a large class of nearly decomposable and hierarchical problems for which there exists no other scalable solution.

## 1 Introduction

A black-box optimization problem may be seen as the task of finding the best solution or the optimum given (1) the set of all potential solutions and (2) a procedure for evaluating quality or performance of competing candidate solutions. That problem is a black box means that there is no knowledge about the semantics of potential solutions to the problem, the actual form of the evaluation procedure, or the relation between the two—the only way of learning about the problem is to sample candidate solutions, evaluate them, and process the results of the evaluation. Since many challenging real-world problems

can be formulated in this fashion, the design of powerful black-box optimization techniques is currently one of the most important challenges in computational optimization. Nonetheless, many common black-box optimization algorithms make strong assumptions about the problem domain and they cannot be applied successfully without introduction of additional problem-specific heuristics or expensive manual problem analysis needed to make the problem fit a specialized optimization technique. One of the most promising approaches to overcoming this difficulty is based on coupling existing optimization techniques with machine learning tools for the automatic learning of regularities in data (Larrañaga & Lozano, 2002; Pelikan, Goldberg, & Lobo, 2002; Pelikan, 2002).

The purpose of this paper is to describe an advanced black-box optimizer called the hierarchical Bayesian optimization algorithm (hBOA), which combines techniques and concepts from genetic and evolutionary computation, machine learning, and statistics to enable quick, accurate, and reliable solution to challenging real-world optimization problems. From genetic and evolutionary computation, hBOA borrows the concepts of population-based search based on selection and recombination, the creation of a number of artificial niches for diversity preservation, and the facetwise theory and design. From machine learning and statistics, hBOA borrows the techniques for learning and sampling probabilistic graphical models, which turn out to be a powerful exploration tool for automated black-box optimization.

The paper makes a thorough empirical analysis of hBOA on the class of random two- and three-dimensional Ising spin-glass problems. The results indicate that hBOA is capable of solving two-dimensional Ising spin glasses in low-order polynomial time that is competitive with techniques specialized in solving 2D Ising spin glasses. Furthermore, hBOA appears to provide efficient solution even in the 3D case, which cannot be efficiently tackled by any other method and is known to be extremely difficult.

## 2 From genetic algorithms to probabilistic models

Genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989; Mitchell, 1996) approach black-box optimization by evolving a population of candidate solutions with the operators inspired by natural evolution and genetics. Maintaining a population of solutions—as opposed to a single solution—has several advantages. Using a population allows simultaneous exploration of multiple basins of attraction. Additionally, a population allows for statistical decision making based on the entire sample of promising solutions even when the evaluation procedure is affected by external noise.

Genetic algorithms pose no significant prior restrictions on the representation of candidate solutions or the performance measure. Representations can vary from binary strings to vectors of real numbers, to permutations, to production rules, to schedules, and to program codes. Performance measures can be based on a computer procedure, a simulation, an interaction with a human, or some combination of the above. Additionally, evaluation of each candidate solution can be affected by external noise and the performance measure might be able to only compare relative performance of competing candidate solutions. Nonetheless, for the sake of simplicity, the rest of this paper assumes that solutions are represented by binary strings of fixed length and that the performance of each candidate solution is represented by a real number called *fitness*. The task is to find the binary string with the highest fitness. It is straightforward to apply all algorithms discussed in this paper to problems with any fixed-length discrete representations. However, the transition to variable-length, structured, or continuous domains is more difficult and in some cases it remains an important topic for future work.

```

Genetic algorithm (GA)
t := 0;
generate initial population P(0);
while (not done) {
    select population of promising solutions S(t);
    apply crossover+mutation to S(t) yielding offspring O(t);
    incorporate O(t) into P(t) yielding P(t+1);
    t := t+1;
};

```

Figure 1: The pseudocode of the basic genetic algorithm procedure.

This section briefly describes the basic genetic algorithm procedure. Linkage learning is then discussed as one of the important challenges for the development of competent genetic algorithms (Goldberg, 2002)—genetic algorithms that can solve hard problems quickly, accurately, and reliably. Finally, the section motivates the use of probabilistic models instead of conventional variation operators inspired by genetics.

## 2.1 Basic genetic algorithm procedure

Figure 2.1 shows the basic genetic algorithm procedure. Genetic algorithms (GAs) evolve a population of candidate solutions to a problem. The first population of candidate solutions is usually generated at random with a uniform distribution over all possible solutions. Each iteration starts by selecting a population of promising solutions from the current population based on the performance of each solution. Various selection operators can be used, but the basic idea of all selection methods is the same—make more copies of solutions that perform better at the expense of solutions that perform worse. Here we consider tournament selection, which iteratively selects one solution at a time by first choosing a random subset of candidate solutions from the current population and then selecting the best solution out of this subset. The size of the selected subsets determines the selection pressure; here we use binary tournaments where two solutions compete in each tournament.

Once the population of promising solutions has been selected, new candidate solutions are created by applying recombination (crossover) and mutation to the promising solutions. Recombination combines subsets of promising solutions by exchanging some of their parts; mutation perturbs the recombined solutions slightly to explore their immediate neighborhood. Most of the commonly used crossover operators combine partial solutions between pairs of promising solutions with a specified probability. For example, one-point crossover first randomly selects a single position in the two strings and exchanges the bits on all the positions after the selected one. On the other hand, uniform crossover exchanges bits in each position with probability 50%. The optimal value of the probability of applying crossover varies, but usually at least a half of parents undergo crossover. For binary strings, bit-flip mutation is usually used, which proceeds by flipping each bit with a fixed probability. The probability of flipping each bit is quite small, so only small changes are expected to occur. Recombination and mutation are often called variation operators because their primary function is to introduce variation and enable exploration of new candidate solutions in GAs. Recombination is the primary source of variation in most GAs; these GAs are often referred to as *selectorecombinative GAs*.

Because of the inspiration from natural evolution and genetics, terminology used in GAs also comes from these areas; candidate solutions are often referred to as individuals, problem variables whose values are to be determined to maximize the performance measure are often referred to as genes, and their values are referred to as alleles.

## 2.2 Linkage learning

While most would probably agree that it is natural evolution that enabled the creation of the broad spectrum of enormously complex biological systems, a direct inspiration from natural evolution combined with a simplification of operators that are believed to play key role in natural evolution does not guarantee a competent, widely applicable, robust, and scalable black-box optimizer. Indeed, over the past few decades many practitioners have failed at solving their problem using simple GAs with one-point or uniform crossover and bit-flip mutation, and started to ask important questions, such as “Why do GAs fail?” and “How to make GAs work better?”.

For selectorecombinative GAs, one of the most important milestones in finding answers to the above and related questions was the development of *IlliGAL GA design decomposition* (Goldberg, Deb, & Clark, 1992; Goldberg, 1994; Goldberg, 2002), also called Goldberg’s GA decomposition, which lists GA facets that must be considered to ensure that GAs can solve difficult problems in a robust and scalable manner. IlliGAL GA decomposition also helped to identify the primary reason for the failure of conventional GAs—common crossover operators such as one-point and uniform crossover are incapable of effective exploration by combining bits and pieces of promising solutions in the general case, because they either disrupt correlated chunks of solutions (building block disruption) or do not ensure effective juxtaposition of promising partial solutions (ineffective mixing) (Thierens & Goldberg, 1993; Thierens, 1995). The problems of building block disruption and ineffective mixing introduced a new direction in genetic and evolutionary computation called *linkage learning* (Goldberg, Korb, & Deb, 1989; Harik & Goldberg, 1996), where the focus is on designing recombination operators that do not disrupt important partial solutions contained in the optimum—also called building blocks (Goldberg, 2002)—but still ensure an effective mixing of partial solutions.

A number of linkage learning GAs have been designed over the past few decades (Bosman, 2003; Goldberg, Korb, & Deb, 1989; Goldberg, Deb, Kargupta, & Harik, 1993; Kargupta, 1995; Kargupta, 1996; Harik, 1997; Laumanns & Ocenasek, 2002; Munetomo & Goldberg, 1998; Harik, 1999; Ocenasek, 2002; Pelikan & Mühlenbein, 1999; Pelikan, Goldberg, & Cantú-Paz, 1999; Yu, Goldberg, & Chen, 2003). One of the most promising research directions in linkage learning derives inspiration from the work of Baluja (1994) and others, and uses a two-step procedure to replace conventional variation operators (mutation and crossover):

1. Build a probabilistic model for the population after selection.
2. Sample the built model to generate new candidate solutions.

GAs with the two-step probabilistic recombination are called probabilistic model-building GAs (PMBGAs) (Pelikan, Goldberg, & Lobo, 2002), estimation of distribution algorithms (EDAs) (Mühlenbein & Paaß, 1996), and iterated density estimation algorithms (IDEAs) (Bosman & Thierens, 2000). It is beyond the scope of this paper to give a thorough overview of PMBGAs, for a survey see Pelikan, Goldberg, and Lobo (2002), Pelikan (2002), and Larrañaga and Lozano (2002).

Clearly, whether or not the learning and sampling of a probabilistic model is going to do a better job at combining promising partial solutions than conventional recombination operators depends on what kind of models are used and how these are learned. This section closes with a simple probabilistic operator similar to uniform crossover, which will lead to a number of more complex and more widely applicable operators described in the section to follow.

### 2.3 Probability vector

Let us assume that candidate solutions are represented by binary strings of fixed length. Probably the simplest way to estimate a probability distribution over  $n$ -bit binary strings is to consider each string position independently and encode only the probabilities of a 0 and a 1 in each position of solution strings. New solutions can be generated by sampling these single-bit probabilities for each new candidate solution. Mathematically, the distribution estimate that considers each variable independently can be written as

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

where  $X = (X_1, \dots, X_n)$  is an input vector of  $n$  binary variables (note that each string position can be seen as a random variable and the bit in the position can be seen as a value of that variable). The above model forms the basis of the first PMBGA, the population-based incremental learning (PBIL) algorithm (Baluja, 1994), the univariate marginal distribution algorithm (UMDA) (Mühlenbein & Paaß, 1996), and the compact GA (cGA) (Harik, Lobo, & Goldberg, 1998).

Learning the univariate probabilistic model is simple—one only needs to compute the proportion of a 0 and a 1 for each string position or random variable. In fact, only one of the two proportions for each position suffices, because the two probabilities must sum to one. Usually, a model consists of a vector specifying the proportion of 1s for each string position; this vector is often referred to as the probability vector. Sampling the probability vector is also simple—the value in each position of a new candidate solution is generated according to the proportions of 1s and 0s in that position.

The probability vector preserves the proportions of 1s and 0s in each position of the population after selection, which is a common feature of most recombination operators including one-point and uniform crossover. Additionally, sampling the probability vector decorrelates different string positions, because the value in every string position is generated independently. As a result, bits that are superior to their competitors in the same string position are going to be propagated by selection and recombination, and they are going to take over the entire population eventually. With the probability vector, premature convergence to inferior bits can be eliminated by using populations that scale at most linearly with the number of bits in the problem (Harik, Cantú-Paz, Goldberg, & Miller, 1999; Thierens, Goldberg, & Pereira, 1998).

### 2.4 Proper Decomposition and Statistics of Higher Order

If single-bit statistics lead toward the optimum and the bits are uniformly scaled, the probability vector coupled with a selection operator of fixed selection intensity (e.g. tournament and truncation selection) and an adequate population sizing (Harik, Cantú-Paz, Goldberg, & Miller, 1997) can solve the problem in  $O(n \ln(1/\alpha))$  function evaluations (Harik, Cantú-Paz, Goldberg, & Miller, 1997; Mühlenbein & Schlierkamp-Voosen, 1993), where  $\alpha$  is the

probability of making an error in each string position. That means that even if the probability of error was fixed for all problem sizes, the probability vector can be expected to yield near linear convergence time  $O(n \log n)$ . For an arbitrary scaling of bit contributions, the number of evaluations can be bounded by  $O(n^2)$  (Thierens, Goldberg, & Pereira, 1998).

If single-bit statistics do *not* lead to the optimum, the probability vector is going to converge to only a local optimum in most cases, even for extremely large populations. To eliminate this drawback, it is sometimes necessary that the model considers higher order statistics, e.g. proportions or marginal probabilities of two or more bits. Once the model uses correct, nonmisleading statistics, and these statistics are of order bounded by a given constant  $k$ , one could show the convergence properties analogically to the case with the probability vector, yielding a quadratic or subquadratic number of evaluations until reliable convergence. We say that a model properly decomposes the problem if the statistics it considers lead toward the optimum. It is beyond the scope of this paper to discuss problem decomposition in more detail; for a thorough discussion of this and related issues, please see Goldberg (2002) and Pelikan (2002). For a complete picture, it should also be useful consult Mühlenbein, Mahnig, and Rodriguez (1999), who directly relate probabilistic models, graph theory, and a proper decomposition of optimization problems.

The next section describes the Bayesian optimization algorithm (BOA), which takes the next step and can both identify what statistics should be considered as well as learn the values of these statistics. Probabilistic models used in BOA can cover high-order and overlapping statistics.

### 3 Bayesian optimization algorithm (BOA)

The Bayesian optimization algorithm (BOA) ((Pelikan, Goldberg, & Cantú-Paz), (1998), (1999), (2000b)) evolves a population of candidate solutions by building and sampling Bayesian networks. BOA generates the initial population of candidate solutions at random with a uniform distribution over all possible solutions; however, it is sometimes possible to bias the generation of the initial population using a problem-specific heuristic (Schwarz & Ocenasek, 2000; Ocenasek, 2002; Sastry, 2001). The population is updated for a number of iterations (generations), each consisting of four steps. First, promising solutions are selected from the current population using a GA selection method, such as tournament and truncation selection. Second, a Bayesian network that fits the population of promising solutions is constructed. Third, new candidate solutions are generated by sampling the built Bayesian network. Fourth, the new candidate solutions are incorporated into the original population, replacing some of the old ones or all of them.

A similar algorithm was independently developed by Etxeberria and Larrañaga (1999a), who called their algorithm the estimation of Bayesian networks algorithm (EBNA). The factorized distribution algorithm (FDA) (Mühlenbein, Mahnig, & Rodriguez, 1999), which uses a fixed distribution factorization specified in advance as the basis for constructing the probabilistic model was also later extended to allow for the learning of the factorization, yielding the learning FDA (LFDA) (Mühlenbein & Mahnig, 1999).

The remainder of this section describes basics of Bayesian networks, and discusses briefly the class of problems that BOA can solve in a robust and scalable manner.

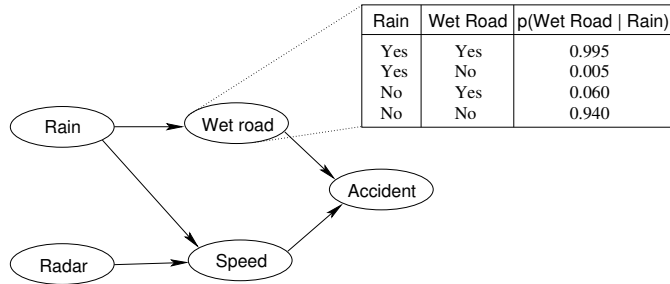


Figure 2: A Bayesian network example with a conditional probability table for one of the variables.

### 3.1 Bayesian network basics

Bayesian networks (Howard & Matheson, 1981; Pearl, 1988; Buntine, 1991) belong to the family of graphical models, which combine statistics, modularity, and graph theory in a practical tool for estimating probability distributions and inference. A Bayesian network is defined by two components:

- (1) **Structure.** 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.
- (2) **Parameters.** The parameters are represented by a set of conditional probability tables specifying a conditional probability for each variable given any instance of the variables that the variable depends on.

Mathematically, a Bayesian network encodes a joint probability distribution given by

$$p(X) = \prod_{i=1}^n p(X_i | \Pi_i), \quad (2)$$

where  $X = (X_0, \dots, X_{n-1})$  is a vector of all the variables in the problem;  $\Pi_i$  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  $p(X_i | \Pi_i)$  is the conditional probability of  $X_i$  given its parents  $\Pi_i$ .

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.

Figure 2 shows a simple example Bayesian network structure. The example network encodes a number of conditional dependencies. For instance, the speed of the car depends on whether it is raining and/or radar is enforced. The road is most likely wet if it is raining. Additionally, the network encodes a number of simple and conditional independence assumptions. For instance, the radar enforcement is independent of whether it is raining or not. A more complex conditional independence assumption is that the probability of an accident is independent of whether the radar is enforced, given a particular speed and

whether the road is wet. To fully specify the Bayesian network with the structure shown in the figure, it would be necessary to add a table of conditional probabilities for each variable. The figure shows a conditional table for the variable determining whether the road is wet.

It is important to understand the semantics of Bayesian networks in the framework of PMBGAs. Conditional *dependencies* will cause the involved variables to remain in the configurations seen in the selected population of promising solutions. On the other hand, conditional *independencies* lead to the mixing of bits and pieces of promising solutions in some contexts (the contexts are determined by the variables in the condition of the independency). The complexity of a proper model is directly related to a proper problem decomposition discussed in the last paragraph of the previous section. If the problem was linear, a good network would be the one with no edges, which is equivalent to the probability vector. More complex problems lead to more complex models, although many problems can be solved with quite simple networks despite the presence of nonlinear interactions of high order. In any case, Bayesian networks provide BOA with a general enough tool to express a wide variety of ways to decompose problems; coupling this expressive representation with methods for learning and sampling Bayesian networks will comprise a powerful tool for the exploration of new candidate solutions in genetic and evolutionary algorithms.

### 3.2 Learning and sampling Bayesian networks

The problem of learning Bayesian networks can be broken up into two subproblems: (1) learn the structure, and (2) learn the parameters for a specific structure.

It is common practice to define the learning of the structure as a black-box optimization problem, where a procedure (a scoring metric) is defined that computes a score for each candidate structure and an optimizer is executed to find the best structure with respect to the specific measure. The score assigned to a network can be computed in different ways, but in all cases it depends on both the structure of the network as well as the data that are to be modeled by the structure. The score can often also incorporate prior knowledge about the problem domain. Two classes of scoring metrics are common: Bayesian metrics (e.g., the Bayesian-Dirichlet metric with likelihood equivalence (BDe) (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994)), and minimum description length (MDL) metrics (e.g., the Bayesian information criterion (BIC) (Schwarz, 1978)).

A simple greedy search algorithm is usually used to construct the network that maximizes the scoring metric. The greedy algorithm starts with an empty network and in each iteration it applies a primitive graph operator to the current network that improves the network score the most compared to other applicable primitive operators. As primitive operators, edge additions, removals, and reversals are usually considered. Of course, the network must remain acyclic at all times, and the search is stopped whenever the current network cannot be improved anymore. In some cases, it is necessary to restrict the complexity of the network to contain dependencies of at most a specified order (Heckerman, Geiger, & Chickering, 1994; Pelikan, Goldberg, & Cantú-Paz, 1999). In BOA the network from the previous generation can be used as a starting point for building the model in each generation as suggested by Etxeberria and Larrañaga (1999b); this can significantly reduce the overall computational complexity of model building.

Once the structure has been constructed, one must compute the conditional probabilities for every variable given its parents. These can be computed from the selected

population of solutions in a straightforward manner.

The sampling of a Bayesian network can be done using probabilistic logic sampling (PLS) (Henrion, 1988). In PLS the variables are first ordered topologically so that every variable is preceded by its parents. For instance, in the example shown earlier in Figure 2, *Radar* would have to precede *Speed*, which would have to precede *Accident*. One of the complete topological orderings of the variables in the aforementioned example would be *Rain, Radar, Speed, Wet Road, Accident*. The values in each position of a new solution are then generated according to the topological ordering. As a result, once one is to generate the value of a variable, its parents would have been generated already, and the probability of a 0 or a 1 in this position can be directly extracted from the conditional probability table for the variable using the values of its parents.

### 3.3 BOA solves decomposable problems

Theoretical and empirical evidence exists that BOA can solve problems decomposable into subproblems of bounded order in quadratic or subquadratic number of evaluations (Pelikan, Goldberg, & Cantú-Paz, 2000a; Pelikan, Sastry, & Goldberg, 2002; Pelikan, 2002; Mühlenbein & Mahnig, 1999). That means that if there exist statistics of bounded order that lead toward the optimum, BOA should be able to find these statistics and use them to find the optimum quickly, accurately, and reliably.

How does BOA compare to conventional GAs and other optimization algorithms? Using only local operators, such as mutation, will lead to  $O(n^k \ln n)$  evaluations until convergence to the optimum (Mühlenbein, 1992), where  $k$  is the minimum order of subproblems in a proper problem decomposition. That means that the order of the number of evaluations required by local operators to solve decomposable problems grows with the order of subproblems, yielding a very inefficient search already for subproblems of order as low as  $k = 4$  or  $k = 5$ . This complexity increase will not be eliminated by other common extensions of local search, such as simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983) and tabu search (Glover, 1986).

The situation gets even worse for selectorecombinative GAs with commonly used crossover operators. It can be easily shown that with one-point or uniform crossover, there exist many decomposable problems that will require exponential population sizes with respect to the size of the problem (Thierens & Goldberg, 1993; Thierens, 1995). That makes even moderately sized problems intractable.

To get a better idea about the differences between BOA and other methods, Figure 4 compares the performance of BOA on trap functions with that of a mutation-based hill climber (Figure 4a) and the GA with uniform crossover (Figure 4b). Trap functions tested here are obtained by first partitioning string positions into independent groups of 5 bits each, and applying a 5-bit trap function to each of these groups of bits. The partitioning is fixed during the whole optimization run, but it is initialized at random to make it impossible to exploit any fixed decomposition. The fitness is equal to the sum of the contributions of all the groups. Each 5-bit trap is defined as shown in Figure 3 (Ackley, 1987; Deb & Goldberg, 1994). Trap functions tested here can thus be decomposed into independent subproblems of 5 bits each. For each problem size, 30 independent runs are performed and the population sizes in the GAs and BOA are set to a minimum population size to achieve convergence in all the 30 runs. For more details on the experimental setup, please see Pelikan (2002). The figure confirms that BOA is capable of solving 5-bit traps in a subquadratic number of evaluations, whereas the hill climber needs  $O(n^5 \log n)$

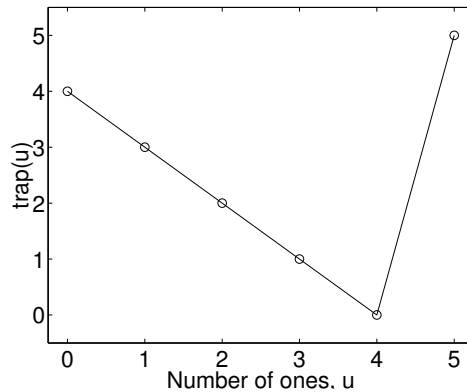


Figure 3: A 5-bit trap.

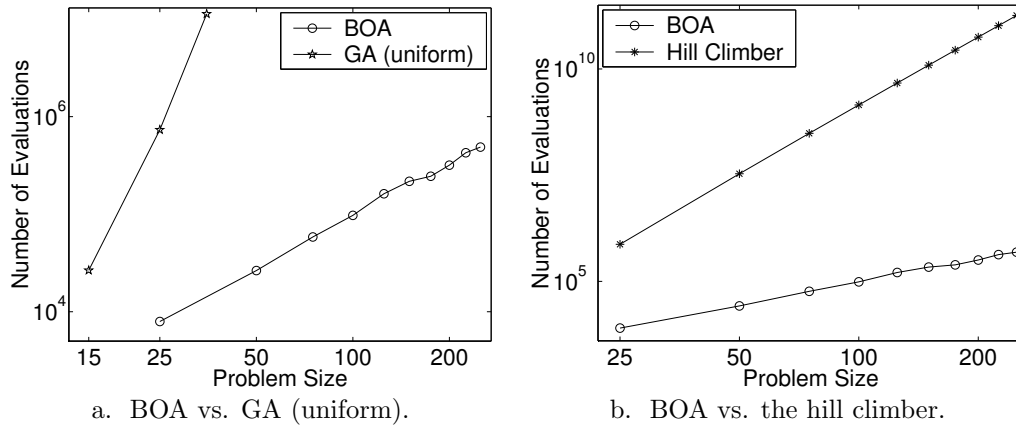


Figure 4: BOA vs. the GA with uniform crossover and the mutation-based hill climber on 5-bit composed traps. The graphs were obtained from Pelikan (2002).

evaluations and the GA with uniform crossover needs exponentially many evaluations.

#### 4 Hierarchical BOA: From single level to hierarchy

That BOA provides robust and scalable solution to boundedly difficult decomposable problems is an important result on its own right because many real-world problems are nearly decomposable and BOA qualitatively outperforms most other conventional black-box optimizers at solving this important class of problems (Pelikan, Sastry, & Goldberg, 2002; Pelikan, 2002). But can BOA be extended beyond problems of bounded difficulty to solve other important classes of problems? What other classes of problems should be considered?

This section attempts to answer these question by deriving inspiration from the way the best problem solvers—humans—approach their problems. The section first argues that hierarchical decomposition is an important and powerful tool for solving challenging problems and developing tractable models of complex systems, and presents a motivating example of using hierarchical decomposition in automotive design. The section then presents the three keys to hierarchy success that must be incorporated into a selectorecom-

binative black-box optimizer to enable the use of hierarchical decomposition as opposed to decomposition on only a single level. The section then presents the hierarchical BOA (hBOA), which incorporates the keys to hierarchy success into BOA. The section closes with a brief discussion of the class of problems that can be solved by hBOA in a robust and scalable manner.

#### 4.1 Hierarchical decomposition

A core component of engineering design and almost any other area of human endeavor that deals with complex systems is *hierarchical decomposition* (Simon, 1968; Pattee, 1973). By hierarchical decomposition we mean decomposition that goes down a number of levels of difficulty, where subproblems on each level can be decomposed into simpler subproblems that form the next level and interactions between the subproblems on each level are of much lower magnitude than the interactions within these subproblems (Simon, 1968).

Let us start with an example. Figure 5 shows a part of hierarchical decomposition that can be used for automotive design. A car can be decomposed into an engine system, a braking system, an electrical system, and so forth. Each of these systems can be further simplified via decomposition on yet another level of hierarchy. For example, the engine system can be decomposed into the fuel system, valves, ignition system, and so forth. Decomposition can go down a number of levels until the subproblems on the bottom level are easy enough (for example, they are all of bounded difficulty). The target system can then be designed in a bottom-up fashion by combining simple systems from lower levels to form systems on higher levels. For instance, an engine can be constructed by combining different types of fuel systems, valves, ignition systems, and so forth. The resulting engine or engines can be used together with braking systems and other components to form the entire car.

Hierarchical decomposition clearly simplifies the problem of designing a car—instead of looking at a car as a complex machine with a complex functional specification, one can separately focus on the design of the engine system, braking system, and so forth. Each of these systems is significantly simpler than the car itself, and its design can be approached by designing even simpler subsystems on lower levels in a recursive fashion. These can be then combined to form systems on higher levels until the entire car is constructed.

The use of hierarchy is not restricted to engineering design; examples of using hierarchy can be found all around us. Just as engineers use hierarchy to design better cars, organizations use hierarchy to maximize their efficiency, software engineers use hierarchy to simplify the design of complex software systems, biologists use hierarchy to better understand complex biological systems, and mathematicians use hierarchy to enable the design of complex mathematical theories. Other examples can be found easily; see for example Simon (1968), Pattee (1973), Sacerdoti (1975), Stefik (1981a), Stefik (1981b), Allen and Starr (1982), Kulish (2002), and Faihe (1999).

In black-box optimization, the hierarchical search for the optimum should start from the bottom level by combining alternative partial solutions to the subproblems on that level. Best partial solutions on the bottom level should be used as building blocks for constructing solutions on the next level. The search should proceed in a bottom-up fashion by combining solutions on each level to construct solutions on the next higher level. Once the search reaches the top level, the final solution can be constructed.

The hierarchical construction of the optimal solution resembles Holland’s building block hypothesis (Holland, 1975), which—loosely said—states that the search in GAs

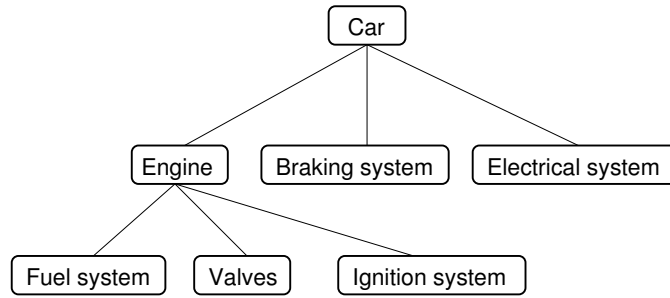


Figure 5: Hierarchical decomposition of automotive design.

should proceed by combining short-order partial solutions to form solutions of higher and higher order until the final solution is constructed. We have seen, however, that conventional GAs cannot do this even on a single level. This paper presents an algorithm—the hierarchical BOA—that puts the hierarchical search from Holland’s building-block hypothesis in practice. The important keys to solving difficult hierarchical problems are first identified; these are then incorporated into BOA to form its hierarchical extension, the hierarchical BOA (hBOA).

#### 4.2 Three keys to hierarchy success

Building on top of IlliGAL GA design decomposition (Goldberg, Deb, & Clark, 1992; Goldberg, 1994; Goldberg, 2002), this section identifies three important facets of black-box optimization via hierarchical decomposition—the three keys to hierarchy success—that are not included in conventional GAs. The three keys to hierarchy success (Pelikan & Goldberg, 2000b; Pelikan & Goldberg, 2001; Pelikan, 2002) are (1) proper decomposition, (2) chunking, and (3) preservation of alternative candidate solutions.

1. **Proper decomposition.** On each level of problem solving, the hierarchical black-box optimizer must be capable of decomposing the problem properly. The decomposition allows the solver to focus on the different subtasks in separation and thus reduce complexity. The decomposition can be as simple as the partitioning of problem variables into several disjoint subsets (for instance, on concatenated 5-bit traps) but it can be as complex as the graphs with a bounded number of neighbors of each node (for instance, on spin glasses discussed in Section 5).
2. **Chunking.** The solutions to the subproblems from the lower level can be seen as *chunks* of the solutions that are used as basic building blocks for constructing the solutions on the current level. In other words, the partial solutions to each subproblem can be treated as the instances of a single variable that is used to construct the solutions on the next level. The hierarchical problem solver must be capable of representing these chunks of solutions from lower levels in a compact way so that only relevant features are considered.
3. **Preservation of alternative candidate solutions.** The hierarchical problem solver must be capable of preserving multiple alternative solutions to each subproblem. There are two reasons for doing this:
  - (i) On the current level there may not be a sufficient feedback to discriminate among a few best alternative solutions to the considered subproblem.

```

Hierarchical BOA (hBOA)
t := 0;
generate initial population P(0);
while (not done) {
    select population of promising solutions S(t);
    build Bayesian network B(t) with local struct. for S(t);
    sample B(t) to generate offspring O(t);
    incorporate O(t) into P(t) using RTR yielding P(t+1);
    t := t+1;
};

```

Figure 6: The pseudocode of the hierarchical BOA (hBOA).

- (ii) Although the subproblems on the current level are considered independent, interactions on some higher level or levels may lead to new information that favors some of the alternatives over others.

The hierarchical BOA (hBOA) extends BOA to ensure the three keys to hierarchy success. For proper decomposition, hBOA uses Bayesian networks to model promising solutions and sample the new ones. For chunking, hBOA introduces local structures into Bayesian networks for efficient representation of dependencies of high order. Finally, hBOA uses the restricted tournament replacement (RTR) to ensure diversity preservation. hBOA thus differs from BOA in two important ways: (1) hBOA uses local structures for more compact representation of models and (2) hBOA introduces a niching technique for effective diversity maintenance. The pseudocode of hBOA is shown in Figure 6. The remainder of this section discusses the two ways in which hBOA differs from BOA in somewhat more detail.

### 4.3 Hierarchical BOA: Chunking

In Bayesian networks, local probability distributions are usually represented by conditional probability tables, the size of which grows at least exponentially with the order of interactions (or the number of parents). That means that if we wanted to represent interactions of order proportional to the size of the problem—which might be the case on top levels of the hierarchical search—the population size would have to grow exponentially with the problem size. This would lead to exponential memory and time complexity of the algorithm, which is clearly intractable for even small problems. Of course, appropriate metrics for scoring Bayesian network structures will never allow such complex structures to appear because the pressure toward simpler models will overshadow the increase in the likelihood of the data when using such complex models.

There is no way to eliminate exponential complexity of complete local conditional probability distributions or provide enough statistical support for dependencies spanning over the entire string without getting trapped in exponential complexity. However, note that despite that the order of dependencies on top levels might grow linearly with the problem size, the number of high-quality instances that the dependent variables take can be assumed to be constant or grow slowly (at most polynomially) with the problem size. The reason for this restriction is the same as that for requiring the order of decomposition to grow at most logarithmically with the problem size for boundedly difficult decomposable problems—if the number of instances would grow faster than polynomially, so would the

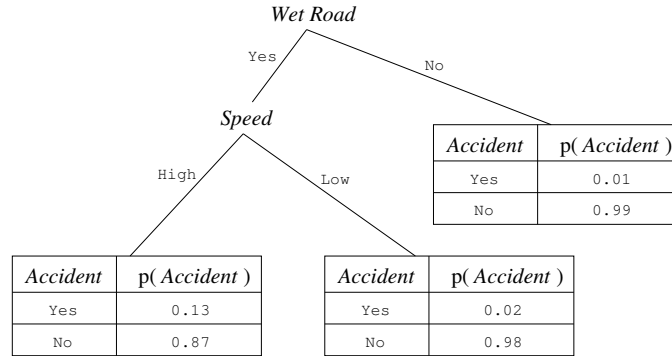


Figure 7: An example decision tree for representing the conditional probability table for *Accident* given *Wet Road* and *Speed*.

computational complexity.

Since the number of instances to cover in the conditional probability tables and scoring metrics is bounded by a polynomial, using exponentially sized complete probability tables for local distributions seems to be inefficient. Instead, more compact structures, such as decision trees, can be used to exploit the small number of instances corresponding to each dependency and these compact structures can then be reflected in the scoring metrics. Bayesian networks with compact representation of local probability distributions are often referred to as Bayesian networks with local structures (Friedman & Goldszmidt, 1999; Chickering, Heckerman, & Meek, 1997).

Decision trees are among the most popular local structures for representing local probability distributions in Bayesian networks (Friedman & Goldszmidt, 1999; Chickering, Heckerman, & Meek, 1997) but also for many other machine learning tasks. A decision tree is a directed acyclic graph where each node except for the root has exactly one parent; the root has no parents. The internal nodes (all nodes except for the leaves) of the tree are labeled by a variable (feature). When a node is labeled by a variable  $v$ , we say that the node is a split on  $v$ . Edges from a split on  $v$  are labeled by non-empty distinct exhaustive subsets of possible values of  $v$ . Given an assignment of all the variables, a traversal of the decision tree starts in the root. On each split on  $v$ , the traversal continues to the child along the edge containing the current value of  $v$ . For each assignment of the involved variables, there is only one possible way of traversing the tree to a leaf, because the edges coming from each split must be labeled by distinct subsets of values.

Each leaf of a decision tree contains a quantity or information of interest associated with all the instances that end up the traversal of the tree in that leaf. To use decision trees for representing the conditional probabilities of a particular variable, each leaf stores the conditional probabilities of the variable given that the variables contained in the path from the root to the leaf are fixed according to the path. The splits in the decision tree for a variable thus correspond to the parents of the variable in conventional Bayesian networks. See Figure 7 for a decision tree that could represent the conditional probabilities the probability of *Accident* in the Bayesian network shown earlier in Figure 2 on page 7.

Learning Bayesian networks with decision trees is similar to learning traditional Bayesian networks. The metrics must be modified slightly to incorporate the different representation, but their general form remains similar (Chickering, Heckerman, & Meek, 1997; Friedman & Goldszmidt, 1999). The greedy search should be modified to replace

operators over edges with operators over decision trees. Decision-tree operators usually introduce a new split. One can also introduce an operator that eliminates an existing split but, according to our experience, eliminating splits is not beneficial in practice. Learning the network can start with an empty decision tree for every variable, which would be expanded as needed based on the value of the scoring metric until no more improvement is possible. For more details on the construction and evaluation of Bayesian networks with local structures, see Chickering, Heckerman, and Meek (1997), Friedman and Goldszmidt (1999), Pelikan, Goldberg, and Sastry (2000), and Pelikan (2002).

#### 4.4 Hierarchical BOA: Preservation of alternative candidate solutions

One of the advantages of using population-based search is that populations can store a diverse sample of candidate solutions spanning across the entire search space. Diversity preservation can be achieved by introducing a number of artificial niches into the population and encouraging competition among solutions within each niche. In combination with Bayesian networks, this can enable BOA and hBOA to maintain a sample corresponding to a probability distribution over the entire search space without having to focus on a specific region or regions as is common in other optimization methods such as simulated annealing or gradient search. However, as noted above, for a successful hierarchical optimizer it is necessary to add an additional mechanism to ensure that useful diversity is maintained in a robust manner.

There are three general approaches to niching: (1) fitness-sharing, (2) selection-based, and (3) island models. The following paragraphs briefly discuss each approach. It is beyond the scope of this paper to give a complete overview, and we refer the reader to Pelikan and Goldberg (2000c).

The first approach modifies the fitness landscape before the selection is performed. Fitness sharing (Goldberg & Richardson, 1987) is based on this idea. In fitness sharing, the location of each candidate solution is set to the solution itself (e.g., a particular binary string). The neighborhood of each candidate solution is defined by the *sharing function* using a distance metric defined over the domain of all candidate solutions. A candidate solution shares a niche with any individual that is within a certain range from its location. The effect may decrease with the distance and completely vanishes for distances greater than a certain threshold.

The second approach modifies the selection operator itself to take into account both the fitness as well as the solution instead of using the fitness as the only criterion. In *preselection* of Cavicchio (1970) the offspring after applying two-parent variation operators replaced the inferior parent. This scheme was later generalized by De Jong (1975) who proposed *crowding*. In crowding, for each new candidate solution a subset of the original population before selection is first selected. The new candidate then replaces the most similar candidate solution in this subset. Harik (1994) proposed the restricted tournament selection as an extension of De Jong's crowding. The difference between RTS and crowding is that in RTS the new candidate solution replaces the closest candidate solution from the selected subset only if it is better in terms of fitness. Therefore, RTS introduces selection pressure and can replace the selection operator.

The third approach is to isolate several groups of candidate solutions rather than to keep the entire population in one location. The location of each solution does not depend on its genotype or phenotype. Candidate solutions can migrate between different locations (islands or demes) at certain intervals and allow the population at each location to develop

in isolation. 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.

BOA uses the set of selected solutions to learn a probabilistic model of promising solutions. Fitness sharing would affect the fitness and, consequently, it may also affect the model construction. That is why it is desirable that we use a different niching method in the BOA. Spatial separation can be directly encoded in the probabilistic model by using mixture distributions or models with hidden variables. A simple method based on mixture models to reduce negative effects of symmetry in the problem on BOA was proposed in Pelikan and Goldberg (2000a). However, using such mixture models requires the population to enlarge by a factor equal to the number of niches, which can lead to an increased computational complexity. That is why we used the restricted tournament selection to incorporate niching into the hierarchical BOA. Since the technique is used as a replacement technique and not as a primary source of selection pressure, we called the method *restricted tournament replacement* (RTR).

#### 4.5 hBOA solves hierarchically decomposable problems

hBOA can solve problems that can be decomposed on a single or multiple levels of difficulty. That means that not only can hBOA solve those problems that can be efficiently solved by BOA, but it can solve a larger class of problems including problems that cannot be solved by BOA. An example hierarchical problem that cannot be efficiently solved using single-level decomposition are hierarchical traps (Pelikan, 2002), created by combining trap functions of order 3 over multiple levels of difficulty. On the lowest level, groups of 3 bits contribute to the overall fitness using 3-bit traps. Each group of 3 bits corresponding to one of the traps is then mapped to a single bit on the next level; a 000 is mapped to a 0, a 111 is mapped to a 1, and everything else is mapped to the null symbol '-'. The bits on the next level again contribute to the overall fitness using 3-bit traps, and the groups are mapped to an even higher level. This continues until the top level is evaluated that contains 3 bits total. That is why the string length should be an integer power of 3. Any group of bits containing the null symbol does not contribute to the overall fitness. For more details on hierarchical traps, please see Pelikan (2002).

The number of evaluations until convergence to the optimum of hierarchical traps is shown in Figure 8. The results indicate that hierarchical traps can be solved in a sub-quadratic number of evaluations. How would other algorithms perform on this class of problems? All algorithms that have failed on single-level traps because of exponential complexity, will fail on hierarchical traps as well. That is why conventional GAs with one-point or uniform crossover are not candidates for solving this class of problems at all. Recall that boundedly difficult single-level decomposable problems can be solved by local search in approximately  $O(n^k \log n)$  evaluations, where  $k$  is the order of appropriate problem decomposition. For hierarchical traps, local search will also become exponentially expensive, because hierarchical traps cannot be decomposed into boundedly difficult sub-problems on a single level. That is why local search is no longer able to retain polynomial convergence. We performed a number of experiments on hierarchical traps using different optimization techniques—including a (1+1)-ES, simulated annealing, and hill climbing—and in all cases the algorithms were not able to solve any but smallest problems of  $n = 9$  or

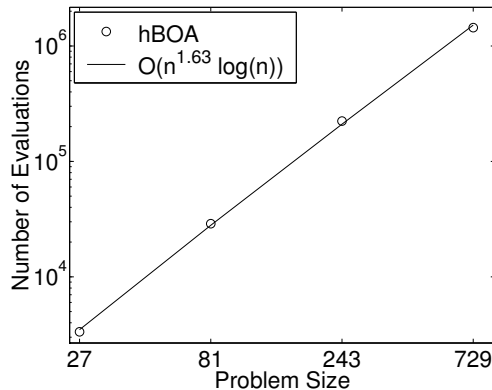


Figure 8: The performance of hBOA on hierarchical traps. The graph was obtained from Pelikan (2002).

$n = 27$  bits even after days of computation, providing additional support for their highly inefficient and unscalable performance. For more details, see Pelikan (2002).

## 5 Experiments

We have just presented an advanced black-box optimizer hBOA and argued that hierarchical decomposition will allow hBOA to solve many challenging real-world problems in a robust and scalable manner. This section applies hBOA to two- and three-dimensional Ising spin glass systems with periodic boundary conditions to confirm that hierarchical decomposition is a useful concept in solving real-world problems. The section shows how easy it is to apply hBOA to combinatorial problems and achieve competitive or better performance than problem-specific approaches without the need for much problem-specific knowledge in advance. Ising spin glasses represent an important class of problems also because it can be shown that within the PMBGA framework, for an exact decomposition of the distribution of high-quality solutions (even in the 2D case) it is necessary to consider statistics of order proportional to  $\sqrt{n}$  (Mühlenbein, Mahnig, & Rodriguez, 1999), which leads to exponential complexity in both computational time and memory; promising results presented here thus show that approximate decomposition may be sufficient in practice.

### 5.1 Problem definition

The task of finding ground states of spin-glass systems is a well known problem of statistical physics. In context of GAs, spin glasses are usually studied because of their interesting properties, such as symmetry and a large number of plateaus (Pelikan, Goldberg, & Cantú-Paz, 2000b; Pelikan & Mühlenbein, 1999; Naudts & Naudts, 1998; Van Hoyweghen, 2001; Mühlenbein, Mahnig, & Rodriguez, 1999).

The physical state of a spin-glass system is defined by (1) a set of spins  $(\sigma_0, \sigma_1, \dots, \sigma_{n-1})$ , where each spin  $\sigma_i$  can obtain a value from  $\{+1, -1\}$ , and (2) a set of coupling constants  $J_{ij}$  relating pairs of spins  $\sigma_i$  and  $\sigma_j$ . A Hamiltonian specifies the energy of the system as  $H(\sigma) = -\sum_{i,j=0}^{n-1} \sigma_i J_{ij} \sigma_j$ . The task is to find a state of spins called the *ground state* for given coupling constants  $J_{ij}$  that *minimizes* the energy of the system. There are at least two ground states of each spin-glass system (the energy of a system

does not change if one inverts all the spins). In practice the number of ground states is usually much greater, and it often grows exponentially with the problem size (the number of spins).

The problem of finding *any ground state* of a spin glass is equivalent to a well known combinatorial problem called *minimum-weight cut* (MIN-CUT). Since MIN-CUT is NP-complete (Monien & Sudborough, 1988), the task of finding a ground state of an unconstrained spin-glass system is NP-complete—that means that there exists no algorithm for solving general Ising spin glasses in worst-case polynomial time and it is believed that it is impossible to do this.

Here we consider a special case, where the spins are arranged on a two- or three-dimensional grid and each spin interacts with only its nearest neighbors in the grid (4 neighbors in 2D, 6 neighbors in 3D). Periodic boundary conditions are used to approximate the behavior of a large-scale system. Therefore, spins are arranged on a 2D or 3D toroid. Additionally, here we consider Ising spin glass systems, where coupling constants are constrained to contain only two values,  $J_{ij} \in \{+1, -1\}$ . The complexity of finding the ground state for Ising spin glasses grows with the dimensionality of the problem. The 1D case is trivial and can be easily solved by starting in an arbitrary spin, and continuing spin by spin along the chain of coupled spins in one direction, always setting the spin value to minimize the energy of the system with respect to the coupling with the spin that was assigned last. In the worst case, one constraint will remain unsatisfied (one coupling will have positive contribution to the energy Hamiltonian). That is why we do not study 1D spin glasses in this paper. In the 2D case, several algorithms exist that can solve the restricted class of spin glasses in polynomial time (Kardar & Saul, 1994; De Simone, Diehl, Jünger, & Reinelt, 1996; Galluccio & Loeb, 1999a; Galluccio & Loeb, 1999b). We will compare the best known algorithms to hBOA later in this section. However, none of these methods is applicable in the 3D case.

## 5.2 Experimental methodology

In hBOA each state of the system is represented by an  $n$ -bit binary string, where  $n$  is the total number of spins. Each bit in a solution string determines the state of the corresponding spin: 0 denotes the state  $-1$ , 1 denotes the state  $+1$ . To estimate the scalability of hBOA on 2D Ising spin-glass systems, we tested hBOA on random instances for problems ranging from  $n = 6 \times 6 = 36$  to  $n = 20 \times 20 = 400$  spins, 1000 random instances for each problem size. To ensure that a correct ground state was found for each system, we verified the results using the Spin Glass Ground State Server provided by the group of Prof. Michael Jünger.<sup>1</sup>

For each problem instance, 10 independent runs are performed and hBOA is required to find the optimum in all the 10 runs. The performance of hBOA is measured by the average number of evaluations until the optimum is found. The population size for each problem instance is determined empirically as the minimal population size for the algorithm to find the optimum in all the runs. A parameter-less population sizing scheme (Harik & Lobo, 1999) could be used to eliminate the need for specifying the population size in advance, which could increase the total number of evaluations by at most a logarithmic factor (Pelikan & Lobo, 1999). Binary tournament selection with replacement is used in all experiments and the window size for RTR is set to the number of bits (spins) in a problem, but it is bounded to be equal to at most 5% of the overall population size. Limiting the size of RTR significantly improved the performance of hBOA compared to

<sup>1</sup><http://www.informatik.uni-koeln.de/lj.juenger/projects/sgs.html>.

the previously published results (Pelikan, 2002). Bayesian networks with decision graphs are used and K2 metric with the term penalizing complex models is used to measure the quality of each candidate model as described in (Pelikan, Goldberg, & Sastry, 2001).

The performance of hBOA is improved by enhancing the initial population of candidate solutions using a simple heuristic based on a random walk through the grid. In each solution of the initial population, a random spin is first selected as a starting point for the random walk. Then, a random unvisited neighbor of the current spin is visited, setting its value to increase the fitness of the solution the most. The walk continues until all spins have been visited. If all neighbors of the current spin have already been visited, the walk continues in a randomly selected unvisited spin.

The performance of hBOA is also improved by combining hBOA with a local searcher referred to as the *discrete hill climber* (DHC). DHC is applied prior to the evaluation of each solution by flipping a bit that improves the solution the most; this is repeated until no more improvement is possible. For most constraint satisfaction problems including Ising spin glasses, DHC increases the computational cost of each evaluation by at most  $n \log n$  time steps; in practice, the increase in computational complexity is still significantly lower because only a few bits are flipped on average. Here, for instance, DHC does not increase the asymptotic computational complexity at all, while it decreases the required number of evaluations approximately tenfold.

Using local search often improves the performance of selectorecombinative search, because the search can focus on local optima, which reveal more information about the problem than randomly generated solutions do. Furthermore, selectorecombinative search can focus its exploration on basins of attraction (peaks around each local optimum) as opposed to individual solutions. On the other hand, in some cases local search may cause premature convergence; nonetheless, we believe that this is rarely going to be the case with advanced algorithms such as BOA and hBOA.

Although the added heuristics improve the efficiency of hBOA, the asymptotic complexity of hBOA does not change much with the addition of the heuristics (Pelikan, 2002; Pelikan & Goldberg, 2003). Since the conclusions made from the experimental results are based on the asymptotic complexity, they are not a consequence of the additional heuristics.

### 5.3 Results

Figure 9a shows the number of evaluations for hBOA with DHC on 2D spin glasses from  $n = 6 \times 6$  to  $n = 20 \times 20$  and Figure 9b shows the number of flips per evaluation for the same set of instances. The number of evaluations appears to grow polynomially as  $O(n^{1.51})$ , whereas the number of flips per evaluation appears to grow as  $O(n^{0.54})$ . The performance of hBOA is thus even slightly better than predicted by theory (Pelikan & Goldberg, 2001; Pelikan, 2002; Pelikan, Sastry, & Goldberg, 2002), which estimates the number of evaluations for difficult hierarchical problems as  $O(n^{1.55} \log n)$ . Moreover, the average number of flips per evaluation indicates that the asymptotic complexity of the evaluation will remain approximately linear with the problem size.

### 5.4 Comparison with other black-box optimizers

Figure 10 compares hBOA's performance with that of simple GAs with one-point and uniform crossover (Figure 10a), and the univariate marginal distribution algorithm (UMDA) (Figure 10b), which uses a probability vector to model promising solutions. The experi-

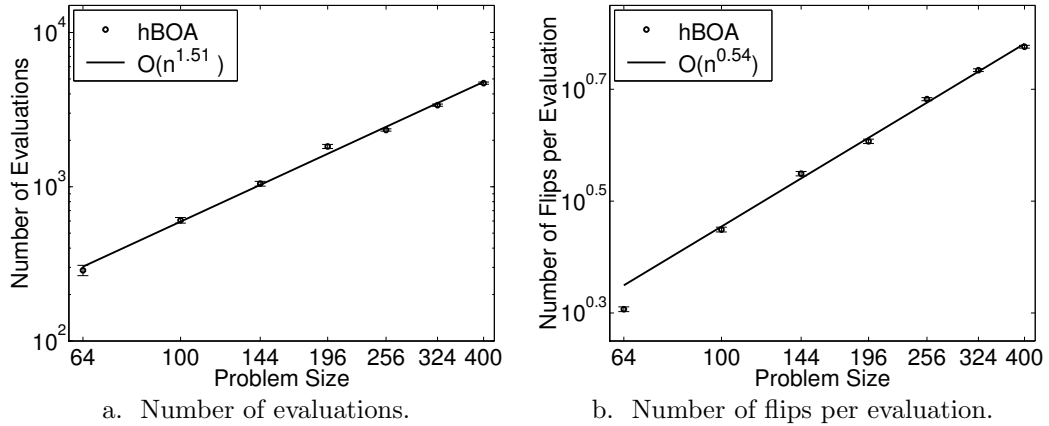


Figure 9: The number of evaluations and DHC flips per evaluation for hBOA on 2D Ising spin glasses (1000 random instances for each problem size).

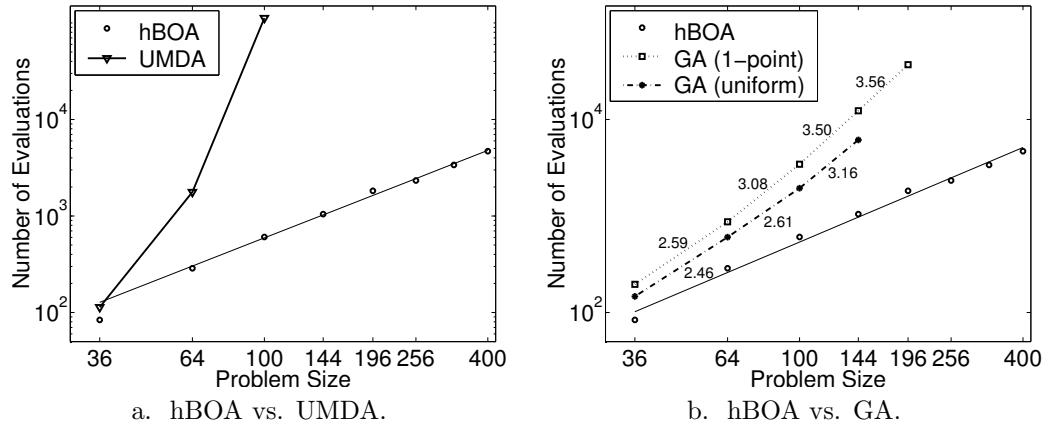


Figure 10: hBOA vs. UMDA and simple GAs on 2D Ising spin glasses (1000 random instances for each problem size). For both GA variants, slopes are displayed adjacent to the corresponding line segments to highlight exponential complexity.

mental methodology was identical to that for hBOA including DHC as a local searcher and a random walk for initializing the population. The figure shows that the number of evaluations required by GAs with uniform crossover and UMDA grows exponentially with the problem size. The number of evaluations for GAs with one-point crossover also starts to grow exponentially fast; however, the exponent is slowly decreasing and it can be expected that the number of evaluations can be bounded by a polynomial of order at least 3.56. A reason for this is that in these experiments, the grid is coded with binary strings so that one-point crossover does not break important interactions often. Such an encoding is the best encoding one could expect for one-point crossover; any perturbation of this encoding would result in the behavior similar to that of the GA with uniform crossover as is true for other problems. But no matter whether the performance in this ideal case is polynomial or not, the performance in all cases seems to be either exponential or qualitatively worse than that of hBOA.

We have also tried a (1+1)-ES with bit-flip mutation, which was not able to solve

any but smallest problems of  $n = 6 \times 6 = 36$  and  $n = 8 \times 8$  bits. Already for problems of size  $n = 10 \times 10 = 100$  bits, (1+1)-ES was not able to find the optimum in most cases even after hours of computation, which suggests exponential performance as well. Exponential performance was also the case for DHC with random restarts if trapped in a local optimum. Similar performance can be expected from other methods based on local search, which suffer from the large number of local optima in the problem and require exponential time to find a ground state.

## 5.5 Comparison with problem-specific methods

To compare the computational complexity of hBOA with problem-specific methods for solving spin glasses, let us first compute the overall computational complexity of hBOA with DHC on the tested spin glass instances. The number of evaluations appears to grow as  $O(n^{1.51})$ , where approximately  $O(n^{0.91})$  comes from the population size, and  $O(n^{0.6})$  comes from the number of generations. Every evaluation can be done in  $O(n)$  steps (including DHC, which in this case takes approximately  $O(\sqrt{n} \log n)$  steps per evaluation). The overall time spent in evaluation is thus bounded by  $O(n^{2.51})$ . The complexity of model building in each generation can be estimated as  $O(kn^2N)$ , where  $N$  is the population size and  $k$  is the depth of decision trees. Assuming that the depth of decision trees grows at most logarithmically with the size of the problem, we get  $O(n^{3.51} \log n)$  steps for model building overall. Replacement using RTR can also be bounded by  $O(n^{3.51})$  steps, whereas selection can be done in only  $O(n^{2.51})$  steps total. The primary source of computational complexity is thus the model building and RTR, which can be bounded by  $O(n^{3.51} \log n)$  steps overall. This result is better than the results reported with hBOA in Pelikan (2002); this qualitative improvement was due to the limited size of RTR window size, which is now limited to be at most 5% of the problem size.

There are several problem-specific algorithms that attempt to solve the above special case of 2D spin glasses (e.g., Kardar and Saul (1994), De Simone, Diehl, Jünger, and Reinelt (1996), Galluccio and Loeb1 (1999a), Galluccio and Loeb1 (1999b)). Most recently, Galluccio and Loeb1 (Galluccio & Loeb1, 1999a; Galluccio & Loeb1, 1999b) proposed an algorithm for solving spin glasses in  $O(n^{3.5})$  for all graphs with bounded genus (two-dimensional toroids are a special case of graphs with bounded genus). So, the overall time complexity of the best currently known algorithm for the considered class of spin glasses is  $O(n^{3.5})$ . However, it is important to note that the method of Galluccio and Loeb1 can find the distribution of states over the entire energy spectrum, while hBOA focuses on only finding one or several configurations with minimum energy. That is why the method of Galluccio and Loeb1 solves in fact a more difficult problem.

The above results indicate that hBOA performs similarly as the best problem-specific approach; the time complexity of hBOA can be estimated as  $O(n^{3.51} \log n)$ , whereas the time complexity of the method of Galluccio and Loeb1 is  $O(n^{3.5})$ . Despite that hBOA does not use any problem-specific knowledge except for the evaluation of suggested states of the system and the method of Galluccio and Loeb1 fully relies on the knowledge of the problem structure and its properties, hBOA is capable of providing almost the same asymptotic complexity. But hBOA is not competitive only with respect to the asymptotic complexity; comparing the running times will reveal that hBOA is capable of outperforming the method of Galluccio and Loeb1 even with respect to the overall running time (Pelikan, 2002), so there seem to be no large constants hidden in  $O(\cdot)$  notation for the computational complexity of hBOA. As previous results suggest, the asymptotic complexity can be expected to be retained even without the use of the random-walk heuristic used for popu-

lation initialization and local search, but the factor by which the overall time complexity increases does not allow for such a detailed complexity analysis.

## 5.6 From 2D to 3D

Despite that competent methods exist for solving 2D spin glasses, none of these methods is directly applicable in the 3D case. In fact, finding a ground state of 3D spin glasses is NP-complete even for coupling constants restricted to  $\{-1, 0, +1\}$  (Barahona, 1982). Despite that, some positive results in solving 3D spin glasses are reported for instance in Hartmann (2001). Here we focus on a special case where the coupling constants are restricted to be  $\{-1, +1\}$ . Since hBOA does not explicitly use the dimensionality of the underlying spin-glass problem, it is straightforward to apply hBOA+DHC to 3D spin glasses. In this section we present preliminary results on several instances of 3D Ising spin glasses.

To test the scalability of hBOA with DHC, eight random spin-glass systems on a 3D cube with periodic boundary conditions were generated for systems of size from  $n = 4 \times 4 \times 4 = 64$  to  $n = 7 \times 7 \times 7 = 343$  spins. To verify whether the found state actually represents the ground state, hBOA with DHC was first run on each instance with an extremely large population of orders of magnitude larger than the expected one. After a number of generations, the best solution found was assumed to represent the ground state.

Figure 11 shows the number of evaluations (Figure 11a) and the number of flips per evaluation (Figure 11b) until hBOA with DHC found the ground state of the tested 3D Ising spin-glass instances. The overall number of evaluations appears to grow polynomially as  $O(n^{2.91})$ . That means that increasing the dimensionality of spin-glass systems increases the complexity of solving these systems; however, efficient performance appears to be retained even in three dimensions. Nonetheless, it is hard to predict what will happen for larger problems, and an intensive analysis of experimental results remains an important topic for future work. The number of flips per evaluation seems to also grow approximately with a square root of the problem size; however, the polynomial fit is much worse than in the 2D case, because the number of tested spin-glass instances for each problem size is significantly lower. A thorough testing of hBOA on 3D Ising spin glasses remains an important topic for future work, but the results presented here indicate that this direction of research holds big promise.

## 6 Summary and Conclusions

This paper discussed two advanced black-box optimizers, the Bayesian optimization algorithm (BOA) and the hierarchical Bayesian optimization algorithm (hBOA). Both algorithms are based on population-based genetic and evolutionary algorithms, but they replace traditional variation operators inspired by genetics by techniques for learning and sampling Bayesian networks. Additionally, hBOA incorporates local structures in Bayesian networks and niching to enable to use of hierarchical decomposition as opposed to single-level decomposition. The paper also presented a thorough empirical analysis of hBOA on the class of random 2D and 3D Ising spin glasses.

The empirical analysis together with the results presented in previous work (Pelikan, 2002; Pelikan & Goldberg, 2003; Ocenasek, 2002; Mühlenbein, Mahnig, & Rodriguez, 1999; Larrañaga & Lozano, 2002) confirm the assumption that decomposition and hierarchical decomposition is an inherent part of many real-world problems, and that effective discovery and exploitation of single-level and hierarchical decomposition enable robust and scalable

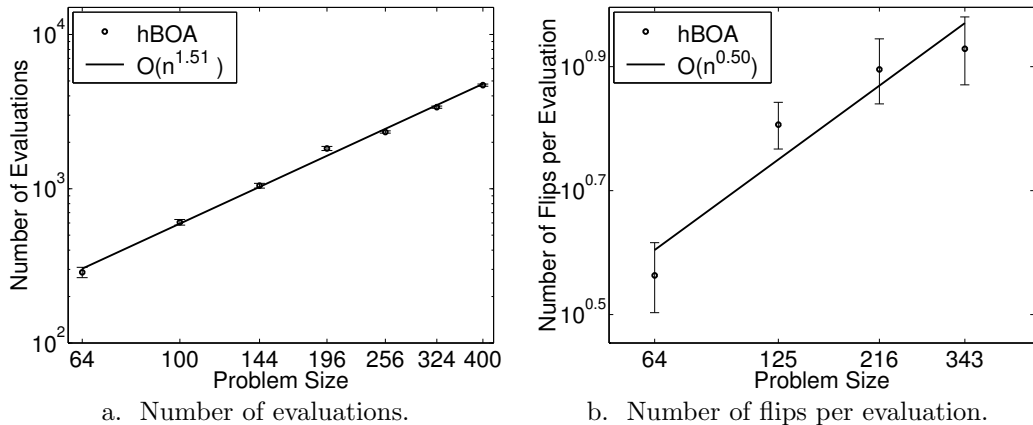


Figure 11: The number of evaluations and DHC flips per evaluation for hBOA on 3D Ising spin glasses (8 random instances for each problem size).

solution to a broad range of optimization problems. hBOA was capable of solving randomly generated Ising spin glass problem instances with competitive or better performance than problem specific approaches.

hBOA was told nothing about the semantics of the problem; initially it did not know whether it was trying to solve a spin glass, MAXSAT, or any other problem. All problem-specific knowledge was acquired automatically without any interaction with the user. Recently, hBOA was successfully applied to other classes of problems also without any knowledge of the semantics of the problem; these problems included onemax, composed traps, exponentially scaled deceptive problems, hierarchical traps, MAXSAT, graph coloring, and graph partitioning. Despite the lack of prior problem-specific knowledge, hBOA was capable of automatic discovery and exploitation of problem regularities that was effective enough to solve the broad range of challenging problems in a robust and scalable manner. This adds a new piece of evidence that hBOA is indeed a robust and scalable optimization technique that should certainly make a difference in current computational optimization practice.

## Acknowledgments

The authors would also like to thank Fabien Alet, Martin Butz, Erick Cantú-Paz, Nikolaus Hansen, Petros Koumoutsakos, Fernando Lobo, Franz Rothlauf, Samarth Swarup, Matthias Troyer, and Dav Zimak for valuable comments and fruitful discussions.

The hBOA software, used by Martin Pelikan, was developed by Martin Pelikan and David E. Goldberg at the University of Illinois at Urbana-Champaign. Most experiments were done on the Asgard Beowulf cluster at the Swiss Federal Institute of Technology (ETH) at Zurich. 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. 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

- Ackley, D. H. (1987). An empirical study of bit vector function optimization. *Genetic Algorithms and Simulated Annealing*, 170–204.
- Allen, T. F. H., & Starr, T. (Eds.) (1982). *Hierarchy: Perspectives for ecological complexity*. Chicago, IL: University of Chicago Press.
- 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.
- Barahona, F. (1982). On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical, Nuclear and General*, 15(10), 3241–3253.
- Bosman, P. A. N. (2003). *Design and application of iterated density-estimation evolutionary algorithms*. Doctoral dissertation, Universiteit Utrecht, Utrecht, The Netherlands.
- Bosman, P. A. N., & Thierens, D. (2000). Continuous iterated density estimation evolutionary algorithms within the IDEA framework. *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 197–200.
- Buntine, W. L. (1991). Theory refinement of Bayesian networks. *Proceedings of the Uncertainty in Artificial Intelligence (UAI-91)*, 52–60.
- Cavicchio, Jr., D. J. (1970). *Adaptive search using simulated evolution*. Unpublished 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.
- Cooper, G. F., & Herskovits, E. H. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9, 309–347.
- 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).
- De Simone, C., Diehl, M., Jünger, M., & Reinelt, G. (1996). Exact ground states of two-dimensional  $+ - J$  Ising spin glasses. *Journal of Statistical Physics*, 84, 1363–1371.
- Deb, K., & Goldberg, D. E. (1994). Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, 10, 385–408.
- Etxeberria, R., & Larrañaga, P. (1999a). Global optimization using Bayesian networks. In Rodriguez, A. A. O., Ortiz, M. R. S., & Hermida, R. S. (Eds.), *Second Symposium on Artificial Intelligence (CIMAF-99)* (pp. 332–339). Habana, Cuba: Institute of Cybernetics, Mathematics, and Physics and Ministry of Science, Technology and Environment.
- Etxeberria, R., & Larrañaga, P. (1999b). Global optimization using Bayesian networks. *Second Symposium on Artificial Intelligence (CIMAF-99)*, 332–339.

- Faihe, Y. (1999). *Hierarchical problem solving using reinforcement learning : Methodology and methods*. Doctoral dissertation, University of Neuchâtel, Neuchâtel, Switzerland.
- Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical models* (pp. 421–459). Cambridge, MA: MIT Press.
- Galluccio, A., & Loeb, M. (1999a). A theory of Pfaffian orientations. I. Perfect matchings and permanents. *Electronic Journal of Combinatorics*, 6(1). Research Paper 6.
- Galluccio, A., & Loeb, M. (1999b). A theory of Pfaffian orientations. II. T-joins, k-cuts, and duality of enumeration. *Electronic Journal of Combinatorics*, 6(1). Research Paper 7.
- Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 5, 533–549.
- Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.
- Goldberg, D. E. (1994). *First flights at genetic-algorithm Kitty Hawk* (IlligAL Report No. 94008). Urbana, IL: University of Illinois at Urbana-Champaign.
- Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*, Volume 7 of *Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers.
- Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6, 333–362.
- Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the International Conference on Genetic Algorithms* (pp. 56–64). San Mateo, CA: Morgan Kaufmann.
- 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. *Proceedings of the International Conference on Genetic Algorithms (ICGA-87)*, 41–49.
- 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., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3), 231–253.
- Harik, G., & Lobo, F. (1999). A parameter-less genetic algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 1, 258–265.
- Harik, G. R. (1994). *Finding multiple solutions in problems of bounded difficulty* (IlligAL Report No. 94002). Urbana, IL: University of Illinois at Urbana-Champaign.
- Harik, G. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. Doctoral dissertation, University of Michigan, Ann Arbor. Also IlligAL Report No. 97005.

- Harik, G. R., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *Proceedings of the International Conference on Evolutionary Computation (ICEC-97)* (pp. 7–12). Piscataway, NJ: IEEE Press. Also IlliGAL Report No. 96004.
- Harik, G. R., & Goldberg, D. E. (1996). Learning linkage. *Foundations of Genetic Algorithms, 4*, 247–262.
- Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1998). The compact genetic algorithm. *Proceedings of the International Conference on Evolutionary Computation (ICEC-98)*, 523–528.
- Hartmann, A. K. (2001). Ground-state clusters of two, three and four-dimensional +/-J Ising spin glasses. *Phys. Rev. E*, *63*, 016106.
- Heckerman, D., Geiger, D., & Chickering, D. M. (1994). *Learning Bayesian networks: The combination of knowledge and statistical data* (Technical Report MSR-TR-94-09). Redmond, WA: Microsoft Research.
- Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J. F., & Kanal, L. N. (Eds.), *Uncertainty in Artificial Intelligence* (pp. 149–163). Amsterdam, London, New York: Elsevier.
- Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.
- Howard, R. A., & Matheson, J. E. (1981). Influence diagrams. In Howard, R. A., & Matheson, J. E. (Eds.), *Readings on the principles and applications of decision analysis*, Volume II (pp. 721–762). Menlo Park, CA: Strategic Decisions Group.
- Kardar, M., & Saul, L. (1994). The 2D +/-J Ising spin glass: Exact partition functions in polynomial time. *Nucl. Phys. B*, *432*, 641–667.
- Kargupta, H. (1995). *SEARCH, polynomial complexity, and the fast messy genetic algorithm*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL.
- Kargupta, H. (1996). The gene expression messy genetic algorithm. In *Proceedings of the International Conference on Evolutionary Computation (ICEC-96)* (pp. 814–819). Piscataway, NJ: IEEE Press.
- Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.
- Kulish, V. V. (2002). *Hierarchical methods: Hierarchy and hierarchical asymptotic methods in electrodynamics*. Dordrecht: Kluwer.
- Larrañaga, P., & Lozano, J. A. (Eds.) (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation*. Boston, MA: Kluwer.
- Laumanns, M., & Ocenasek, J. (2002). Bayesian optimization algorithms for multi-objective optimization. *Parallel Problem Solving from Nature*, 298–307.
- Mitchell, M. (1996). *An introduction to genetic algorithms*. Cambridge, MA: MIT Press.
- Monien, B., & Sudborough, I. H. (1988). Min cut is NP-complete for edge weighted trees. *Theoretical Computer Science*, *58*(1–3), 209–229.
- Mühlenbein, H. (1992). How genetic algorithms really work: I. Mutation and Hillclimbing. *Parallel Problem Solving from Nature*, 15–25.

- Mühlenbein, H., & Mahnig, T. (1999). FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4), 353–376.
- Mühlenbein, H., Mahnig, T., & Rodriguez, A. O. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5, 215–247.
- Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature*, 178–187.
- Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, 1(1), 25–49.
- Munetomo, M., & Goldberg, D. E. (1998). *Designing a genetic algorithm using the linkage identification by nonlinearity check* (Technical Report 98014). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.
- Naudts, B., & Naudts, J. (1998). The effect of spin-flip symmetry on the performance of the simple GA. *Parallel Problem Solving from Nature*, 67–76.
- Ocenasek, J. (2002). *Parallel estimation of distribution algorithms*. Doctoral dissertation, Faculty of Information Technology, Brno University of Technology, Brno.
- Pattee, H. H. (Ed.) (1973). *Hierarchy theory: The challenge of complex systems*. New York, NY: Braziller.
- Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.
- Pelikan, M. (2002). *Bayesian optimization algorithm: From single level to hierarchy*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL. Also IlliGAL Report No. 2002023.
- Pelikan, M., & Goldberg, D. E. (2000a). *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. (2000b). Hierarchical problem solving by the Bayesian optimization algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 275–282. Also IlliGAL Report No. 2000002.
- Pelikan, M., & Goldberg, D. E. (2000c). Research on the Bayesian optimization algorithm. In Wu, A. (Ed.), *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)* (pp. 216–219). San Francisco, CA: Morgan Kaufmann.
- Pelikan, M., & Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 511–518. Also IlliGAL Report No. 2000020.
- Pelikan, M., & Goldberg, D. E. (2003). Hierarchical boa solves ising spin glasses and maxsat. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, II, 1275–1286. Also IlliGAL Report No. 2003001.
- 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. (1999). BOA: The Bayesian optimization algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, I, 525–532. Also IlliGAL Report No. 99003.

- Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000a). Bayesian optimization algorithm, population sizing, and time to convergence. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 275–282. Also IlliGAL Report No. 2000001.
- Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000b). Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation*, 8(3), 311–341. Also IlliGAL Report No. 98013.
- Pelikan, M., Goldberg, D. E., & Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1), 5–20. Also IlliGAL Report No. 99018.
- 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., Goldberg, D. E., & Sastry, K. (2001). Bayesian optimization algorithm, decision graphs, and Occam's razor. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 519–526. Also IlliGAL Report No. 2000020.
- Pelikan, M., & Lobo, F. G. (1999). *Parameter-less genetic algorithm: A worst-case time and space complexity analysis* (IlliGAL Report No. 99014). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.
- Pelikan, M., & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. *Advances in Soft Computing—Engineering Design and Manufacturing*, 521–535.
- Pelikan, M., Sastry, K., & Goldberg, D. E. (2002). Scalability of the Bayesian optimization algorithm. *International Journal of Approximate Reasoning*, 31(3), 221–258. Also IlliGAL Report No. 2001029.
- Sacerdoti, E. D. (1975). The nonlinear nature of plans. In *Proceedings of the Fourth Annual International Joint Conference on Artificial Intelligence* (pp. 206–214). Tbilisi, Georgia, USSR.
- Sastry, K. (2001). *Efficient atomic cluster optimization using a hybrid extended compact genetic algorithm with seeded population* (IlliGAL Report No. 2001018). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.
- Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6, 461–464.
- Schwarz, J., & Ocenasek, J. (2000). A problem-knowledge based evolutionary algorithm KBOA for hypergraph partitioning. In *Proceedings of the Fourth Joint Conference on Knowledge-Based Software Engineering* (pp. 51–58). Brno, Czech Republic: IO Press.
- Simon, H. A. (1968). *The sciences of the artificial*. Cambridge, MA: The MIT Press.
- Stefik, M. (1981a). Planning and meta-planning (MOLGEN: Part 2). *Artificial Intelligence*, 16(2), 141–170.
- Stefik, M. (1981b). Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16(2), 111–140.
- Thierens, D. (1995). *Analysis and design of genetic algorithms*. Doctoral dissertation, Katholieke Universiteit Leuven, Leuven, Belgium.
- Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms (ICGA-93)*, 38–45.

- Thierens, D., Goldberg, D. E., & Pereira, A. G. (1998). Domino convergence, drift, and the temporal-salience structure of problems. *Proceedings of the International Conference on Evolutionary Computation (ICEC-98)*, 535–540.
- Van Hoyweghen, C. (2001). Detecting spin-flip symmetry in optimization problems. In Kallel, L., et al. (Eds.), *Theoretical Aspects of Evolutionary Computing* (pp. 423–437). Berlin: Springer.
- Yu, T.-L., Goldberg, D. E., & Chen, Y.-P. (2003). *A genetic algorithm design inspired by organizational theory: A pilot study of a dependency structure matrix driven genetic algorithm* (IlliGAL Report No. 2003007). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.