1-11-2014

Using Prior Knowledge and Learning from Experience in Estimation of Distribution Algorithms

Mark Walter Hauschild
University of Missouri-St. Louis, markhauschild@gmail.com

USING PRIOR KNOWLEDGE AND LEARNING FROM EXPERIENCE IN ESTIMATION OF DISTRIBUTION ALGORITHMS

BY

MARK HAUSCHILD

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mathematics and Computer Science
in the Graduate School of the
University of Missouri–St. Louis, 2013

St. Louis, Missouri

Doctoral Committee:

Cezary Z. Janikow, Ph.D.
Sanjiv K. Bhatia, Ph.D.
Uday K. Chakraborty, Ph.D.
Henry Kang, Ph.D.

Abstract

Estimation of distribution algorithms (EDAs) are stochastic optimization techniques that explore the space of potential solutions by building and sampling explicit probabilistic models of promising candidate solutions. One of the primary advantages of EDAs over many other stochastic optimization techniques is that after each run they leave behind a sequence of probabilistic models describing useful decompositions of the problem. This sequence of models can be seen as a roadmap of how the EDA solves the problem. While this *roadmap* holds a great deal of information about the problem, until recently this information has largely been ignored. *My thesis is that it is possible to exploit this information to speed up problem solving in EDAs in a principled way.*

The main contribution of this dissertation will be to show that there are multiple ways to exploit this problem-specific knowledge. Most importantly, it can be done in a principled way such that these methods lead to substantial speedups without requiring parameter tuning or hand-inspection of models.

Acknowledgments

The first person I must thank for getting me to this point is my mother Marilyn. She instilled in me a love for computer science and mathematics from an early age. Her gift of a pascal compiler when I was young probably did more to put me on this path than any single thing I can imagine.

Of course, none of my research would have been possible without my first thesis advisor, Martin Pelikan. Initially I only intended to do a simple independent study on genetic algorithms. Instead, Martin put me on the track of doing research into EDAs and the models that they produce. From the very start he gave me every opportunity to succeed and the debt of gratitude that I owe him can never be repaid.

Lastly, of course a huge thanks to Cezary Janikow for helping me finish this thesis. It has been a long road and without his help at the end it might never have been finished.

The work was sponsored by the National Science Foundation under grants ECS-0547013 and IIS-1115352, by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096, and by the University of Missouri in St. Louis through the High Performance Computing Collaboratory sponsored by Information Technology Services, and the Research Award and Research Board programs. Most experiments were performed on the Beowulf cluster maintained by ITS at the University of Missouri in St. Louis and the HPC resources at the University of Missouri Bioinformatics Consortium. hBOA was developed at the Illinois Genetics Algorithm Laboratory at the University of Illinois at Urbana-Champaign. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Air Force Office of Scientific Research, or the U.S. Government.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Estimation of distribution algorithms (EDAs) (Baluja, 1994; Larrañaga and Lozano, 2002; Pelikan et al., 2002a; Pelikan et al., 2006c) are population-based stochastic optimization techniques that replace the traditional variation operators of genetic algorithms with a model building phase. In this model building phase, a probabilistic model is built that models promising solutions and then this model is sampled to generate new candidate solutions. By using this model building stage to learn the important linkages between important parts of the problem, EDAs have been able to solve a broad variety of problems. Whether it be to solve test problems such as multiobjective knapsack (Shah and Reed, 2010), physics based problems such as finding the ground states of spin glasses (Pelikan and Hartmann, 2006) or real-world application problems such as military antenna design (Yu et al., 2006), groundwater remediation design (Arst et al., 2002; Hayes and Minsker, 2005), amino-acid alphabet reduction for protein structure prediction (Bacardit et al., 2007), identification of clusters of genes with similar expression profiles (Peña et al., 2004), economic dispatch (Chen and p. Chen, 2007), forest management (Ducheyne et al., 2004), portfolio management (Lipinski, 2007), cancer chemotherapy optimization (Petrovski et al., 2006) and environmental monitoring network design (Kollat et al., 2008), EDAs have been shown to solve many problems with better results than other techniques were able to achieve.

While EDAs have many advantages over standard genetic algorithms (Larrañaga and Lozano, 2002; Pelikan et al., 2006c), one in particular this thesis will focus on is that at the end of an EDA run, a series of probabilistic models of our solution space have been built, which series holds a great deal of information about the problem. These models provide practitioners with a *roadmap of how the EDA solved the problem*. Although such information should be useful for effective efficiency enhancement and it has often been argued that using problem-specific knowledge should significantly improve EDA performance (Schwarz and Ocenasek, 2000; Baluja, 2006), many of the previous attempts to do so have relied on hand-inspection of previous models and required a set of parameters to be tuned to ensure efficient performance (Baluja, 2006; Schwarz and Ocenasek, 2000).

1.1 Thesis Objectives

There are three primary objectives of this thesis:

- Show that the models in EDAs capture important regularities of problem structure.
- Design principled methods to speed up EDA problem solving by exploiting the knowledge gained from previous runs of an EDA.
- Design specialized crossover operators to take advantage of these learned regularities to speed up problem solving.

Specifically, this thesis will show that it is possible to exploit prior knowledge in a principled way to speed up problem solving in EDAs. First, results will show that during the model building phase of EDAs, they capture important problem regularities. Most importantly, these regularities are shared between similar problem instances. By exploiting this information gained from either previous runs of an EDA or trial runs on similar problems, this thesis will then show that it is possible to speed up problem solving. Additionally, it will be shown that this information gathered from previous runs can also be used to develop specialized crossover operators that can be used in genetic algorithms. While this work mainly focuses on the hierarchical Bayesian Optimization Algorithm (hBOA), one of the most powerful EDAs currently on the market, the methods should be applicable to many other EDAs.

1.2 Road Map

The thesis starts with chapter 2 introducing the basic concepts of EDAs in general and the EDA that this thesis will focus on, the hierarchical Bayesian Optimization Algorithm. Chapter 3 focuses on the model accuracy in hBOA and examines whether hBOA is able to capture important regularities in problem structure. Chapters 4 and 5 present experimental results of using several different methods to exploit prior knowledge to speed up subsequent runs of hBOA. Chapter 6 develops specialized crossover operators from information either gathered from previous knowledge of the problem or by trial runs of an EDA and then this crossover operator is used in a genetic algorithm (GA). Chapter 7 discusses avenues for future research in learning from experience in EDAs. Finally, the thesis closes by providing the conclusions in chapter 8. The following subsections present the content of each chapter in greater detail.

1.2.1 Chapter 2: Estimation of Distribution Algorithms and hBOA

Chapter 2 starts by describing the basic EDA procedure. Then a simple EDA is simulated to show how an EDA works in practice. A second EDA simulation is then shown to motivate the need for linkage learning in EDAs. The hierarchical Bayesian Optimization Algorithm (hBOA), the EDA used for most of this thesis, is then described in detail. Next, the most common efficiency enhancement techniques used in EDAs are briefly surveyed. Lastly, the central importance of models to EDAs is discussed in detail.

1.2.2 Chapter 3: Model Accuracy in hBOA

Chapter 3 examines model accuracy in hBOA. The chapter starts by looking at four types of problems, each with different problem structure. Each of these problems is discussed in detail, with an examination of what a "perfect" model for this type of problem would be. Then hBOA is used to solve these problems. The resulting models are then analyzed against the theoretical "perfect" models to examine the accuracy of hBOA model building. In addition, the results are examined to show that important regularities are shared between models in different instances. Lastly, a preliminary examination of the possible benefits of restricting hBOA model building is done.

1.2.3 Chapter 4: Hard Biasing hBOA Model Building

Chapter 4 proposes two methods to bias model building in hBOA based on information gathered in previous runs. This chapter considers methods that bias by limiting the number of total edges considered during hBOA model building. The first method, the probability coincidence matrix (PCM), is described and then used to bias model building on several test problems, with the experimental results showing significant speedups. The second method, using a distance metric to bias model building, is then discussed and tested, with the results again showing significant speedups on a wide variety of test problems.

1.2.4 Chapter 5: Soft Biasing hBOA Model Building

Chapter 5 proposes a method to bias the model building metric itself in hBOA based on the probabilistic models from previous EDA runs on problems of a similar type. This method uses a bias introduced through structural priors of Bayesian metrics using the split probability matrix (SPM). This method is then tested on several test problems. The results show that this method requires no unique parameter settings to lead to speedups.

1.2.5 Chapter 6: Network Crossover

Chapter 6 proposed a method to construct a network crossover that could be used by a GA to easily incorporate problem-specific knowledge. A GA using this network crossover was tested against the hBOA and other common crossover operators on NK landscapes and trap-5. The results showed that the network crossover outperformed all other algorithms on nearest neighbor NK landscapes and trap-5. However, for unrestricted NK landscapes hBOA performed the best.

1.2.6 Chapter 7: Future Work

Chapter 7 discusses important topics of future research.

1.2.7 Chapter 8: Conclusions

Chapter 8 concludes the thesis and lists its most important consequences.

Chapter 2

Estimation of Distribution Algorithms and hBOA[1]

Estimation of distribution algorithms (EDAs) (Baluja, 1994; Mühlenbein and Paaß, 1996; Larrañaga and Lozano, 2002; Pelikan et al., 2002a; Hauschild and Pelikan, 2011) are stochastic optimization algorithms that explore the space of candidate solutions by sampling an explicit probabilistic model constructed from promising solutions found so far. This chapter starts by reviewing the EDA procedure in general and then describes the primary EDA used in this thesis, the hierarchical Bayesian Optimization Algorithm (hBOA) (Pelikan and Goldberg, 2001; Pelikan, 2005).We then describe the central importance of models to EDAs. Lastly, we discuss the importance of efficiency enhancements to speed up EDA performance and present a brief survey of some of the techniques.

To start, we describe general black-box optimization in Section 2.1. Then we describe the general EDA procedure in Section 2.2. Section 2.2.1 then describes a simple univariate EDA and uses it to solve the onemax problem. In Section 2.2.2, we show an example that the simple model used by the multivariate EDA are unable to solve the problem but a multivariate model is able to, motivating the need for more complex models in problem solving. Section 2.3 describes hBOA, one of the strongest multivariate EDAs currently available and the primary algorithm that this thesis will focus on. Section 2.5 discusses the central nature of models and model building to EDAs. Section 2.4 briefly covers some efficiency enhancements used to speed up EDA problem solving. Finally, Section 2.7 summaries the chapter.

2.1 Black-box optimization

In this work we will be primarily be focused on solving optimization problems, where the goal is to find the optimal solution to a problem from the set of all possible solutions. In order to solve these problems, only two things are necessary: (1) a way to represent all possible solutions to the problem and (2) a way to determine which solutions are better than others. In the rest of this thesis we will call the representation of a solution to a problem as a *candidate solution*. The function that ranks solutions and determines how fit

[1]This chapter is based in part on work previously published in Hauschild, M., & Pelikan, M. (2011). An Introduction and Survey of Estimation of Distribution Algorithms. *Swarm and Evolutionary Computation*, *1*(3), 111-128

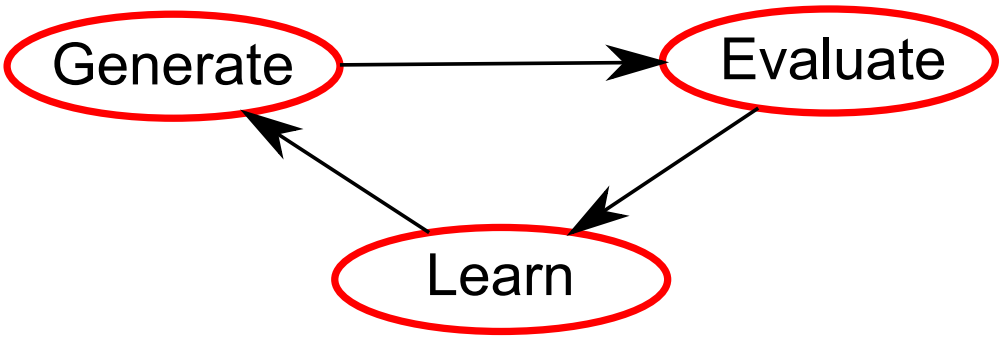


Figure 2.1: An illustration of the process used by a black-box algorithm.

they are will be called the *fitness function*.

For example, consider the minimum vertex cover problem, where the goal is to find the smallest number of vertices in a graph such that every edge of the graph touches one of these vertices. To represent all possible solutions, we could use a binary string whose length equals number of vertices in the graph, with a 1 in any position representing that vertex being part of the set of our minimum cover. It is then quite easy to rank the valid solutions using a fitness function, with the fitness function returning the number of 1s in the solution and better solutions having fewer. Those solutions that are not valid (where all edges are not incident to a vertex) could then be assigned a fitness equal to the size of the string (so that all valid solutions are superior). In this problem, we represented each solution as a binary string, and in the rest of this work all candidate solutions will be represented by binary strings. This is not as restrictive as one might think, as it is possible to code problems of other cardinality alphabets as binary strings. Even in the cases of real-valued functions, it is still possible to discretize the variables and represent solutions as binary strings.

By defining the problem in this way, it is not necessary to know exactly how good solutions are constructed. Instead, in order to solve a problem, we must simply develop a way to generate new candidate solutions and then evaluate them to determine the superior solutions. This is often called *black-box optimization*, because we are not requiring any specific information on the mechanics of the problem. All we are requiring is a way to represent solutions, a way to generate new solutions and a way to evaluate or rank them.

Black-box optimization works over time by continually generating new candidate solutions and evaluating them to determine the better solutions. An example of this process is shown in Figure 2.1. We see that the algorithm starts by *generating* new solutions, the solutions are then *evaluated* to find promising solutions and then the algorithm attempts to *learn* what made these solutions most promising. This information is then used to generate new solutions and the cycle repeats until a solution of sufficient quality is reached.

The difficulty in black-box optimization is often in determining how to go about generating new higher quality solutions. The simplest way would be to simply generate random solutions until the optimum was found. A much better way might be to use information from good solutions generated so far to generate new solutions that shared many of the characteristics of those good solutions. Estimation of distribution algorithms (EDAs) do just that. We will see that given only binary strings of fixed length and a fitness function, that it is possible to generate new but different candidate solutions that share similarities with promising candidate solutions generated earlier.

2.2 Estimation of Distribution Algorithms

EDAs typically start by generating a population of candidate solutions to the problem, with the population randomly generated uniformly over all admissible solutions. The population members are then ranked using a *fitness function*, which gives each solution a numerical ranking indicating the quality of the solution, with higher fitness values indicating better solutions. The EDA then selects a promising subset of the population using a *selection* operator. While the type of selection operators vary, one of the most common is truncation selection, which selects a certain percentage of the best solutions. Next a probabilistic model is constructed which attempts to estimate the probability distribution of the promising selected solutions. New solutions are then generated by sampling the distribution encoded by this model, with new solutions incorporated into the old population or used to entirely replace the old population. This process is repeated until some termination criteria is met, which is usually when a solution of sufficient quality is reached or when the number of iterations reaches some threshold. Each iteration of the algorithm is usually referred to as one *generation* of the EDA. The basic EDA procedure is outlined in Algorithm 1.

Algorithm 1 EDA pseudocode

```
g ← 0
randomly generate initial population P(0)
while (not done) do
   select population of promising solutions S(g) from P(g)
   build probabilistic model M(g) from S(g)
   sample M(g) to generate new candidate solutions O(g)
   incorporate O(g) into P(g)
   g ← g + 1
end while
```

While it obvious that EDAs share many similarities with other search algorithms such as GAs, for example with the use of a population and a replacement operator, the incorporation of the model building phase is the key difference. The model building phase of an EDA attempts to capture the probability distribution

of promising solutions and therefore discovering how the different parts of the problem interact to form a good solution. If done correctly, sampling this model should generate quality solutions that share many characteristics with the promising subset of solutions picked by the selection operator. However, this is not an easy task as the goal is not to perfectly represent the population of promising solutions. If this was the case, then the algorithm would not do sufficient exploration of the search space and might simply repeat the promising solutions of the last generation. Instead, the goal is to capture the features of the promising solutions that make these solutions better than other candidate solutions. Also, since new probabilistic models are usually built each generation, it is of paramount importance that the models be generated and sampled efficiently.

2.2.1 Solving Onemax with a Simple EDA

To help see the basic EDA procedure in action, let us look at a simple example of an EDA solving the onemax problem. In onemax, candidate solutions are represented by vectors of n binary variables and the fitness is computed by

$$onemax(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} X_i \ , \tag{2.1}$$

where n is the number of variables and X_i is the i^{th} variable in the problem (i^{th} position in the input binary string). This function has one global optimum in the string of all 1s.

In this example, a population size of $N = 6$ will be used, with each solution having $n = 5$ binary variables. As our selection operator we will use truncation selection with $\tau = 50\%$ to select the most promising solutions, which will ensure that the 50% best fit solutions are selected. The next important step is to pick the model that we will use to estimate the probability distribution of the promising solutions. In this case, let us a *probability vector* model, which stores the probability of a 1 in each position of the solution strings. While this model might seem simplistic, it provides a fast and efficient model for solving onemax and other optimization problems, mainly because it is based on the assumption that all problem variables are independent. In this probability vector model, the probability p_i of a 1 in position i is set to the proportion of selected solutions containing a 1 in this position. To sample this model (generate new solutions), for each position i in the new candidate solution, a 1 is generated in this position with probability p_i. For example, if $p_3 = 0.6$, we generate a 1 in the 3rd position of a new candidate solution with the probability 60%. On each generation (iteration), the current model is sampled $N = 6$ times to generate a completely new population of size $N = 6$. This simulation is outlined in Figure 2.2.

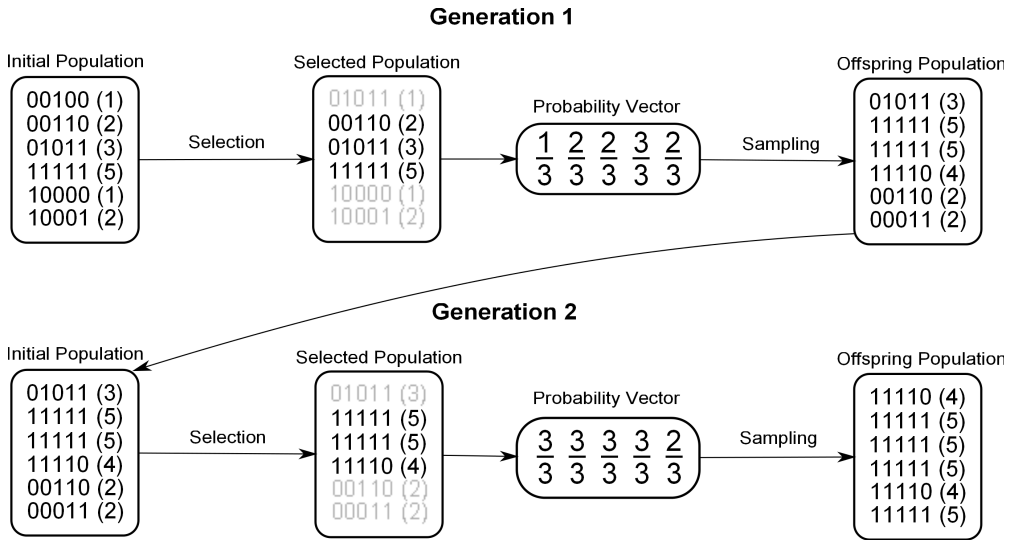


Figure 2.2: The first two generations of a probability-vector model EDA solving onemax.

We see from Figure 2.2 that the initial population is of poor quality, with four solutions having a fitness of 2 or less. However, even after just one iteration of the algorithm it is clear that the population is improving. The average fitness of the offspring population is higher, with significantly more 1s than the original population and even contains several copies of the optimal solution of 11111. Since the probability of a 1 in most positions has increased, in the second generation the probability vector has increased in value in most positions, which should lead to a higher probability of generating the optimal solution. It is easy to see that if the simulation was continued for more generations, that eventually the probability vector model would converge to generating a 1 in all positions, thereby generating only the global optimum.

While the previous example was small, we can show that this procedure works on larger problems. To show this example on a larger problem, the probability of ones in each generation from an example run of an EDA is shown in Figure 2.3. In this problem we are using a solution string of 50 variables and 100 individuals in the population, yet we still see that over time the probability of a 1 in any given position increases over time. While we do see the probability of a 1 in some positions fluctuate initially, eventually the population is overtaken by a 1 in all positions.

As long as the population is large enough (Harik et al., 1997; Goldberg, 2002), the probability vector EDA is able to solve onemax and many other optimization problems. It is also able to solve these problems very efficiently, as building and sampling a probability vector is quite fast. While this example showed that a model using a probability vector was able to solve onemax efficiently, what happens when this simple probability vector is not sufficient to model the important characteristics of promising solutions?

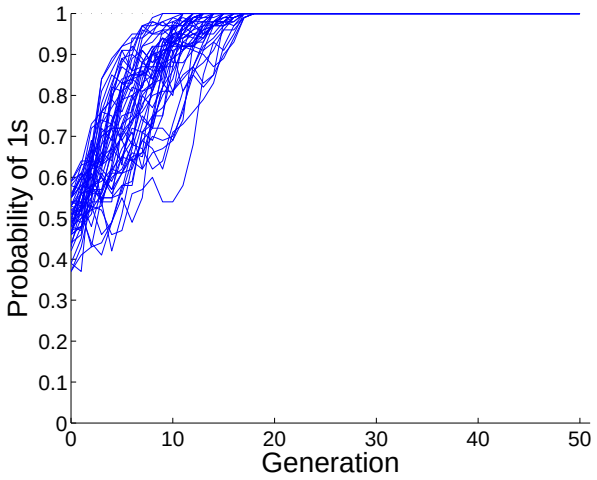


Figure 2.3: Probability of a 1 in a given position by generation in a probability-vector EDA when solving the onemax problem with a population of $N = 100$ and a problem size of $n = 50$.

2.2.2 Linkage Learning EDAs: Using an EDA to Solve Trap-5

While the probability vector model used in the previous example has some strengths, it is based around the assumption that the value of each variable is independent from other variables in the problem (that is, it assumes a perfect decomposition). This is often the case for a great many problems. However, when this condition is not met, the probability vector model can lead the EDA to failure. To help illustrate this, let us consider the concatenated trap of order 5 (trap-5) (Ackley, 1987; Deb and Goldberg, 1991). In trap-5, the input string is first partitioned into independent groups of 5 bits each. The contribution of each group of 5 bits (trap partition) is computed as

$$trap_5(u) = \begin{cases} 5 & \text{if } u = 5 \\ 4 - u & \text{otherwise} \end{cases}, \tag{2.2}$$

where u is the number of 1s in the input string of 5 bits, with the total fitness being the sum of all trap partitions. While the trap-5 function has the same global optimum as the onemax function, the string of all 1s, it also has $(2^{n/5} - 1)$ other local optima, namely those strings where all bits in at least one trap partition are 0 and all bits in each of the remaining partitions are either 0 or 1. Trap-5 necessitates that all bits in each group be treated together, because statistics of lower order are misleading (Thierens and Goldberg, 1993); this means that trap-5 can provide an excellent example of the limitations of the probability vector as a model.

In trap-5, using a probability vector and assuming that the bits are independent will lead the algorithm to failure. For onemax, the average fitness of a solution with a 1 in any position in better than a 0 in that

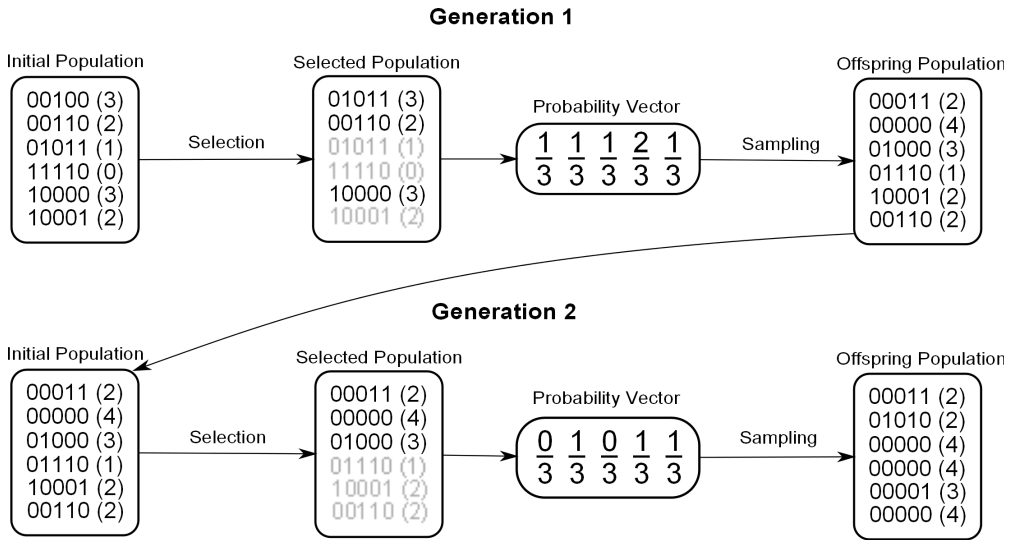


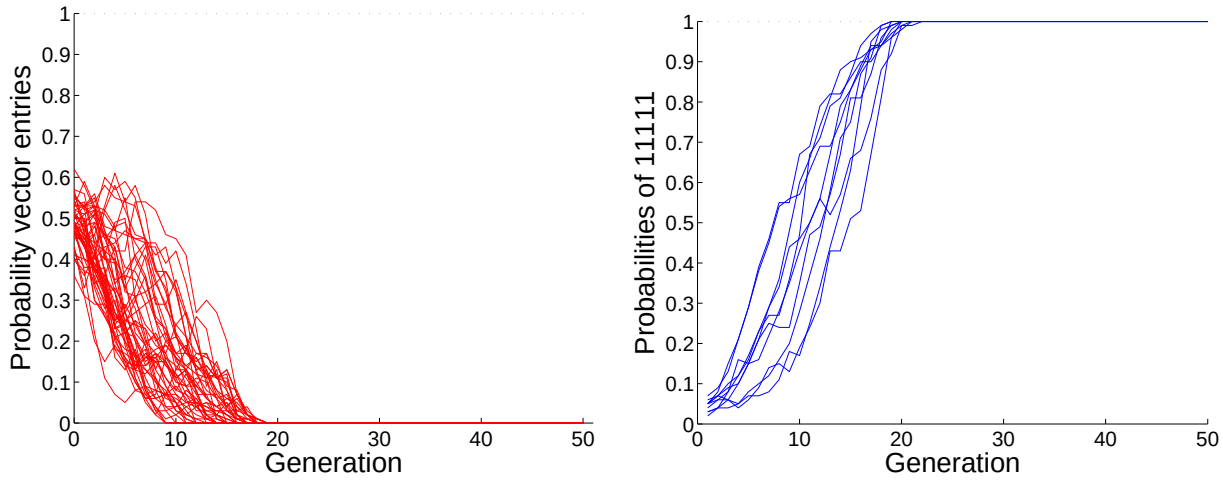
Figure 2.4: The probability vector entries of a probability-vector model EDA attempting to solve trap-5.

position. However this situation is reversed for trap-5. Instead, the average fitness of a solution with a 0 in any position in greater than 1 in any position. This leads to the probability vector being strongly biased towards a 0 in all positions when using a simple probability vector on trap-5.

An illustration of this effect can be seen in Figure 2.4, which shows two generations of a simple EDA attempting to solve trap-5. We see that in the first generation and the start of the second generation that the sampling has given solutions that are more fit than the first generation. However, even though these solutions have high fitness, the probability of a one in any position decreases, as we can see from the probability vector. Over time the probability vector gets farther and farther away from the correct distribution that would generate the global optimum of all 1s. The population after the second sampling shows this clearly, with three individuals that contain no 1s.

Let us see if this behavior repeats itself when the problem is larger. Figure 2.5a shows the results of using an EDA with a probability vector to solve a trap-5 problem with 50 bits and a population of size $N = 100$. We see that the probability vector entries on average decrease over time. Indeed, by generation 27 the EDA is simply generating strings that are all 0s. A possible solution one might consider is to simply increase the population. However, a larger population would succeed only in making the decrease in the probability of a 1 in each position more precise.

It is clear that the probability vector is not an effective model for trap-5. However, what if we could use a model that respected the linkages between bits in the same trap partition? If the algorithm had a way to learn the structure of trap-5, then it could treat each trap partition as if it was its own variable. In this case, instead of storing the probability of each bit, the algorithm would store the probability of each combination



(a) Probability vector EDA

(b) Marginal product model EDA with linkage groups corresponding to trap partitions.

Figure 2.5: Statistics from two EDAs using different probabilistic models to solve trap-5 on a problem of $n = 50$ bits.

of 5 bits in each trap partition. We would then expect that the average fitness of a solution with 1s in all of a trap partition to be higher than the average fitness of solutions with at least one 0 in that trap partition. To generate a string from this new model, each trap partition would be sampled to generate 5 bits at once based on the probability of any combination of the 5 bits. Over time we would expect that the proportion of trap partitions with 1s in all positions in the population would increase.

Probabilistic models that combine variables into independent linkage groups are often called marginal product models (Harik, 1999). One of the most popular of these types of EDAs is the extended compact genetic algorithm (ECGA) (Harik, 1999). By finding the important linkages in each trap partition and then combining them into independent linkage groups, ECGA can be extremely effective on trap-5. To help illustrate the dramatic difference that considering linkage groups can have on solving trap-5, Figure 2.5b shows the probability of a trap partition having all 1s in the population of a run of ECGA with $n = 50$ and $N = 100$. The results clearly show that by treating each trap partition as a single variable, the EDA is able to quickly solve trap-5, with the performance being quite similar to the onemax example shown in Figure 2.3.

This example clearly demonstrates that if an EDA is able to build a model each generation that captures the important features of selected solutions and generates new solutions with these features, then the EDA should be able to quickly converge to the optimum (Mühlenbein and Mahnig, 1999). In addition to just simply solving the problem, we also see in these examples that the model learned by the EDA each generation gives us information about the problem structure. For example, after solving the trap-5 function, it should be

possible to examine the marginal product model and determine the location and size of each trap partition. This is an important insight, as it is the first example that shows how it should be possible to mine EDA model building to gain information about a problem's structure. For more information on EDAs, please see (Hauschild and Pelikan, 2011).

2.3 Hierarchical Bayesian Optimization Algorithm

One of the most powerful EDAs is the hierarchical Bayesian optimization algorithm (hBOA) (Pelikan and Goldberg, 2001; Pelikan, 2005), which is able to solve many difficult problems scalably and reliably. hBOA evolves a population of candidate solutions represented by fixed-length strings over a finite alphabet (for example, binary strings). The initial population is generated at random according to the uniform distribution over the set of all potential solutions. Each iteration (generation) starts by selecting promising solutions from the current population using any standard selection method. For example we use truncation selection with threshold $\tau = 50\%$, which selects the best half of the current population based on the objective function. After selecting the promising solutions, hBOA builds a Bayesian network (Howard and Matheson, 1981; Pearl, 1988) with local structures in the form of decision trees (Chickering et al., 1997; Friedman and Goldszmidt, 1999) as a model of these solutions. New solutions are generated by sampling the built network. These are then incorporated into the original population using restricted tournament replacement (RTR) (Harik, 1995), which ensures effective diversity maintenance. The window size w in RTR was set as $w = \min\{n, N/20\}$, where n is the number of decision variables and N is the population size, as suggested in ref. (Pelikan, 2005). The next iteration is then executed unless some predefined termination criteria are met. The pseudocode of hBOA is shown in Algorithm 2.

Algorithm 2 hBOA pseudocode

$g \leftarrow 0$
randomly generate initial population $P(0)$
while (termination criteria not met) **do**
  select population of promising solutions $S(g)$ from $P(g)$
  build the network B by constructing decision graphs G_i encoding cond. probabilities.
  generate a set of new strings $O(g)$ according to the joint distribution encoded by B with dec. graphs G_i.
  create a new population $P(g+1)$ using RTR to incorporate each member of $O(g)$ into $P(g)$.
  $g \leftarrow g + 1$
end while

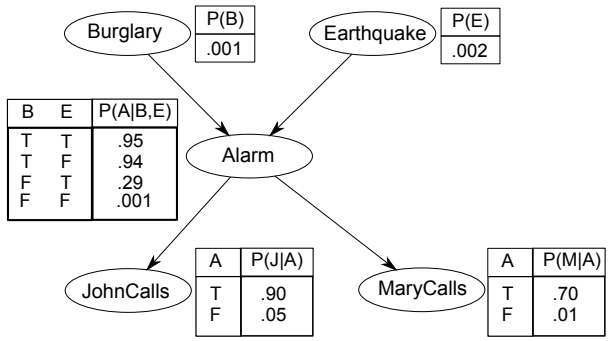


Figure 2.6: A simple example of Bayesian network structure and its associated conditional probabilities.

2.3.1 Bayesian Networks

Bayesian networks (Pearl, 1988; Howard and Matheson, 1981) combine graph theory, probability theory and statistics to provide a flexible and practical tool for probabilistic modeling and inference. hBOA use Bayesian networks to model promising solutions found so far and sample new candidate solutions.

A Bayesian network (BN) consists of two components: (1) *Structure*, which is defined by an acyclic directed graph with one node per variable and the edges corresponding to conditional dependencies between the variables, and (2) *parameters*, which consist of the conditional probabilities of each variable given the variables that this variable depends on. A BN with n nodes encodes a joint probability distribution of n random variables $X_1, X_2, \ldots, X_n$:

$$p(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} p(X_i|\Pi_i), \tag{2.3}$$

where Π_i is the set of variables from which there exists an edge into X_i (members of Π_i are called parents of X_i), and $p(X_i|\Pi_i)$ is the conditional probability of X_i given Π_i.

Figure 2.6 shows an example of a simple Bayesian network structure (Pearl, 1988). It is easy to see that this network encodes many conditional dependencies. For example, the probability of an alarm depends on whether there has been a burglary and/or an earthquake. The probability that John calls depends on whether an alarm has gone off. In addition to these conditional dependencies, the network also encodes a number of independence assumptions. This is shown by the probability of a burglary being independent from the probability of an earthquake. In the same way, the probability that John will make a call is independent from the probability of Mary making a call. A more complex conditional assumption is that given whether an alarm went off, the probability of Mary calling is independent from whether or not a burglary or an earthquake happened.

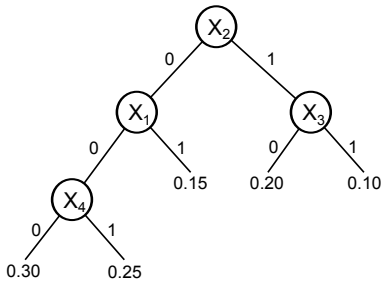


Figure 2.7: A simple possible decision tree that could be generated by hBOA to express dependencies and their associated conditional probabilities.

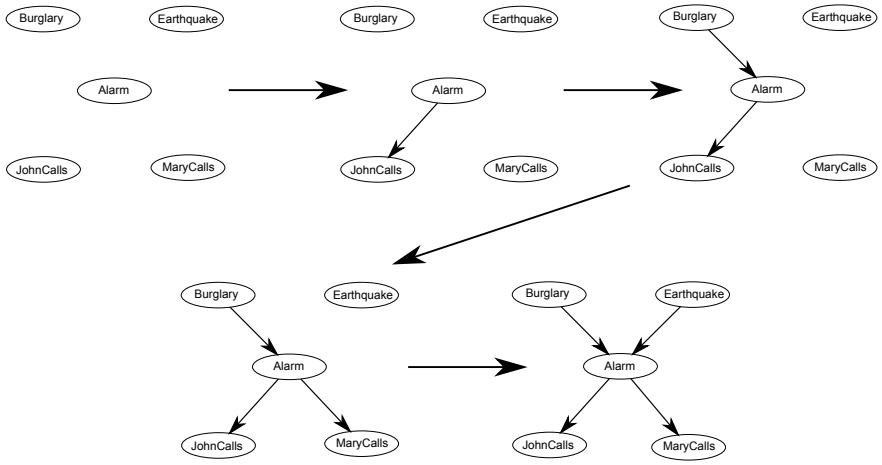


Figure 2.8: An example of the possible steps a greedy algorithm might use to build the network structure in Figure 2.6.

To fully specify the Bayesian network structure and associated conditional probabilities, hBOA uses decision trees. In BNs with decision trees, the conditional probabilities $p(X_i|\Pi_i)$ are encoded by a decision tree T_i; for n variables, there are n decision trees. Each internal node of the decision tree T_i is labeled by a variable $X_j \in \Pi_i$ where $j \neq i$. Children of a node labeled by X_j correspond to disjoint subsets of the potential values of X_j; for each value of X_j, there is one child corresponding to this value. Each traversal of a decision tree T_i for X_i thus corresponds to a constraint on the values of other variables from Π_i. Each leaf of T_i then stores the probabilities of X_i given the constraint defined by the traversal of T_i ending in this leaf. An example of a decision tree for variable X_0 is given in Figure 2.7. A decision tree like this fully specifies all conditional probabilities given the parents of X_0. In this case, if X_2 is 1, then X_0 is twice as likely to be a 0 instead of a 1.

To learn the structure of a Bayesian network with decision trees, a simple greedy algorithm is used (Heckerman et al., 1994; Chickering et al., 1997). The greedy algorithm starts with an empty network represented by single-node decision trees. Each iteration splits one leaf of any decision tree that improves the score

of the network most until no more improvement is possible. Pseudocode for the greedy algorithm used to construct the Bayesian network is shown in Algorithm 3. Figure 2.8 shows an example sequence to learn the Bayesian network structure from Figure 2.6. We can see that only one dependency is found at each step, but even with this restriction, complicated networks can still be found. For more details on learning BNs with decision trees, see (Chickering et al., 1997; Friedman and Goldszmidt, 1999; Pelikan, 2005).

Algorithm 3 Greedy algorithm for Bayesian network construction

```
Initialize a decision graph G_i for each node X_i to a graph containing only a single leaf.
Initialize the network B into an empty network.
loop
  Choose the best split that does not result in a cycle in B.
  If the best split does not improve the score, finish.
  Execute the chosen operator.
  Add the corresponding edge into B.
end loop
```

Learning the structure (decision trees) is the most challenging task in hBOA model building (Pelikan, 2005). That is why when enhancing the efficiency of model building in hBOA, the primary focus is typically on speeding up the learning of the structure of the network.

2.4 Efficiency Enhancements

While EDAs provide scalable solutions to many problems that are intractable with other techniques, solving enormously complex problems often necessitates that additional efficiency enhancement (EE) (Goldberg, 2002; Sastry et al., 2006; Pelikan, 2005) techniques are used. There are two primary computational bottlenecks in EDAs: (1) fitness evaluation and (2) model building. To alleviate these bottlenecks, numerous efficiency enhancement techniques (Goldberg, 2002; Sastry et al., 2006; Pelikan, 2005) can be incorporated into an EDA, many of which can be adopted from genetic algorithms or other evolutionary algorithms. Efficiency enhancement techniques for EDAs can be roughly divided into the following categories (Pelikan, 2005):

1. Parallelization (Cantú-Paz, 2000; Ocenasek, 2002; Ocenasek et al., 2006; Mendiburu et al., 2006).
2. Evaluation relaxation (Smith et al., 1995; Sastry et al., 2001; Pelikan and Sastry, 2004; Sastry and Goldberg, 2004).
3. Hybridization (Hartmann, 1996; Pelikan and Goldberg, 2003; Pelikan and Hartmann, 2006).
4. Time continuation (Goldberg, 1999; Srivastava and Goldberg, 2001; Goldberg, 2002).

5. Sporadic and incremental model building (Ocenasek and Schwarz, 2000; Pelikan, 2005; Pelikan et al., 2006d; Etxeberria and Larrañaga, 1999).

6. Incorporating problem-specific knowledge and learning from experience (Schwarz and Ocenasek, 2000; Sastry, 2001a; Pelikan and Goldberg, 2003; Mühlenbein and Mahnig, 2002; Baluja, 2006).

While all these techniques are important in general, in this section we will describe in more detail those that deal with the model building in EDAs, in particular *sporadic and incremental model building* and *incorporating problem-specific knowledge and learning from experience*. For a more detailed discussion of other efficiency enhancement techniques in EDAs, please see Sastry et al., 2006).

2.4.1 Sporadic and Incremental Model Building

Model building is often the bottleneck in EDA performance and consists of two parts: (1) learning the structure and (2) learning the parameters of the identified structure. In most cases, learning the structure of the model is the more complex task (Ocenasek and Schwarz, 2000; Pelikan, 2005), and so most efficiency enhancement techniques focus on lowering the time spent learning the structure. One of the ways to do this is to exploit the fact that model structure does not often change between consequent generations. This is done by *sporadic model building*, where the model structure is only learned periodically during the run (Pelikan et al., 2008), instead of every generation.

For many problems the models change gradually over time, starting with a fairly simple model that grows more complex over time. In this case, it is better to incrementally change the model each generation instead. This *incremental model building* (Etxeberria and Larrañaga, 1999) does away with the overhead of having to generate the entire model from scratch each generation, which can result in an overall improvement in performance.

2.4.2 Incorporating Problem-specific Knowledge and Learning From Experience

EDAs typically do not require any information about the problem being solved except for the representation of candidate solutions and the fitness function. Nonetheless, if problem-specific information is available, it may be possible to use this information to improve performance of these algorithms significantly. There are two basic approaches to speed up EDAs by incorporating problem-specific knowledge: (1) bias the procedure for generating the initial population (Schwarz and Ocenasek, 2000; Sastry, 2001a; Pelikan and Goldberg, 2003) and (2) bias or restrict the model building procedure (Schwarz and Ocenasek, 2000; Mühlenbein and

Mahnig, 2002; Baluja, 2006).

One of the most straightforward ways to bias the initial population in an EDA towards good solutions is to use *seeding* (Schwarz and Ocenasek, 2000; Pelikan and Goldberg, 2003; Sastry, 2001a). With seeding, high-quality solutions gathered in some way are put into the initial population along with other solutions generated randomly. By improving the initial fitness in the population, the resulting model quality should also be improved. These initial good solutions can either be obtained from previous runs on similar problems or given by specialized heuristics (Schwarz and Ocenasek, 2000; Pelikan and Goldberg, 2003). They can also be obtained by scaling up smaller instances of a problem (Sastry, 2001a).

While seeding can be quite successful when used with a broad variety of algorithms, EDAs with their models provide additional options to use prior information in a principled way to improve the quality of model generation. One way is to use problem-specific knowledge about the problem to bias model building in some way. Schwarz and Ocenasek, 2000) biased BOA model building by using the underlying graph when solving graph bipartitioning. By making BOA prefer those edges contained in this underlying graph, model quality was improved, which resulted in significant speedups. In later work by Mühlenbein and Mahnig, 2002), edges not in the underlying graph in graph bipartitioning were strictly dissallowed and again the result was significant speedups. Baluja, 2006) improved the performance of BOA on the graph-coloring problem by also dissallowing edges that did not correspond to the underlying problem graph.

2.5 Importance of Models in EDAs

Model building is of central importance to EDA design and problem solving. The process of model building and the methods for learning and sampling these models is what separates EDAs from other algorithms, and the types of models used often differentiate one EDA from another and determine what types of problems a particular EDA will excel at solving.

There are several important factors related to probabilistic models that directly affect the success of EDAs and that motivate much of the work on efficiency enhancement of EDAs:

Model Complexity The class of allowable models that an EDA can generate has to be able to efficiently represent important regularities in the problem landscape. If the EDA is not able to do so across the types of problems it is trying to solve, then the EDA will be unable to solve the problem efficiently or only be able to find low quality solutions. This was shown in Section 2.2.2 where due to the nature of the probability vector model, the simple EDA was unable to learn the correct statistics for the trap-5 function. This led to progressively worse solutions over time.

Model Quality Simply having a class of models able to represent important regularities in the problem landscape is not enough. The model building in the EDA must be able to capture the dependencies in the problem accurately and learn the correct parameters associated with these dependencies. This is important because a high quality model can lead rapidly to high quality solutions. On the other hand, low quality models can either slow down problem solving or even lead the search away from optima.

Learning Complexity Another important factor is how time consuming it is to learn the model. If the complexity of the learning is too high, the EDA could spend too much time learning the model each generation, greatly slowing down overall convergence. For example, finding the best Bayesian network model given most Bayesian and non-Bayesian metrics is NP-complete (Chickering et al., 1994). To avoid this bottleneck and efficiently learn the model, hBOA uses a greedy algorithm to determine which dependencies to add to its Bayesian network each generation.

Sampling Complexity While the complexity of learning a model is important, also of importance is the efficiency of sampling the model to generate new solutions. Some class of allowable models can be easier to learn given data, but take much longer to sample in order to generate new solutions. For example, some EDAs use Markove networks, which are similar in structure to Bayesian networks except that the connections between variables are undirected. While Markov networks have some advantages over Bayesian networks (Mühlenbein, 2008), sampling Markov networks is often more difficult and can bottleneck the performance of such algorithms.

Overfitting While learning models that represent regularities in the promising solutions is important, this must not happen at the cost of generalization. It is important that diversity is maintained after sampling to ensure that the EDA is generating solutions similar to high quality solutions but not simply repeating previous solutions. If the statistics gathered are too strongly represented in the model, it is possible for the EDA to converge prematurely to a poor solution.

All these factors together are of importance in making sure an EDA can solve a problem efficiently. For example, the simple probability vector EDA used in Section 2.2.1 has a very low learning complexity. However, its model complexity is not enough to represent the deceptive trap function in Section 2.2.2 and so it is unable to find high-quality solutions for this type of problem. On the other hand, hBOA's models are much more complex and can easily find the solution to trap functions. This greater class of allowable models allows hBOA to solve a much broader variety of problems but at the cost of a more complex model building phase, hence the importance of developing efficiency enhancements to help speed up model building.

2.6 Discussion

In this thesis the primary focus is on the efficiency enhancement of hBOA through learning from experience to speed up model building. This is important, as model building is the primary bottleneck in hBOA and most EDAs. As the previous section showed, efforts have been made to use prior knowledge garnered from previous runs to speed up EDAs but these methods were tailored to one particular problem type. What this thesis is interested in is developing principled methods to exploit prior knowledge learned from previous EDA runs that can be extended to a broad variety of problems. While the focus is on hBOA as the example EDA used, the methods described later on in this thesis are applicable to many other EDAs.

However, before we develop these methods, we must first be sure that hBOA is able to accurately learn important problem regularities. This is of primary importance, as if the models learned by hBOA are of poor quality or change rapidly over time, then it should be quite difficult to exploit this information in a general way. The next chapter examines this question by looking in depth at the accuracy and stability of hBOA models on a variety of problem types.

2.7 Summary

This chapter starts by describing the basic EDA procedure and then motivates the importance of linkage learning in GAs. hBOA is then described in detail and the importance of model building and efficiency enhancement of EDAs is discussed. A summary of the key points of this chapter follows:

- EDAs are stochastic optimization algorithms that explore the space of candidate solutions by sampling an explicit probability model built from promising solutions.
- Linkage learning is extremely important in EDAs, as without it EDAs cannot solve many problems reliably.
- hBOA is an EDA that performs linkage learning by building a Bayesian network model of promising solutions. This Bayesian network represents both independences and dependencies between different bits in the problem.
- The types of models used and the quality of the models are extremely important parts of EDA design. Often there is a tradeoff between elements of model design, as the more complex models take more time to be generated by the EDA. Yet at the same time, if the model is not complex enough, the EDA might be unable to solve the problem.

- Since EDA performance is often bottlenecked by model building speed, a great many efficiency enhancements have been developed to help speed up model building.

Chapter 3

Model Accuracy in hBOA[2]

The previous chapter discussed the central importance of EDA model building. However, relatively little work has been done to determine how accurately the models generated by an EDA represent the structure of the underlying optimization problem (Lima et al., 2006; Wu and Shapiro, 2006; Lima et al., 2007). This is crucial, as the effectiveness of any information we attempt to mine from the models of an EDA depends on the quality of these models. If the models generated by an EDA do not closely represent the underlying problem structure of the optimization problem, we cannot hope to gain much from using those models to help guide exploration in future runs of the EDA on similar problems.

The purpose of this chapter is to analyze the structure and complexity of hBOA models and compare the models hBOA generates to the so-called "perfect" model for each problem. Since the complexity and structure of hBOA models can change dramatically between problem types, four common test problems will be examined that each test a unique element of hBOA model building. Concatenated traps will be used as an example of a problem that requires a model that can separate the input string into independent chunks, since solving trap-5 efficiently requires that interactions between the entire trap partition be modeled correctly (Deb and Goldberg, 1991; Ackley, 1987). Next, NK landscapes with nearest neighbor interactions are examined to show the effect on hBOA models of different levels of difficulty and overlap between subproblems. Then, hierarchical traps are examined to see how hBOA model accuracy holds up when no single level decomposition is sufficent to solve the problem. Lastly, two-dimensional Ising spin glasses are examined as they are difficult for most optimization techniques. This is because 2D Ising spin glasses cannot be decomposed into subproblems of bounded order (Mühlenbein et al., 1999) and they have an extremely rugged fitness landscape (Barahona, 1982; Mezard et al., 1987; Fischer and Hertz, 1991; Dayal et al., 2004; Pelikan and Hartmann, 2006). Due to these difficulties, 2D Ising spin glasses should be an almost ideal final problem to test the accuracy of hBOA model building.

For each of the four types of problems examined, the analysis will focus on answering the following questions:

[2]This chapter is based in part on work previously published in Hauschild, M., Pelikan, M., Sastry, K., & Lima, C. F. (2009). Analyzing Probabilistic Models in Hierarchical BOA. *IEEE Transactions on Evolutionary Computation*, *13*(6), 1199–1217

Accuracy Do the hBOA models correspond closely with the underlying problem structure of a so-called "perfect" model of the optimization problem?

Dynamics How do the models change from generation to generation?

Commonality Do the models share regularities between different problem sizes?

The first question is important because if the model built does not strongly correspond to the underlying problem structure, even if the model is of sufficient quality to solve the problem, it is unlikely (though still possible) that we could use the information generated in the models to help guide exploration on similar problems. The second question is important because if the complexity or nature of the models change dramatically between generations, it will be difficult to gather useful statistics on the models. We explore the last question because if instances of different sizes share important commonalities, then it should be possible to exploit this information for speedups when scaling up to larger problem sizes.

This chapter starts by analyzing probabilistic models obtained when solving trap-5 in section 3.1. Nearest neighbor NK landscapes and their model quality when solved by hBOA is then examined in section 3.2. Section 3.3 discusses the problem of solving hierarchical traps, and then analyzes probabilistic models obtained with hBOA when solving hierarchical deceptive traps of order 3. Section 3.4 discusses the problem of finding ground states of 2D $\pm J$ Ising spin glasses with periodic boundary conditions, and presents the analysis of hBOA models in this class of problems. Finally, Section 3.5 summarizes the results on model quality.

3.1 hBOA Models on Trap-5

This section begins the analysis of hBOA models on trap-5 in particular, for several important reasons. First, separable problems of bounded difficulty represent a very broad class of problems (Goldberg, 2002) and many real-world problems are thought to be nearly decomposable (Simon, 1968). Second, unlike some more complicated problems like we will examine later on in this chapter, for separable problems and trap-5 in particular there is a good amount of theory that what a probabilistic model would need to be such that the EDA using such a model will scale well as problem size and difficulty increases (Goldberg, 2002; Pelikan et al., 2002b; Pelikan, 2005). Lastly, since trap-5 has fully deceptive subproblems that must be considered together to solve the problem (Deb and Goldberg, 1991), if the model fails to represent the underlying problem structure then we would expect the algorithm to fail to scale up as problem size increases (Deb and Goldberg, 1991; Thierens and Goldberg, 1993; Goldberg, 2002; Pelikan, 2005).

Answering the above questions is important to better understand how hBOA works on separable problems. But even more importantly, these results provide important information for developing theory and efficiency enhancement techniques for hBOA and nearly decomposable problems of bounded difficulty. Our primary focus will be on the second question as an in-depth analysis of model accuracy in hBOA on traps can be found in (Lima et al., 2007).

3.1.1 Concatenated 5-bit Trap

In concatenated 5-bit traps (Ackley, 1987; Deb and Goldberg, 1991), the input string is partitioned into disjoint groups of 5 bits each. This partitioning is unknown to the algorithm, and it does not change during the run. A 5-bit trap function is then applied to each of the groups and the contributions of all the groups are added together to form the fitness, which we want to maximize. The 5-bit trap function is defined as follows:

$$trap_5(u) = \begin{cases} 5 & \text{if } u = 5 \\ 4 - u & \text{otherwise} \end{cases}, \tag{3.1}$$

where u is the number of ones in the input string of 5 bits. An n-bit trap-5 has one global optimum in the string of all ones and $2^{n/5} - 1$ other local optima (the number of local optima grows exponentially fast with problem size).

Trap-5 necessitates that all bits in each trap partition (linkage group) be treated together, because statistics of lower order mislead the search away from the global optimum (Deb and Goldberg, 1991), leading to highly inefficient performance (Thierens and Goldberg, 1993; Goldberg, 2002).

3.1.2 A Perfect Model for Trap-5

The first question to be asked when attempting to determine model accuracy for any type of problem is, what is the "perfect" model for solving the problem. That is, in an ideal world with a perfect probabilistic model generator, what would we expect the model to look like? This is often not a trivial problem at all and for many problems (as we will see later on in this chapter) it does not have a clear answer. However, the reason we start with trap-5 in this chapter is that this answer is relatively straightforward. Since trap-5 is fully deceptive (Deb and Goldberg, 1991), it is necessary that the probabilistic model contains dependencies between nearly all pairs of variables within each trap partition. Also, since the variables in each trap partition are independent, it is best if no variables in different trap partitions have any dependencies between them. Given these constraints, the "perfect" model of trap-5 in hBOA should contain dependencies between all bits in each trap partition and no dependencies between bits in other trap partitions. An example of this is

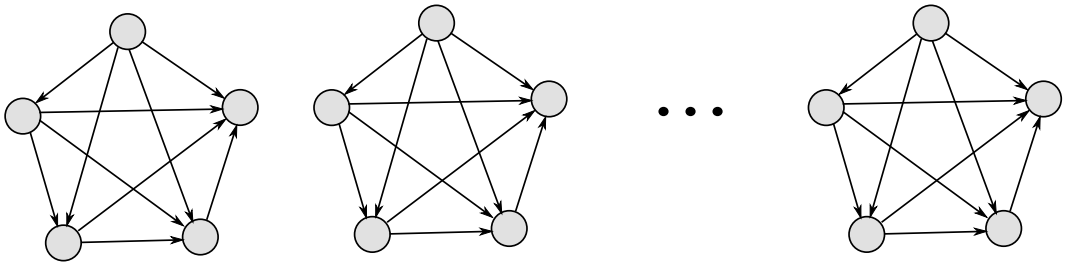


Figure 3.1: A perfect model for solving trap-5. Each trap partition is fully connected and there are no dependencies between different trap partitions.

shown in Figure 3.1.

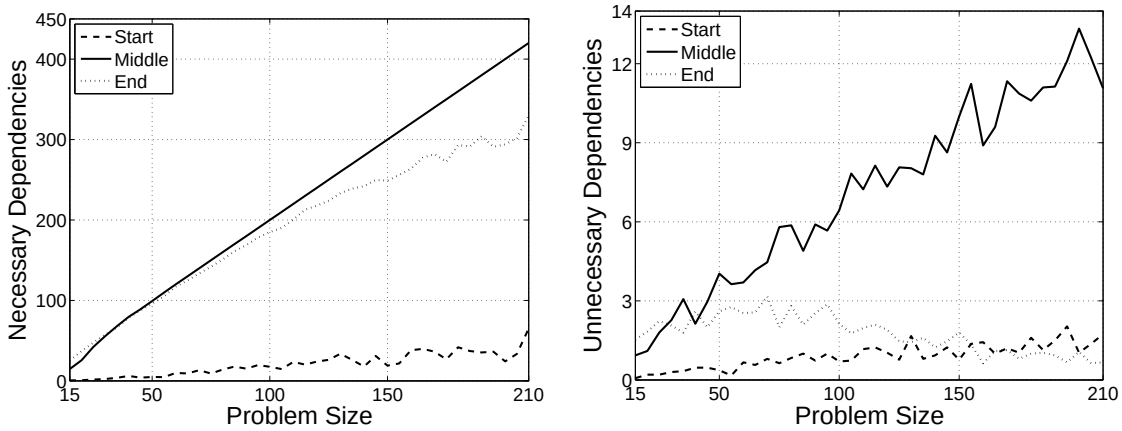
In the remainder of this section, we call the dependencies that we would like to discover the *necessary* dependencies whereas the dependencies that we would like to avoid in order to maximize the mixing will be called *unnecessary*. Note that in some cases it is not necessary for the model to have all of the *necessary* dependencies.

3.1.3 Experimental Setup

To ensure that our results are not biased by an incorrectly chosen population size, bisection (Sastry, 2001b) is used to ensure that the population size is large enough to obtain reliable convergence to the global optimum in 30 out of 30 independent runs. The results are then averaged over 30 successful runs. The number of generations is upper bounded according to preliminary experiments and hBOA scalability theory (Pelikan et al., 2002b) by $2\sqrt{n}$ where n is the number of bits in the problem. Each run of hBOA is terminated when the global optimum has been found (success) or when the upper bound on the number of generations has been reached without discovering the global optimum (failure). In order to examine the effects that problem size has on the probabilistic models, problem sizes of $n = 15$ to $n = 210$ bits are considered. Truncation selection with $\tau = 50\%$ is used, which selects the 50% best solutions in the population to generate the model each generation.

3.1.4 Model Accuracy for Trap-5

To start the analysis of accuracy of the probabilistic models in hBOA on trap-5, we need to examine how many of the necessary and unnecessary dependecies are discovered by hBOA during model building. Numerous emperical results have shown that hBOA can solve trap-5 with evaluations that grow polynomially with problem size (Pelikan et al., 1999; Pelikan, 2005), it should be expected that hBOA will discover most of the necessary dependencies. If hBOA did not discover enough of the necessary dependencies, then it



(a) Necessary dependencies

(b) Unnecessary dependencies

Figure 3.2: The average number of necessary and unnecessary dependencies with respect to problem size on trap-5. Three model snapshots were taken in each run (the start of the run, the middle generation, and the final generation).

should suffer from exponential scaling as problem size increases (Thierens and Goldberg, 1993; Goldberg, 2002; Pelikan, 2005). The scalability of hBOA on trap-5 is much less reliant on the number of unnecessary dependencies, as it has been shown that if tournament selection is used, hBOA often finds many unnecessary dependencies (Lima et al., 2006). However, in this chapter we use truncation selection and the results show a very small number of unnecessary dependencies with respect to the overall number of dependencies. In the remainder of this section we will see that with truncation selection the models show trap partitions that are nearly fully connected while only a relatively few dependencies are between bits in other partitions.

Figure 3.2 shows the number of necessary and unnecessary dependencies discovered when solving trap-5 of sizes 15 to 210. Since different models are discovered each generation and each particular run can last a varying number of generations, specific models were selected from each run corresponding to the model generated during the first generation, the model generated during the middle of the run and the last model generated before the run was terminated. The results show that during the first generation, only a small percentage of the necessary dependencies are discovered. However, by the middle of the run, almost all necessary dependencies are discovered, with the number of necessary dependencies being equal to twice the problem size. For example, with a problem size of $n = 100$, we see that on average 200 necessary dependencies were discovered during the middle snapshot. At the end of the run, the model still covers many but not quite all of the necessary dependencies.

Figure 3.2b are what we would expect in regards to necessary dependencies. Since the population is initialized at random and the first model is built after only one selection step, hBOA will only have enough statistical information to identify some of the subproblems. As the problem progresses, statistical noise

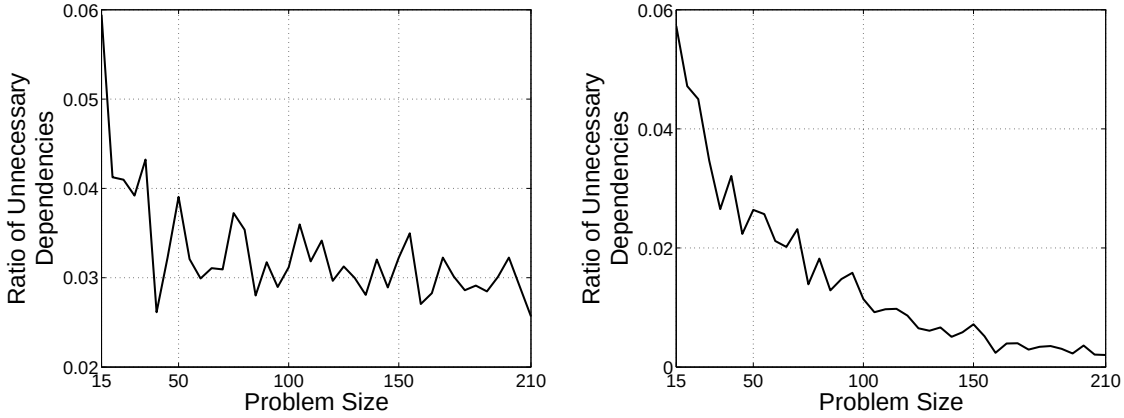
decreases in the population (Goldberg, 2002) as the overall population fitness increases and hBOA is able to find enough information to build more accurate models. At the end of the run, as the population starts to converge to the optimum, some of the subproblems have become simple enough to be represented by short-order conditional probabilities. This is because near the end of a run of hBOA on trap-5, most of the trap partitions are assigned to one of the two best local optimas, either 00000 or 11111.

This simplification of the necessary dependencies near the end of an hBOA run on trap-5 can be seen in a simple example. Consider a population of 5-bit binary strings that have all converged to either 00000 or 11111, which is similar to the end of an hBOA run on trap-5. Since the value of any bit depends on the value of any of the remaining bits, it can be easily encoded by a simple chain model probability distribution defined as To illustrate the argument that in the end of the run the models can significantly simplify without affecting the encoded distribution, we look at a simple example. Consider a population of 5-bit binary strings with only two alternative candidate solutions: 00000 and 11111. Late in the run of hBOA on trap-5, every trap partition is expected to have the same or at least a very similar distribution to this. Clearly, the value of any bit depends on the value of the remaining bits. Nonetheless, to fully encode this probability distribution, it is possible to use a simple chain model defined as

$$p(X_1, X_2, X_3, X_4, X_5) = p(X_1)p(X_2|X_1)p(X_3|X_2)p(X_4|X_3)p(X_5|X_4).$$

Therefore, as the population diversity decreases and some partial solutions are eliminated, many of the necessary dependencies become unnecessary.

In the previous paragraphs of this section we only discussed the necessary dependencies. Figure 3.2 shows the number of unnecessary dependencies discovered during the three snapshots taken during hBOA model building. The results show that the number of unnecessary dependencies discovered is quite small in relation to the total number of dependencies discovered. This is made even clearer in figure 3.3, which shows that the ratio of the number of unnecessary dependencies to the number of necessary dependencies decreases with problem size. For example, for $n = 100$, only 7 unnecessary dependencies are found but nearly 200 necessary ones. These results clearly show that hBOA is not only capable of discovering the necessary dependencies to solve trap-5 scalably, but that it is not overwhelmed by the discovering of too many unnecessary dependencies which could slow down the mixing. Note though that this is only the case with truncation selection. It has been shown (Lima et al., 2006) that if tournament selection is used that the number of unnecessary dependencies can become relatively significant. However, it is possible to alter the metric score of hBOA when using tournament selection to increase model accuracy (Lima et al., 2008)



(a) Middle of the run.

(b) End of the run.

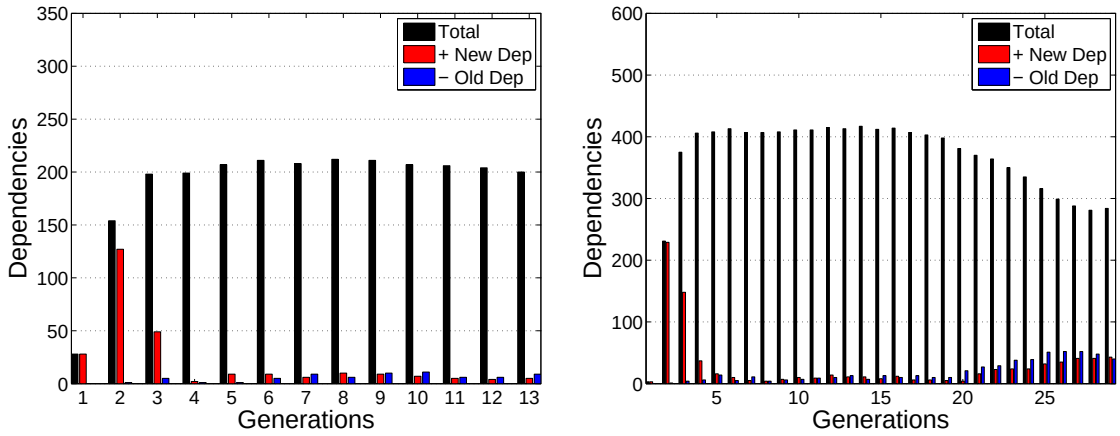
Figure 3.3: Ratio of the number of unnecessary dependencies versus the total number of dependencies by problem size for the middle generation and the final generation for trap-5 of sizes $n = 50$ to $n = 210$.

One of the most important results in this chapter is that the model structure in hBOA signficantly depends on the selection method used to select the promising solutions from the population. These results were also seen in work on hybridization of HBOA (Lima et al., 2006), as well as work on the accuracy of hBOA models with tournament selection (Lima et al., 2007). In (Lima et al., 2007) it was shown that while hBOA with tournament selection is able to find most necessary dependencies early in the run, the number of unnecessary dependencies discovered was quite significant. This seems to indicate that hBOA with tournament selection builds an overly complex model for trap-5. On the other hand, the results shown in figures 3.2 and 3.3 strongly show that with truncation selection the number of unnecessary dependencies discovered is quite small. However, since hBOA scalability is asymptotically the same for both selection methods (Pelikan et al., 1999; Pelikan, 2005), more work must be done in studying the influence of selection methods on the performance of EDAs and model building.

3.1.5 Model Dynamics for Trap-5

The previous section showed that for trap-5, hBOA models were quite accurate. However, another question is how do the models change over time? Are the models in each generation relatively similar to models in nearby generations? When is it that most of the model changes take place? Do the models become relatively stable early on or do they continually change up until convergence? These are important questions as in order to efficiently mine hBOA models for information to improve speedups, we must know how many models we will need for a valid sample. If the models change dramatically between generations, this could make it quite difficult to gather useful statistics from models.

We begin the analysis of model dynamics in hBOA by analyzing the changes in hBOA models in sub-

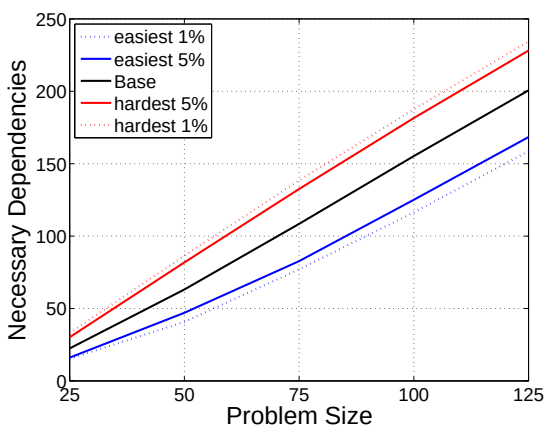


(a) $n = 100$ bits. (b) $n = 200$ bits.

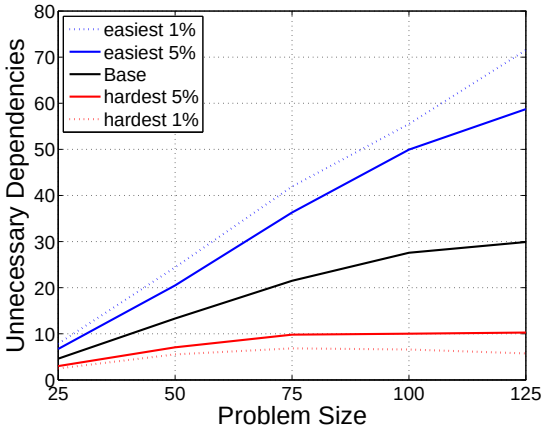
Figure 3.4: The number of dependencies that change by generation for a specific run of hBOA on trap-5. The first bar (total) represents the total number of dependencies discovered during that generation. The second bar represents the number of dependencies discovered that were not discovered during the previous generation. The final bar represents those dependencies that were discovered in the previous generation but were not discovered in the current generation.

sequent generations in hBOA. For each generation we first record the number of dependencies that were not present in the previous generation but were discovered in the current generation, as well as record the number of dependencies that were discovered in the previous generation but were not used in the current one. Figure 3.4 shows the results for two problem sizes of trap-5, $n = 100$ and $n = 200$. The results strongly show that the primary dynamic of model change occurs in the first few generations. Once a good model is discovered, then relatively little change occurs until the population starts to converge to a solution. While figure 3.4 only shows two problem sizes, the results for other problem sizes were very similar.

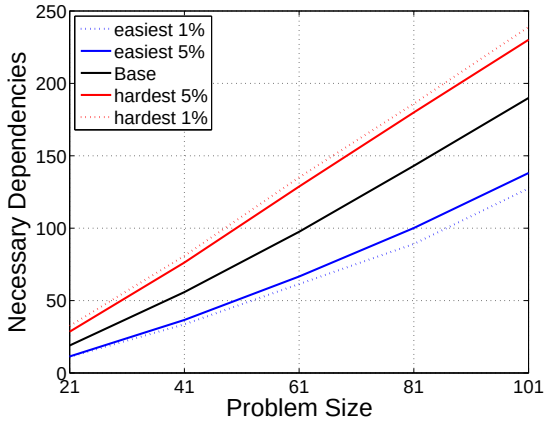
The stability of hBOA models on trap-5 is great news for our goal of mining the models for information on how to speed up hBOA model building in the future. It is also good news for many efficiency enhancements like sporadic and incremental model building (Pelikan et al., 2006d), which focus on improving efficiency of model building in hBOA. More specifically, both sporadic and incremental model building lead to the highest gains in performance when models in subsequent generations have similar structure. Our results indicate that this is indeed the case, at least for separable problems of bounded difficulty. In addition, the rapid learning of an accurate model is good news for efficient hBOA hybrids that use specialized local search operators based on the models generated (Lima et al., 2006; Sastry and Goldberg, 2004).



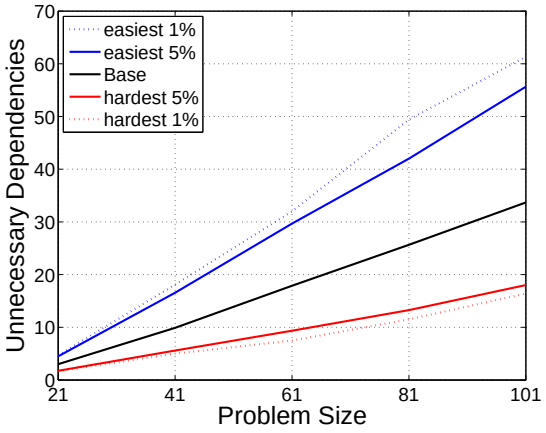
(a) Necessary dependencies, o=0



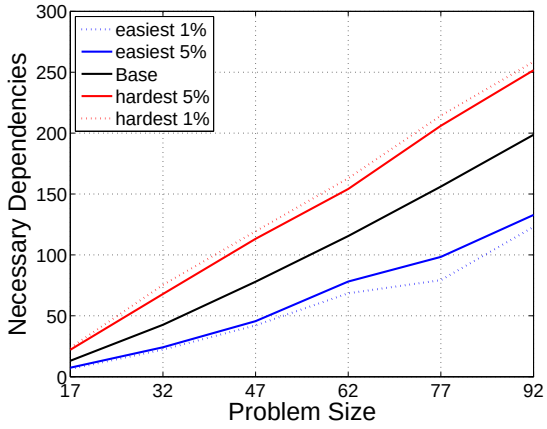
(b) Unnecessary dependencies, o=0



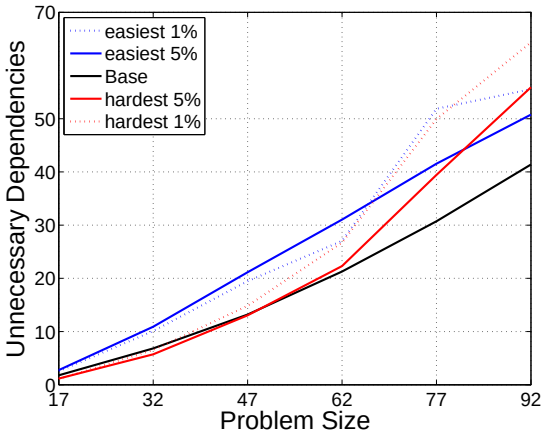
(c) Necessary dependencies, o=1



(d) Unnecessary dependencies, o=1

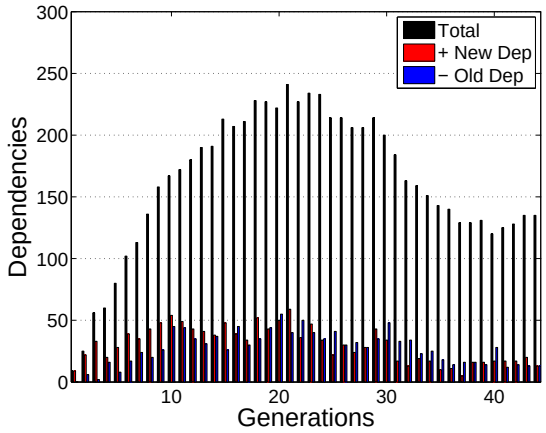


(e) Necessary dependencies, o=2

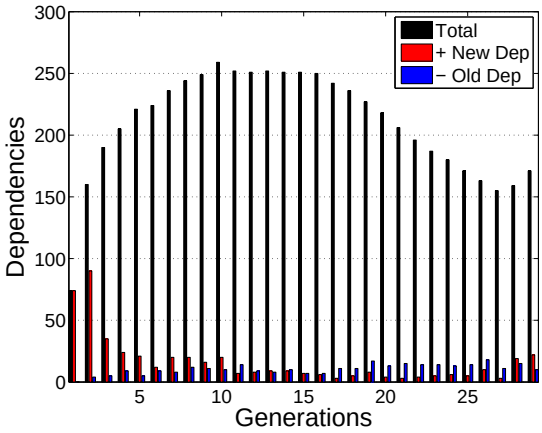


(f) Unnecessary dependencies, o=2

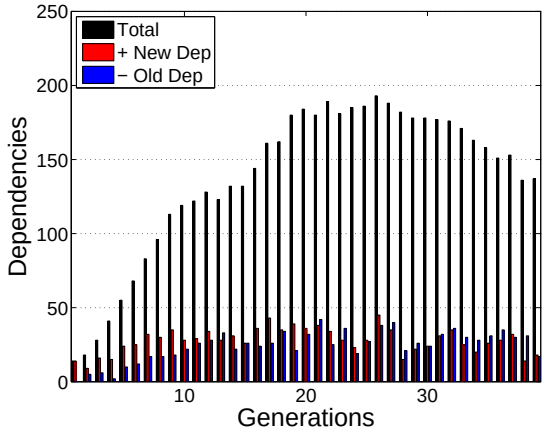
Figure 3.5: Average number of necessary and unnecessary dependencies versus problem size for nearest neighbor NK landscapes with different overlaps during the middle generation. Each line represents a different set of instances ranked by difficulty, with difficulty determined by the number of evaluations required to solve the problem.



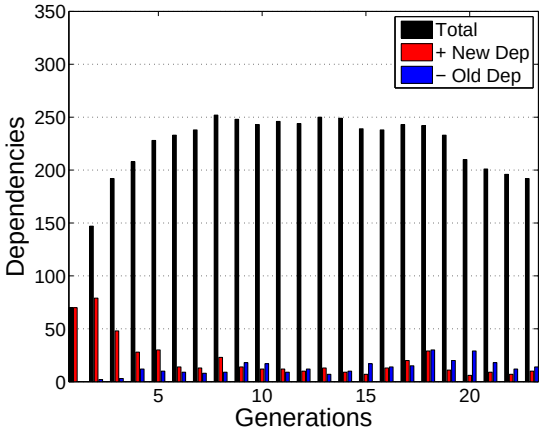
(a) 5% easiest, n=125, o=0



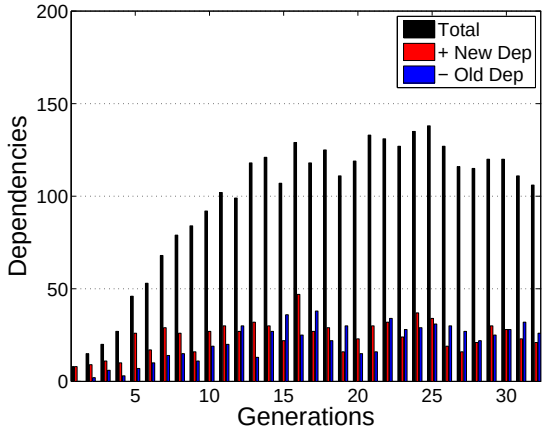
(b) 5% hardest, n=125, o=0



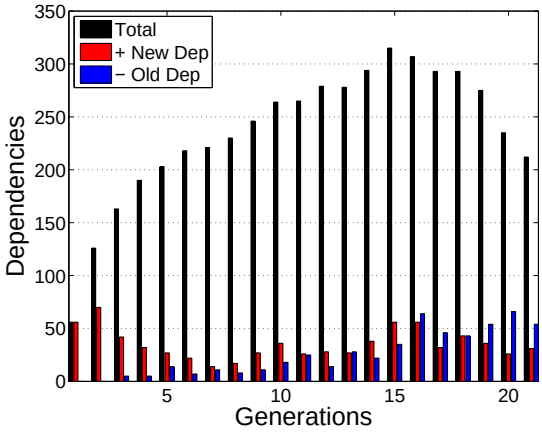
(c) 5% easiest, n=101, o=1



(d) 5% hardest, n=101, o=1

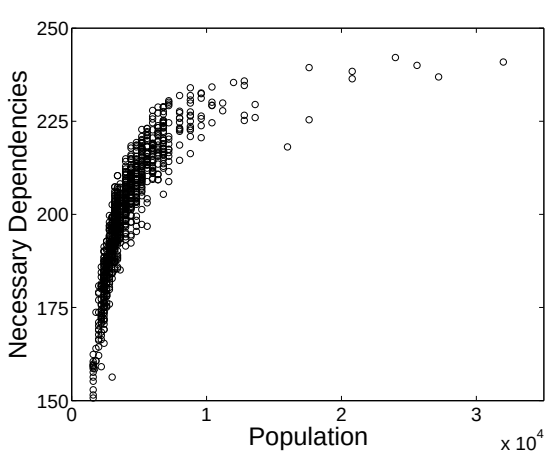


(e) 5% easiest, n=77, o=2

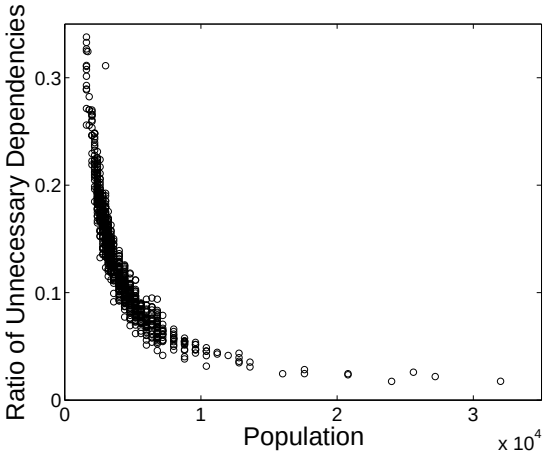


(f) 5% hardest, n=77, o=2

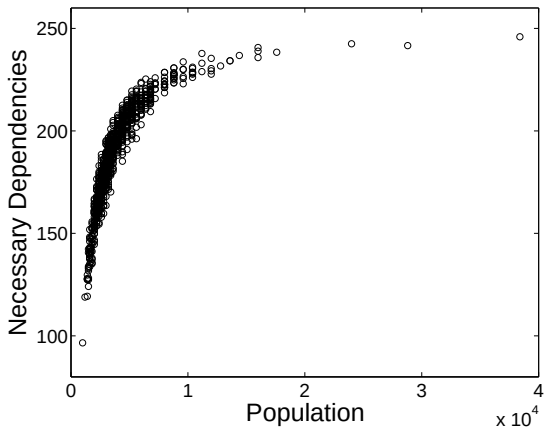
Figure 3.6: The number of dependencies that change by generation for specific runs of hBOA on nearest neighbor NK landscapes. The first bar (total) represents the total number of dependencies discovered during that generation. The second bar represents the number of dependencies discovered that were not discovered during the previous generation. The final bar represents those dependencies that were discovered in the previous generation but were not discovered in the current generation.



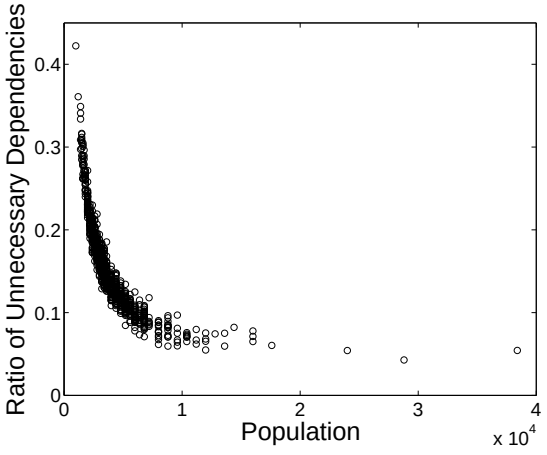
(a) Necessary dependencies



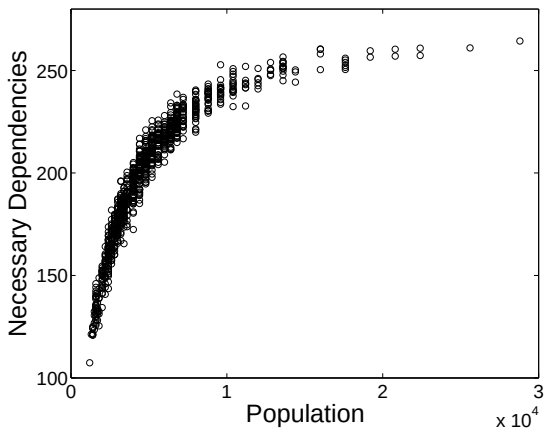
(b) Ratio of Unnecessary dependencies



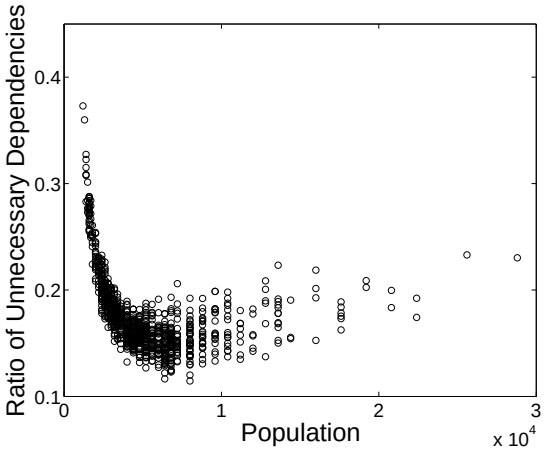
(c) Necessary dependencies



(d) Ratio of Unnecessary dependencies



(e) Necessary dependencies



(f) Ratio of Unnecessary dependencies

Figure 3.7: Average number of necessary dependencies and the ratio of unnecessary dependencies with overall dependencies found in the middle generation versus the population required to solve the instance on nearest neighbor NK landscapes.

3.2 hBoa Models on Nearest Neighbor NK Landscapes

The previous section showed that hBOA was able to learn an adequate model quickly for trap-5. However, trap-5 has subproblems that do not interact and each subproblem is of the same difficulty. What happens to model accuracy and dynamics when hBOA is presented with a problem where the subproblems overlap and each subproblem can vary in difficulty? NK landscapes (Kauffman, 1989; Kauffman, 1993) are an artificial test problem where it is possible to vary the amount of overlap, problem difficulty and the size of the subproblems. In this section we will explore how hBOA divides up the problem and how the models change over time.

3.2.1 NK Landscapes

An NK fitness landscape (Kauffman, 1989; Kauffman, 1993) is fully defined by the following components: (1) The number of bits, n, (2) the number of neighbors per bit, k, (3) a set of k neighbors $\prod(X_i)$ for the i-th bit, X_i for every $i \in \{0, \ldots, n-1\}$, and (4) a subfunction f_i defining a real value for each combination of values of X_i and $\prod(X_i)$ for every $i \in \{0, \ldots, n-1\}$. Typically, each subfunction is defined as a lookup table. The objective function f_{nk} to maximize is defined as

$$f_{nk}(X_0, \ldots, X_{n-1}) = \sum_{i=0}^{n-1} f_i(X_i, \prod(X_i)) \tag{3.2}$$

The difficulty of optimizing NK landscapes depends on all components defining an NK problem landscape For $k > 1$, the problem of finding the global optimum of unrestricted NK landscapes is NP-complete (Wright et al., 2000). The problem becomes polynomially solvable with dynamic programming even for $k > 1$ if the neighbors are restricted to only adjacent string positions (Wright et al., 2000) or if the subfunctions are generated according to some distributions (Gao and Culberson, 2002). For unrestricted NK landscapes with $k > 1$, a polynomial-time approximation algorithm exists with the approximation threshold $1 - 1/2^{k+1}$ (Wright et al., 2000).

In this chapter we consider nearest neighbor NK landscapes. In nearest neighbor NK landscapes, we have the following two restrictions:

1. Bits are arranged in a circle and the neighbors of each bit are restricted to the k bits that follow this bit on the circle. This restriction to nearest neighbors ensures that even those instances of $k > 1$ can be solved in polynomial time using dynamic programming.

2. Some subproblems may be excluded to provide a mechanism for tuning the size of the overlap between

subsequent subproblems. Specifically, the fitness is defined as

$$f_{nk}(X_0, X_1, \ldots, X_{n-1}) = \sum_{i=0}^{\lfloor \frac{n-1}{step} \rfloor} f_i(X_{i \times step}, \prod(X_i)) \tag{3.3}$$

where $step \in \{1, 2, \ldots, k+1\}$ is a parameter denoting the step with which the basis bits are selected. The amount of overlap between consequent subproblems can be reduced by increasing the value of *step*.

To make the instances more challenging, string positions in each instance are *shuffled* by reording string positions according to a randomly generated permutation using the uniform distribution over all permutations.

The dynamic programming algorithm used to solve the nearest neighbor class of NK landscape instances is based on refs. (Pelikan et al., 2006b; Pelikan et al., 2009).

In this section we consider nearest neighbor NK landscapes of order $k = 4$ and with overlap $o = 0, 1$, and 2, so nearest neighbor NK landscapes with zero overlap, 1 bit overlap and 2 bit overlap. All instances are generated using the implementation described in (Pelikan et al., 2006a).

3.2.2 Perfect Model for Nearest Neighbor NK Landscapes

The perfect model for nearest neighbor NK landscapes is not as simple to describe as that for trap-5. However, we would expect that much like trap-5, bits in the same partition should gain dependencies. Since the difficulty of each partition can vary, we would expect that the amount of interconnectivity between bits in a partition would vary. It is possible for a subproblem of NK landscapes to be fully deceptive, much like trap-5, and in this case we would expect that all the bits in the subproblem would need to be interconnected (Mühlenbein and Mahnig, 1999). However, it is also possible for a subproblem to have bits whose contribution is nearly independent, which would mean that none of the bits in the subproblem need to have dependencies between them. So, for the most difficult subproblems where each bit depends on all other bits, we would expect dependencies between nearly all the bits. For the simplest subproblems we would expect much less dependencies. Also, since the subproblems can overlap by varying degrees, we should see some dependencies between subproblems.

As in the previous section on trap-5, dependencies between bits in each subproblem will be called *necessary* and the remaining dependencies *unnecessary*. However, for easier subproblems many of these necessary dependencies will not truly be necessary and we would also expect some dependencies between subproblems if the subproblems overlap. Thus, for nearest neighbor NK landscapes the sharp distinction between

necessary and unnecessary dependencies is not as clear as with trap-5.

3.2.3 Experimental setup

In this section we consider nearest neighbor NK landscapes with $k = 4$ and $o = 0, 1,$ and 2. In order to examine how the models change as problem size increases, for each overlap problems containing from 5 to 25 subproblems were examined. Since NK landscapes vary in difficulty, the experiments were performed on 1000 different instances for each problem size and overlap (for a total of 15,000 instances), with all problem instances generated by (Pelikan et al., 2006a).Truncation selection with $\tau = 50\%$ was used to select the promising solutions each generation. The population size for any given instance was determined by bisection, with each run required to converge in 10 out of 10 independent runs.

3.2.4 Model Accuracy for Nearest Neighbor NK Landscapes

We first examine the model accuracy of hBOA when solving instance of NK landscapes by looking at problems of overlap 0. With overlap 0, the necessary dependencies are much clearer as we hope to have no dependencies between subproblems. Due to the number of results, we only look at the middle snapshot of the model, taken during the middle of the run. Figures 3.5a and 3.5b show the number of necessary and unnecessary dependencies for NK landscapes of sizes 25 to 125 with no overlap. Unlike trap-5, NK landscape instances vary greatly in difficulty. To examine how model accuracy changes with respect to problem difficulty, we look at the easiest and hardest 1% and 5% of instances, with the ranking determined by the number of evaluations required to solve the problem. For each problem size and difficulty, the results show the average number of necessary and unnecessary dependencies over the 10 independent runs.

It is expected that for the hardest instances, most of the necessary dependencies would be required as these problems require a greater connectivity between bits in the subproblems. The results from Figure 3.5a show that this is indeed the case. However, unlike in trap-5, we do not see all the necessary dependencies, since the maximum number of necessary dependencies is again twice the problem size. However, this is not surprising, as even in the hardest NK landscapes some of the subproblems will still be easier than others and so some subproblems will sequentially converge (Thierens et al., 1998).

Figure 3.5b shows the number of unnecessary dependencies as problem size increases for NK landscapes with 0 overlap. Much as in trap-5, the results show that the number of unnecessary dependencies is quite small in relation to the number of necessary dependencies. The results also show that for the harder problems, the number of unnecessary dependencies is actually less than for the harder problems, indicating that the ratio of necessary dependencies to unnecessary dependencies actually increases as the problem becomes

harder.

So far we have only looked at models of problems with the subproblems did not overlap. Figure 3.5c and Figure 3.5d examine problems of overlap 1 and show the necessary and unnecessary dependencies for problem sizes $n = 21$ to $n = 101$. The results show a similar pattern as those with no overlap. The most necessary dependencies are discovered in the hardest subproblems. However, in this case we do not see the increase in the ratio of necessary to unnecessary dependencies. This is because the overlap between subproblems adds interactions between bits in the different subproblems and the strongest interactions get reflected in the model even though they are counted as unnecessary dependencies. However, even in the case of overlap the overall number of unecessary dependencies is quite small in relation to the total number of dependencies.

In the final case we consider problems of overlap 2. In these problems almost half the subproblems overlap each other and so we can expect much stronger interaction between subproblems. Figure 3.5e and Figure 3.5f show the number of necessary and unnecessary dependencies for problem sizes 17 through 77. Again we see that hBOA finds most of the necessary dependencies. With the unnecessary dependencies, while they are still relatively small in comparison to the number of necessary dependencies, the ratio of unnecessary to necessary dependencies seems to increase as problem size increases. This is to be expected, as the more overlap the more likely the population is to show statistical dependencies between bits in different subproblems. While these dependencies are usually weaker than those between bits in the same subproblem, they can still be strong enough to be required by the model.

Due to the difference in difficulty between NK landscape instances, often the population varied significantly between different instances of the same population size. We next examine the change in hBOA behavior depending on the different population sizes found by bisection in Figure 3.7. Figure 3.7a shows the number of necessary dependencies found during the middle snapshot with respect to the population size for all instances of NK Landscapes with the largest problem size for all overlaps, with figure 3.7b showing the ratio of unnecessary dependencies found during the middle snapshot with respect to population size. The results from these figures show that as the population size gets larger, more necessary dependencies are discovered. However, the number of unnecessary dependencies decreases in proportion as the population increases. It is also easy to see that for the easiest problems, very few necessary dependencies were requiresd.

3.2.5 Model Dynamics for Nearest Neighbor NK Landscapes

The next question is how stable are the models and dependencies over time? This is especially important for NK landscapes since as discussed earlier the subproblems vary in difficulty and so some subproblems would

be expected to converge faster than others. Much as in trap-5, we examine this by looking at the number of dependencies that are changed and remain the same in each generation. Since all the instances vary, we selected 6 sample runs, with each one selected from a different difficulty rating and overlap, and show the results in Figure 3.6. We only show 6 sample runs but other runs of the same type showed similar results and should thus be indicative of the general model dynamics of hBOA when solving nearest neighbor NK landscapes.

The results in figure 3.6 show that for the harder instances, hBOA adds almost all of the necessary dependencies in the first few generations of the run. After that, the model is relatively stable till near the end of the run when the necessary dependencies starts to decrease. This is most likely due to some of the subproblems converging and therefore being able to be modeld with short-order probabilities. On the other hand, for the easier problems, hBOA seems to add necessary dependencies over time. However, by the middle of the run, the models are relatively stable, with most dependencies the same between generations. So even in these cases, hBOA finds a stable model relatively quickly.

3.2.6 Commonality in Nearest Neighbor NK Landscapes instances

In the previous sections we have shown that hBOA models accurately capture many important characteristics of nearest-neighbor NK landscapes. In addition, it was shown that the models quickly learn most of the important dependencies and after this is done the models are relatively stable over time. However, another question is do the models generated from different sizes of nearest-neighbor NK landscapes share similarities? If this the case, it is further proof that hBOA models capture important regularities in nearest-neighbor NK landscape instances.

To examine this, we first want to consider the *distance* between variables in a nearest-neighbor NK landscape problem. We define any two variables that are in the same subproblem as being of distance 1 from each other. This information can then be used to construct a graph G that represents the distance between any two variables in the instance, with that distance represented by the shortest path between these variables in G. It is expected that variables in the same subproblem would depend strongly on each other and variables that are far apart in distance would interact less strongly.

To examine whether the level of interaction between variables by distance is similar as problem size increases, we examined the models generated from different sized instances when solving nearest-neighbor NK landscapes. Statistics were then gathered on the probability of a split between variables by distance. Figure 3.8 shows the proportion of splits between variables for nearest-neighbor NK landscapes of size $n = 50$ and $n = 100$ with $k = 4$ and $o = 4$ using truncation selection. 1000 instances of each size was used, with 10

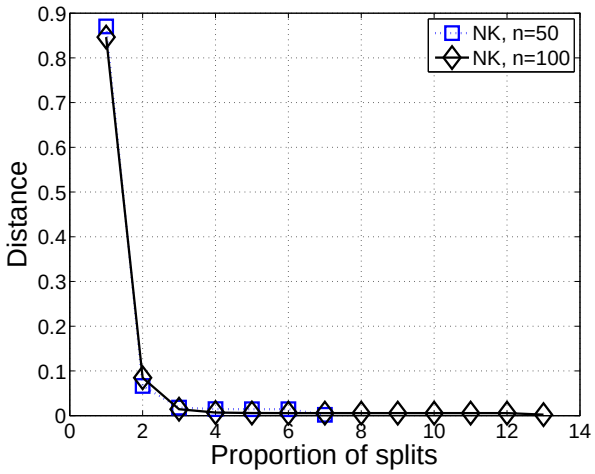


Figure 3.8: The proportion of splits between variables of different distances on nearest-neighbor NK landscapes of sizes $n = 50$ and $n = 100$ with $k = 5$ and $o = 4$.

runs for each instance.

The results show that the likelyhood that variables will have a split between them in the decision graph is strongly dependent on distance and that the proportion of total splits by distance remains almost identical as problem size increases. While this is further proof that hBOA model building gathers important information about problem structure, it also means that it should be possible to use statistics gathered on smaller problem sizes to help speed up problem solving of much larger problems.

3.3 hBOA Models on Hierarchical Traps of Order 3

The previous two problems considered, trap-5 and rADPs, both could be solved by a fixed decomposition at a single level. That is, the interaction between subproblems did not change as the problem progressed. However, not all difficult problems can be decomposed in this way and in some problems it is necessary to use hierarchical decomposition (Simon, 1968; Goldberg, 2002; Pelikan, 2005). In these types of problems it is necessary to consider solutions over multiple levels of difficulty in order to find a solution. When solving hierarchical problems, it is necessary for the algorithm to start at the bottom level with subproblems of small order and at each level of optimization the algorithm must use subproblems of the lower level as the basic building block to solve solutions at the current level.

It has been shown empirically that hBOA can solve difficult hierarchical problems scalably and reliably (Pelikan, 2005; Pelikan et al., 2002b; Goldberg, 2002). However, just as with the single level decomposition problems of trap-5 and rADP, do the models generated by hBOA when solving hierarchical problems match the expected structure of the problem and how do they look over time? In this section we will ex-

amine hBOA models on hierarchical traps. Similarily to how trap-5 bounds the class of separable problems, hierarchical traps bound the class of hierarchical problems. They are deceptive at all levels and are also highly multimodel (Goldberg, 2002; Pelikan, 2005). In this section we will only examine model dynamics for hTraps as the distinction between necessary and unnecessary dependencies is clear in hTrap.

3.3.1 Hierarchical Traps of Order 3

Hierarchical traps (hTraps) (Pelikan and Goldberg, 2001; Pelikan, 2005) of order 3 are created by combining trap functions of order 3 over multiple levels of difficulty. In the variant of hTrap used in this work, on the lowest level, groups of 3 bits contribute to the overall fitness using a generalized 3-bit trap

$$trap_3(u) = \begin{cases} f_{high} & \text{if } u = 3 \\ f_{low} - u(f_{low}/2) & \text{otherwise} \end{cases}, \tag{3.4}$$

where $f_{high} = 1$ and $f_{low} = 1 + 0.1/l$. Note that for these values of f_{high} and f_{low} the optimum of the trap is 000.

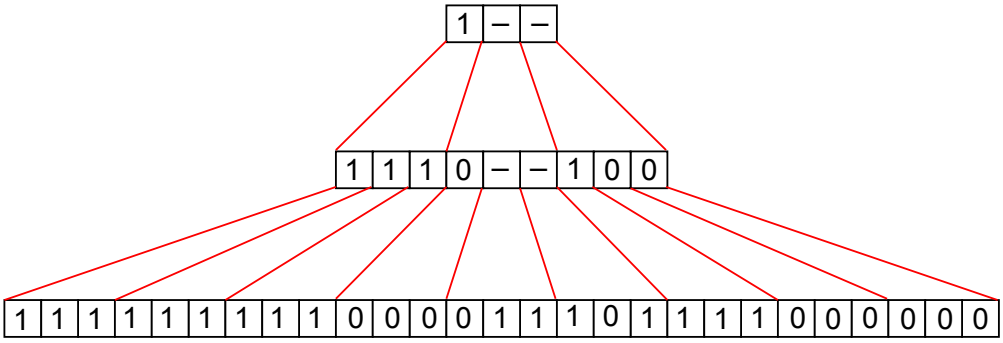


Figure 3.9: Sample mapping function for hTrap of order 3 with 3 levels. A 000 is mapped to a 0, a 111 is mapped to a 1, and everything else is mapped to the null symbol '-'.

Each group of 3 bits corresponding to one of the traps is then mapped to a single symbol 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 '-'. See figure 3.9 for an example of this mapping function. The bits on the next level again contribute to the overall fitness using 3-bit traps defined above, and the groups are mapped to an even higher level. This continues until the top level is evaluated that contains 3 bits total. However, on the top level, a trap with $f_{high} = 1$ and $f_{low} = 0.9$ is applied. As a result, the optimum of hTrap is in the string of all ones despite the superior performance of blocks of zeros on any level except for the top one. Note that any group of bits containing the null symbol do not contribute to the overall fitness.

To ensure that the total contribution from each level is of the same magnitude, the contributions of traps

on the lth level from the bottom are multiplied by 3^l.

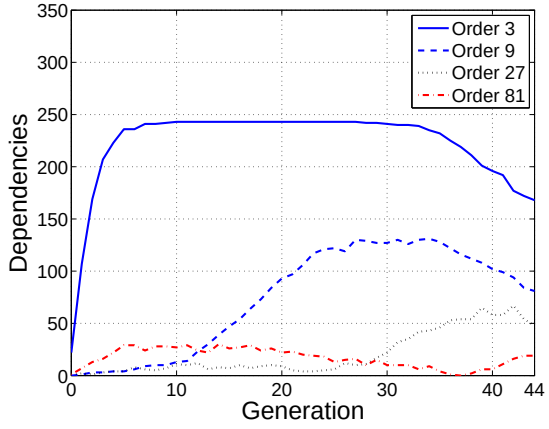
While hTrap has many local optima, the global optimum is the string all 1s. The reason that hTrap is difficult for most optimizers is that any single-level decomposition into subproblems of bounded order will lead away from this global optimum. So hTrap requires an optimizer that can build hierarchically on good partial solutions over multiple levels of difficulty until the global optimum is found.

3.3.2 Perfect Model for Hierarchical Traps of Order 3

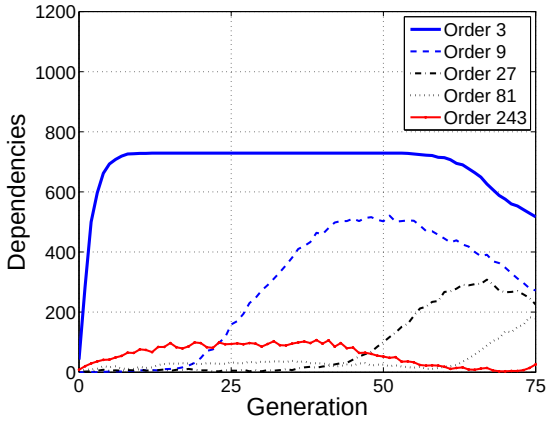
The question of what a perfect model is to solve hTrap requires a bit more thought than for rADP and trap-5. rADPs and trap-5 can both be solved by the algorithm developing a single fixed decomposition that divides the problem up, and indeed our results from the previous sections shows that hBOA is able to find this decomposition quickly. However, for hTrap a single fixed decomposition of bounded order is not sufficient. Any fixed decomposition of bounded order will instead mislead hBOA away from the optimum and will result in exponential scaling. Instead, for hBOA to solve hTrap successfully, it must proceed level by level, finding the optima of the subproblems at each level and then proceeding to the next level and using those partial solutions to solve that level.

In particular, hBOA needs to start by first finding the local optima of the 3-bit trap at the lowest level, so the lowest level should show subsets of bits corresponding to either 000 or 111. Once these are mostly solved, then hBOA can proceed by solving the next level, which should give blocks of nine 0s or nine 1s. Then once this level is complete, hBOA sets about solving the next level and so forth, with hBOA composing solutions of higher and higher order with blocks of 0s and 1s. An important thing to remember is that once a level is "solved" the algorithm cannot simplify it right away to a simple string of conditional probabilities, as it might need to mix the lower levels to gain higher level fitness.

Now that we know how hBOA would need to go about to solve the problem, we can consider what the perfect model for a hTrap would be. To do this, we will examine the dependencies between bits in each subproblem on all different levels of the hierarchical trap. Dependencies between bits on the lowest level, that is those that connect bits within a 3-bit subproblem, will be referred to as dependencies of order 3. Dependencies of the next level will be referred to as dependencies of order 9 and so forth, however dependencies of lower order subproblems will be excluded from higher order subproblems, so for example order 3 dependencies will be excluded from order 9 and up dependencies. Continuing on up higher levels, he dependencies of order 3^l for any level l will denote the dependencies between the 3^l bits of the subproblems on the lth level from the bottom, excluding dependencies of order $3^1, 3^2, \ldots, 3^{l-1}$. For example, in a string of 243 bits, a dependency between bits 0 and 1 would be considered of order 3, as they connect bits in



(a) n=243 bits



(b) n=729 bits

Figure 3.10: The number of dependencies by their order discovered during each generation of a particular run of hTrap of sizes $n = 243$ and $n = 729$.

the first level partion 0-2. A dependency between bits 0 and 50 would be of order 81, as it would connect subpartitions 0-26 and 27-53 of the 3rd level decomposition.

As hBOA solves an hTrap, it should start by discovering dependencies of order 3. Next, it should discover dependencies of order 9. In this manner, it should discover dependencies of higher and higher order until the global optimum is found and the run is terminated.

3.3.3 Experimental setup

Population size was determined by bisection, with hBOA required to converge in 30 out of 30 independent runs. In order to look at the probabilistic models of hBOA on hTrap as the problem size increased, we looked at problems of sizes $n = 243$ and $n = 729$. Truncation selection was used with $\tau = 50\%$.

3.3.4 Model Dynamics for Hierarchical Traps

Figure 3.10 shows the number of dependencies of different orders by generation in two sample runs of hBOA. While only two runs are shown, the other runs showed similar results and these runs should be indicitive of the general pattern hBOA exhibits when solving hTrap. The results show that in the first few generations hBOA quickly finds almost all the dependencies of order 3, similarly to how hBOA found the necessary dependencies quickly in trap-5. Also of note is that initially very few dependencies of higher order are found.

Once hBOA has found enough of the dependencies of the lowest order of 3, it must start to encode dependencies of order 9 if it hopes to solve the problem. The results show that at generation 10 for $n = 243$ and for generation 20 for $n = 729$, there is a sudden increase in the number of dependencies found of order 9.

This indicates that hBOA has solved many of the lower order subproblems of order 3 and is not proceeding up the hierarchical level to attempting to solve problems of order 9. Once hBOA has spent generations working on solving the subproblems of order 9, it is easy to see hBOA again jump to the next hierarchical level and start to find dependencies of order 27. At generations 25 and 45 respectively, each of the two graphs show an increase in the number of dependencies of order 27 discovered. Some generations later, as hBOA again solves the current hierarchical level and moves to the next, each of the graphs again show a jump in the number of order 81 dependencies. We also see that as the higher order subproblems are solved, the lower order dependencies start to decrease. This is because, similarly to trap-5 and rADPs, these subproblems are converging to only a few different optima and can be encoded with less dependencies. The very last level of hierarchical order is not nearly as pronounced. This is because only a few of these dependencies are required to solve the final hierarchical level and most of these dependencies have already been included due to stochastic errors from the finite population size.

Initially it was argued that hTraps could not be solved using a single fixed decomposition. The results clearly show the hierarchical search progressing up each level in a regular fashion, with hBOA changing its decomposition over time. Each of the series of models examined showed that the types of models hBOA was generated changed at sharply defined stages when the previous hierarchical level was solved. The results also show that hBOA keeps most of the low order dependencies over the entire run. This is required as hBOA must preserve alternative solutions to the lower order building problems in order to solve higher levels.

The regularity of this process is a good sign for using hBOA to learn about our problems through automated means. It shows that it is possible for either researchers to be able to examine EDA models to see if problems are hierarchical in nature. Also, due to the stability in the models during any single hierarchical problem solving step, it should be possible to use these models for efficiency enhancement.

3.4 hBOA Models for 2D Spin Glasses

So far we have shown that on separable problems of bounded order hBOA is able to learn an adequate problem decomposition quickly and accuratly. Even on hierarchical problems, hBOA is able to find an adequate problem decomposition at each hierarchical level and is able to solve the problem step by step. While these problems are difficult for many optimization techniques, they are also artificial test problems and have a structure that is relatively easy to understand. The next step is to examine how hBOA reacts when solving a real-world problem where the structure is much harder to understand. This can be difficult, as when we move away from test problems it becomes much harder to define what a “perfect” or even

a "good" model is. However, regardless of the problem chosen, we hope that hBOA will be able to find adequate models to solve the problem and that the models will not change much during the runs.

To test hBOA's model building performance in a more complicated real-world scenario, we look at the problem of finding ground states of 2D Ising spin glasses (Binder and Young, 1986; Mezard et al., 1987; Fischer and Hertz, 1991; Young, 1998; Pelikan and Goldberg, 2003; Pelikan et al., 2004; Santana, 2005; Pelikan and Hartmann, 2006; Shakya et al., 2006), where the task is to find spin configurations that minimize the energy of a given spin glass instance. While it is relatively easy to undersatnd the structure of the energy function of 2D ising spin glasses, finding the ground states (minimal energy state) of these spin glasses is quite challenging for most optimization techniques. This is because the problems (1) contain a large number of local optima, (2) the fitness landscape is quite rugged and the local optima are often separated by regions of high energy, and (3) no decomposition of bounded order is sufficient to solve the problem. While the problem is solvable in polynomial time using analytical methods, most optimization techniques such as GAs and even state-of-the-art Markov chain Monte Carlo (MCMC) methods such as the flat-histogram MCMC (Dayal et al., 2004) fail to solve the problem in polynomial time. However, hBOA is able to achieve empirical asymptotic performance equal to the best analytical methods without any problem-specific knowledge (Pelikan and Goldberg, 2003), and as such it is a good real-world problem to use to gain insights into how hBOA model building deals with a more complicated real-world problem.

3.4.1 2D Ising Spin Glass

An Ising spin glass is typically arranged on a regular 2D or 3D grid where each node i corresponds to a spin s_i and each edge $\langle i,j\rangle$ corresponds to a coupling between two spins s_i and s_j. For the classical Ising model, each spin s_i can be in one of two states: $s_i = +1$ or $s_i = -1$. Each edge $\langle i,j\rangle$ has a real value (coupling) $J_{i,j}$ associated with it that defines the relationship between the two connected spins. To approximate the behavior of the large-scale system, in this section we consider the 2D Ising spin glass with periodic boundary conditions, which introduce a coupling between the first and the last element along each dimension.

A specific set of coupling constants define a spin glass instance. Each possible setting of all spins is called a spin configuration. Given a set of coupling constants $J_{i,j}$, and a configuration of spins $C = \{s_i\}$, the energy can be computed as

$$E(C) = \sum_{\langle i,j\rangle} s_i J_{i,j} s_j \ , \tag{3.5}$$

where the sum runs over all couplings $\langle i,j\rangle$. Here the task is to find a spin configuration given couplings $\{J_{i,j}\}$ that minimizes the energy of the spin glass; see figure 3.11 for an illustration. The states with minimum

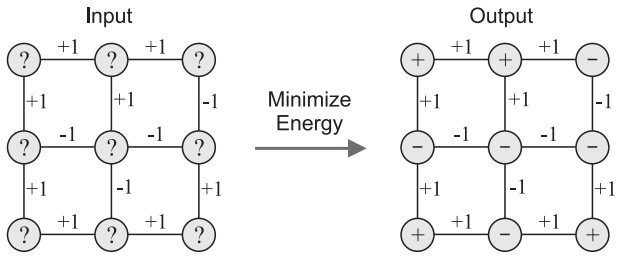


Figure 3.11: An exampling of finding the minimum energy (ground state) of a 2D Ising spin glass. For simplicity, periodic boundary conditions are omitted in this figure.

energy are called *ground states*. In hBOA, the spin configurations are encoded with binary strings where each bit specifies the value of one spin (0 for +1, 1 for -1).

In order to obtain a quantitative understanding of the disorder in a spin glass system introduced by the random spin-spin couplings, one generally analyzes a large set of random spin glass instances for a given distribution of the spin-spin couplings. For each spin glass instance, the optimization algorithm is applied and the results are analyzed. Here we consider the $\pm J$ spin glass, where each spin-spin coupling constant is set randomly to either $+1$ or -1 with equal probability. All instances of sizes up to 18×18 with ground states were obtained from S. Sabhapandit and S. N. Coppersmith from the University of Wisconsin who identified the ground states using flat-histogram Markov chain Monte Carlo simulations (Dayal et al., 2004). The ground states of the remaining instances were obtained from the Spin Glass Ground State Server at the University of Cologne (Spin Glass Ground State Server, 2004).

To improve the performance of hBOA, we incorporated a deterministic local search—deterministic hill climber (DHC)—to improve quality of each evaluated solution (Pelikan and Goldberg, 2003; Pelikan, 2005). DHC proceeds by making single-bit flips that improve the quality of the solution most until no single-bit flip improves the candidate solution. DHC is applied to every solution in the population before it is evaluated.

3.4.2 Perfect Model for 2D Ising Spin Glass

Determining the perfect model for 2D Isin spin glass is much harder than for the previous three problems. Every spin to some degree depends on every other spin, either directly through a neighbor coupling or through a chain of neighbor couplings. It is not clear at all what dependencies are necessary to solve the problem in a scalable fashion and what dependencies should be avoided. It has been argued (Mühlenbein et al., 1999) that to achieve provable convergence on 2D spin glasses, the order of dependencies in the probabilistic model must grow at least as fast as $\Omega(\sqrt{n})$. However, hBOA has been shown to solve 2D Ising spin glasses in polynomial time and the population size in hBOA is lower bounded by $\Omega(2^k)$ (where k is the order of dependencies covered by the probabilistic model). If it was really required for hBOA to consider all

dependencies of order $\Omega(\sqrt{n})$ or more to solve 2D Ising spin glasses, it could not solve them in polynomial time.

So, while it is not clear what a perfect model for a 2D Ising spin glass is, it should be expected that interactions between neighbors will be the strongest interactions and that the level of interaction will decrease as the distance between spins increases (with the distance measured by the shortest number of couplings between spins). Given this intuition, it should be possible to judge the accuracy of hBOA model building on 2D Ising spin glasses. If the majority of dependencies that hBOA finds are of short coupling distance, then that would imply that hBOA is finding mostly good dependencies.

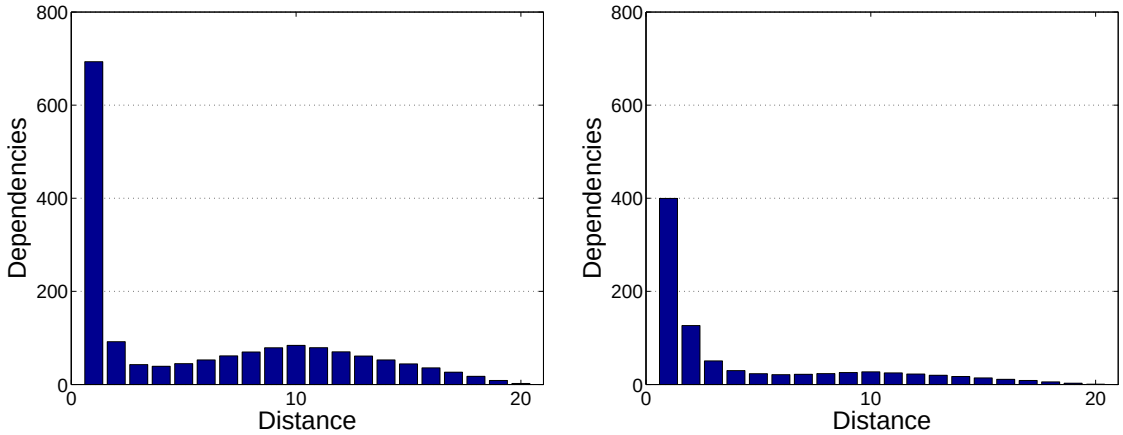
3.4.3 Experimental Setup

Since the difficulty of spin glass instances varies significantly depending on the couplings (Dayal et al., 2004; Pelikan and Goldberg, 2003), we generated 100 random instances for each considered problem size. On each spin glass instance we used the population size obtained by bisection but since we used 100 random instances for each problem size, we required hBOA to converge only in 5 out of 5 independent runs. Again, truncation selection was used with $\tau = 50\%$. Before each solution is evaluated for fitness, a local searcher (DHC) is used on the solution.

3.4.4 Model Structure for 2D Ising Spin Glass

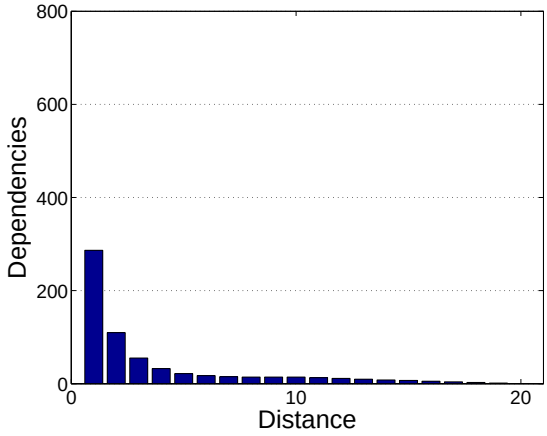
We start the analysis of hBOA models on 2D Ising spin glasses by examining the coupling distance between spins, where the distance between spins is defined as the minimum number of couplings in the 2D grind that must be passed to get from one spin to another. Due to periodic boundary conditions, the maximum distance is equal to the square root of problem size. So for example, in a 20×20 2D Ising spin glass of size $n = 400$, the maximum distance is 20. The number of possible dependencies at any given distance is at the maximum at half of the maximum distance. There is only one possible dependency of max distance. For example, a 2D spin glass of size 20×20 has 4 dependencies of distance 1, 8 dependencies of distance 2 and so forth until it reaches the maximum of 38 dependencies of distance 10. It then has 36 dependencies of distance 11, 32 dependencies of distance 12 and so forth until only one dependency of the maximum distance of 20.

In order to examine the number of dependencies at each distance discovered in the probabilistic model, we will use histograms of the average number of dependencies at each distance for four snapshots of hBOA model building. The four snapshots considered are the model generated in the first generation, the model generated a third of the way through a run, the model generated in the final third of the run and the last

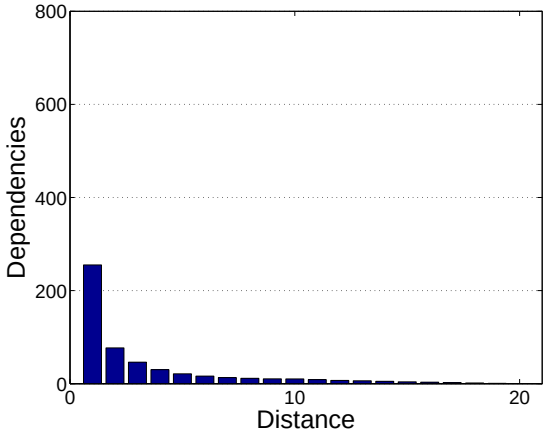


(a) First generation

(b) Second snapshot



(c) Third snapshot



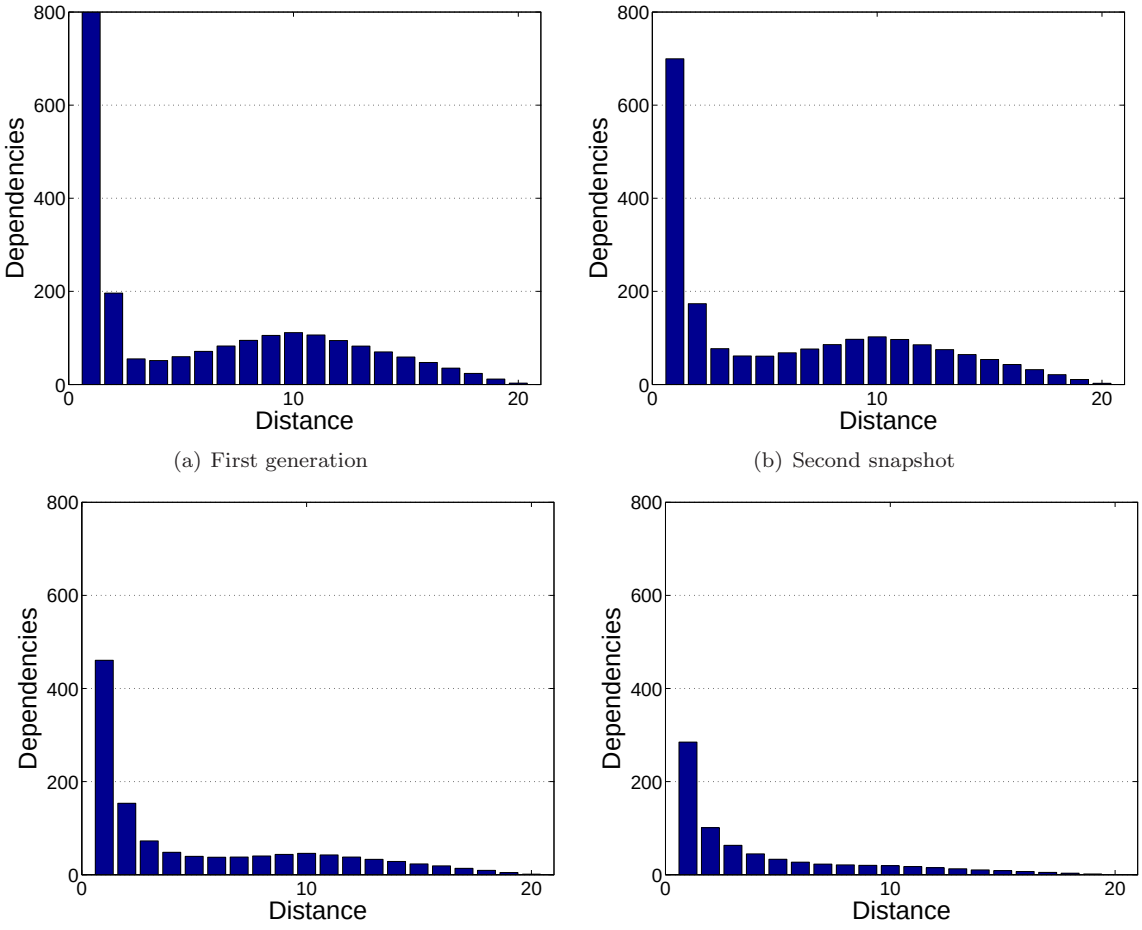
(d) Last Generation

Figure 3.12: The distribution of dependencies by distance on 2D Ising spin glasses using DHC. The four snapshots were equally distributed over each run.

model generated.

As discussed in section 3.4.2, we would hope that most of the dependencies would be between neighbor spins. Figure 3.12 confirms that this is the case, with the most dependencies being found of distance 1 in all four snapshots. In addition, the the number of dependencies between immediate neighbors is as large as the total number of the remaining dependencies. Since the influence of each spin decreases significantly with distance, it is possible that the dependencies of long-distance spins are not necessary and are only a result of strong problem regularities and noisy information in the initial population.

It is also immediately obvious from figure 3.12 that the models become simpler over time. Each snapshot of hBOA model building shows less dependencies at any distance. While the effective order of dependencies encoded by the model might increase over the run, these dependencies become simpler over time as they have to cover less alternative partial solutions. So, similarly to trap-5 and rADPS, some dependencies



(a) First generation

(b) Second snapshot

(c) Third snapshot

(d) Last generation

Figure 3.13: The distribution of dependencies by distance on 2D Ising spin glasses without any local search. The four snapshots were equally distributed over each run.

that are necessary early can be covered with chains of short-order dependencies. Another interesting result demonstrated in figure 3.12 is the increased number of mid-distance dependencies early on in the snapshots. This is believed to be a result of the increased total number of dependencies at this distance (as mentioned earlier, 38 dependencies of distance 20 in a 20×20 spin glass but only 4 possible neighbor dependencies), some of which are found from stochastic errors due to the use of a finite population.

For the previous results on model quality in this chapter, DHC was not used. To examine whether DHC significantly alters hBOA model dynamics for 2D Ising spin glass, the histogram experiments were repeated without DHC in figure 3.13. These results show the same general pattern as the experiments without DHC. Again, the most dependencies found are of distance 1. However, more long distance dependencies are discovered, with the models in general more complex than the results with DHC. This is to be expected, as the population has increased diversity compared to the DHC experiments, which would ensure that all

solutions are at least a local optima.

3.4.5 Model Dynamics for 2D Ising Spin Glass

The next question in regards to hBOA model quality for 2D Ising spin glass is to examine how stable individual dependencies are between generations. Much as we examined previous test problems, we do this by looking at the number of dependencies that are changed and remain the same in each generation. Figure 3.15a shows the obtained results for one run of a 20×20 spin glass. These results seem to show a great deal of change between generations in the model, unlike what all previous test problems have shown. While this only is one run, other runs examined showed a similar pattern. However, if instead of examining all dependencies, we only consider the strongest dependencies when examining what dependencies are changed between generations, a different pattern emerges. Figure 3.15b only considers those dependencies between neighbors. When considering only the strongest dependencies, we see a similar pattern to that seen in the other problems in this chapter, with relatively little change between generations, indicating hBOA finds a relatively stable model very early on. The large variation in dependencies seen in figure 3.15a is a result of the long-range dependencies. Since many long-range dependencies are most likely unnecessary, it would be expected that the stability of long-range dependencies is much less important than the stability of short-range dependencies. In addition, long-range dependencies can be represented in many ways. Still, it is clear that for spin glasses the models do change over time. This is not surprising, as spin glasses are very complex and might require hierarchical problem solving.

3.4.6 Commonality in 2D Ising Spin Glass instances

As with NK landscape instances, the next question to explore is whether models generated from different sized 2D Ising spin glass instances still share important commonalities. If they do, in other words if those important dependencies in one instance share a quality with the important dependencies found in other instances of different sizes, then we should be able to exploit this information in the future to speed up problem solving when scaling up to larger problem sizes.

We examine the proportion of splits between variables by distance for different sizes of 2D Ising spin glass instances in Figure 3.14. 100 instances of sizes 24×24 and 28×28 were examined, with 10 runs examined per instance. The results show a strong correlation between the different problem sizes. In fact, the proportion of splits by distance is almost identical between the two sizes of problems. Much as with NK landscapes, this is good news, as it means that it is possible to gather statistics on smaller problems and that these statistics can then be used to speed up problem solving on much larger problems.

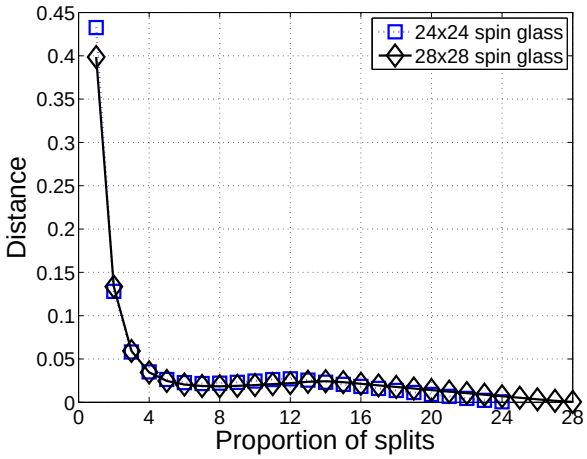


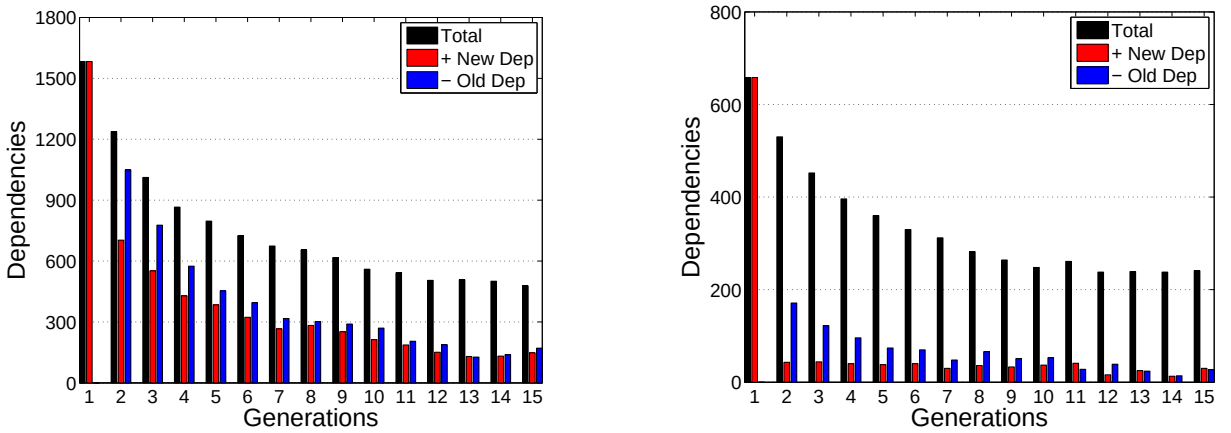
Figure 3.14: The proportion of splits between variables of different distances on 2D Ising spin glasses of sizes 24×24 and 28×28.

3.4.7 Restricting hBOA Models on Spin Glass

One of the primary goals of this thesis is to show that it is possible to use the data gathered from hBOA model building to bias future model building on similar problems. We have just observed that for 2D Ising spin glasses, the majority of the dependencies discovered are of distance one. The obvious question to ask is, if restricted to only dependencies of distance one, would hBOA still be able to solve the problem scalably and reliably? If this is too restrictive, would it be possible to pick an arbitrary distance to restrict the model building and then be able to solve all 2D Ising spin glasses of any size? This is an important question, as if we can restrict hBOA models significantly without effecting scalability, this should result in dramatic speedups of hBOA on 2D Ising spin glasses and similar problems.

To examine whether it is possible to restrict hBOA model building significantly without effecting scalability on spin glasses, the scalability of hBOA with different restrictions on the dependencies was examined. Figure 3.16 shows the number of fitness evaluations needed for hBOA to solve spin glass instances of sizes $10 \times 10 = 100$ spins to $30 \times 30 = 900$ spins with three different levels of restrictions on hBOA model building. 100 instances of each problem size were used. The results show that if hBOA is restricted to only distance 1 dependencies (neighbors), then in all cases it performs worse than the unrestricted version of hBOA, with the performance getting comparatively worse as problem size increases. It is clear that restricting hBOA to only distance one is ineffective for spin glasses. Restricting hBOA to only distances of 2 or less does lead to a performance increase for the smaller problem sizes.

To answer the above question, we looked at the scalability of hBOA with the different restrictions on the dependencies using spin glass instances of sizes from $10 \times 10 = 100$ spins to $30 \times 30 = 900$ spins, using



(a) All dependencies

(b) Neighbor dependencies

Figure 3.15: The number of dependencies that change by generation for specific runs of hBOA on 20×20 spin glass. The first bar (total) represents the total number of dependencies discovered during that generation. The second bar represents the number of dependencies discovered that were not discovered during the previous generation. The final bar represents those dependencies that were discovered in the previous generation but were not discovered in the current generation.

100 instances for each problem size. For each problem instance and each hBOA variant, we used bisection to find an adequate population size and recorded the average number of evaluations until convergence. The results are shown in figure 3.16. However, as the problem size is further increased, the performance gets much worse than the unrestricted hBOA and starts to scale exponentially, with hBOA not able to solve some of the larger instances even with a population size of 500,000.

The results indicate that to solve 2D Ising spin glasses scalably it is not possible to restrict the dependencies to only those of neighbors or those spins at only a distance of two apart. These results agree with those presented in (Yu, 2006) and show that some of the hand-designed Bayesian network structures based on the running intersection property proposed in (Mühlenbein et al., 1999) are unlikely to scale well. In other words, in spite of the availability of complete knowledge of problem specifics, models obtained with automatic model building procedures of hBOA outperform hand-crafted models.

However, it is clear that it is possible to restrict hBOA models significantly, it is just not possible to fix them irregardless of problem size on spin glasses without effecting hBOA scalability. In the next chapter we will show that it is possible to restrict hBOA model building in general to increase performance of hBOA on different sizes of spin glasses, but this is done based on trial runs of hBOA and so does not restrict to the same level regardless of problem size.

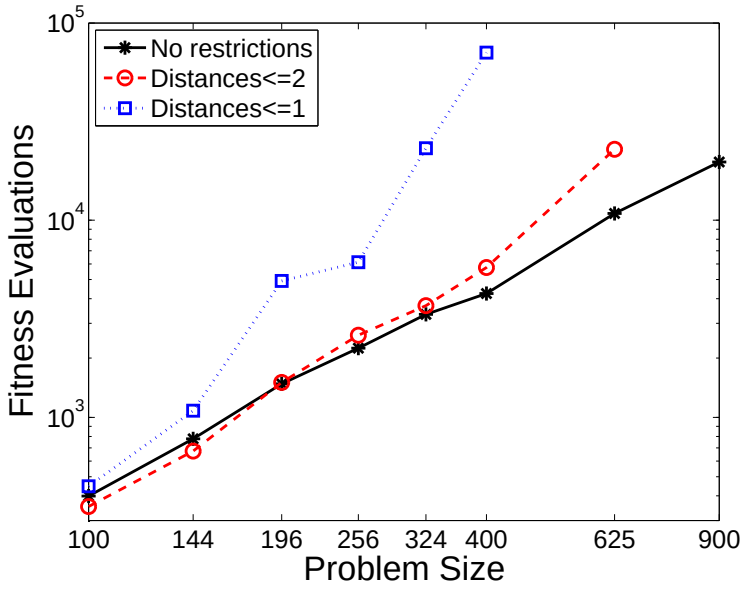


Figure 3.16: Number of evaluations needed by hBOA to solve various sizes of 2D Ising spin glass based on different levels of model restrictions. The base case has no restrictions at all. Then model building is restricted to only allow dependencies of distance 2 or less. Finally, the case where only neighbor dependencies are allowed is considered.

3.5 Summary of Results on Model Quality

This chapter applied hBOA to four test problems: (1) concatenated traps, (2) nearest neighbor NK landscapes, (3) hierarchical traps and (4) two-dimensional $\pm J$ Ising spin glasses with periodic boundary conditions. The models were then examined to see how closely they resembled the "perfect" theoretical model that corresponds to the underlying optimization problem. Then based on observing the most common dependencies discovered when solving the Ising spin glass problem, experiments were run to determine the benefits of restricting hBOA model building to only the most promising dependencies. A summary of the key points of this chapter follows:

- The results strongly show that hBOA is able to learn accurate probabilistic models quickly and that the learned problem decomposition corresponds closely to the underlying problem, with hBOA able to accurately learn important problem regularities.
- For concatenated traps, nearest neighbor NK landscapes, and 2D Ising spin glasses, the models are relatively stable over the entire run, with hBOA able to learn a reasonably accurate model early in the run.
- The models generated when solving nearest neighbor NK landscapes and 2D Ising spin glasses share many commonalities, with the proportion of dependencies by distance staying almost identical as

problem size scales up.

- For hierarchical traps, the models must change over time in order to adapt to each level of difficulty as hBOA models up the hierarchy one level at a time. However, at any specific level of the hierarchy, models change only slightly until the next level.

- While it was discovered that it is indeed possible to restrict hBOA model building by hand to increase performance, this could result in poor scaling.

- Since hBOA is able to capture important problem regularities, the results show that hBOA is learning valuable information about the problem structure that should be useful when solving similar problems in the future.

- The results show that the models change only a small amount over time. This is promising since if the models changed drastically over the lifetime of a run it would be very difficult to mine the models for useful information.

- Given the previous two results, it should be possible to develop methods to exploit this information to speed up problem solving. Several proposed methods to do just that will be discussed in the next section.

Chapter 4

Hard Biasing hBOA Model Building[3]

Chapter 3 showed that hBOA model building captures many of the essential problem features of a broad variety of test problems. Chapter 3 also showed that it was possible to restrict hBOA model building to increase speed when solving 2D Ising spin glasses, but it was clear that the restriction can not simply be fixed in place regardless of problem size. These results were important, as they showed that hBOA not only learns enough information to solve the problem (a necessary prerequisite for hBOA's effectiveness) but also captures enough information about problem features that we would hope to be able to exploit that information when solving similar problems.

This chapter will attempt to take the first step towards designing an automated technique for exploiting the information in probabilistic models learned in the past to speed up future EDA runs. Since from chapter 3 we saw that it is not possible to restrict hBOA model building arbitrarily without considering problem size, our method will first gather information about the problem from previous runs on similar problems.This information will then be used to restrict hBOA model building to only the most promising dependencies for future runs. If done correctly, hBOA should have a smaller and more promising search space to consider when building the probabilistic model, resulting in performance gain. In this chapter we only consider hBOA model building, but the methods discussed should also be applicable to other EDAs that use multivariate probabilistic models.

This chapter starts by proposing two methods for automatically biasing model building in hBOA using the probabilistic models learned in previous hBOA runs on similar problems. Section 4.1.2 describes the probability coincidence matrix (PCM) and how it is used to bias model building. Section 4.1.3 then describes how to bias hBOA model building based on a distance metric that rates the level of interaction between variables in the problem. Section 4.2 then presents the experimental results. Finally, section 4.3 summarizes the results of restricting hBOA model building based on hard bias.

[3]This chapter is based in part on work previously published in Hauschild, M., Pelikan, M., Sastry, K., & Goldberg, D. E. (2011). Using Previous Models to Bias Structural Learning in the Hierarchical BOA. *Evolutionary Computation*, 20 (1), 135-160 and Hauschild, M. & Pelikan, M. (2008) Enhancing Efficiency of Hierarchical BOA Via Distance-Based Model Restrictions. *Proceedings of the Parallel Problem Solving from Nature Conference (PPSN-2008)*, 417–427

4.1 Biasing the Model Building

The series of models produced by an EDA as it solves a problem contain a tremendous amount of information about the problem landscape. While it should be possible to exploit that information to point the EDA to areas worthy of further exploration, developing methods to do so automatically have been difficult. One way to attempt to exploit this information would be to perform some trial runs of an EDA on the problem at hand and use the information contained in these models to restrict hBOA model building in some way. If the restrictions are good, then hBOA should only consider a subset of possible dependencies and this should lead to an inprovement in overall performance. However, as section 3.4.7 showed, if the restrictions are poorly done or overly restrictive, we would expect hBOA scaling to be negatively effected or be unable to solve the problem.

This section describes two approaches to restricting the model building in hBOA to speed up performance:

(1) Restrict model building using the probability coincidence matrix, and

(2) Restrict model building using a distance threshold.

In the first method, some instances of the problem are run to generate a sequence of probabilistic models. These models are then processed to generate information that is then used to restrict hBOA to consider only the most promising dependencies during model building. In the second method, prior knowledge of the problem structure is used to restrict hBOA to consider only the most promising dependencies.

To test the performance of these methods on the model building efficiency and speed of hBOA, it is necessary to examine a variety of problems. When testing the probability coincidence matrix, we will look at two optimization problems: (1) Concatenated traps of order 5 and (2) 2D Ising spin glass with $\pm J$ couplings and periodic boundary conditions. For restricting by distance threshold, we test four additional optimization problems: (4) Maxsat, (5) Nearest neighbor NK landscapes, (6) 3D Ising spin glasses and (7) Minimum vertex cover.

This section begins by describing the test problems. The probability coincidence matrix is then described and how it is used to restrict hBOA model building is discussed. Then the distance threshold method is discussed, which uses a defined distance metric on a problem to bias hBOA model building. Next, the expected benefits of correctly restricting hBOA model building are discussed. Lastly, the difficulties encountered if the restrictings are chosen incorrectly are discussed.

4.1.1 Test Problems

This section describes the test problems MAXSAT, minimum vertex cover (MVC) and 3D Ising spin glasses. The other test problems, trap-5, nearest neighbor NK landscapes and 2D Ising spin glasses, are described in Section 3.1.1, Section 3.2 and Section 3.4.1.

MAXSAT

In MAXSAT, the task is to find the maximum number of clauses which can be satisfied in a given propositional logic formula in conjunctive normal form. MAXSAT is an important problem in complexity theory and artificial intelligence because it is NP-complete and many other important problems can be mapped to MAXSAT in a straightforward manner. MAXSAT has also become popular in evolutionary computation and a number of researchers studied performance of various genetic and evolutionary algorithms on this class of problems (Rana and Whitley, 1998; Gottlieb et al., 2002; Pelikan and Goldberg, 2003; Boughaci and Drias, 2004; Brownlee et al., 2007).

While in general MAXSAT is hard for many optimization techniques, it can be particularly difficult for some search algorithms such as GAs (Rana and Whitley, 1998). Rana and Whitley hypothesized that this is due to short-order partial solutions leading these methods away from the optimum. In addition, the traditional recombination operators of GAs are not able to handle the complex interaction between variables in many MAXSAT instances. However, it has been shown that hBOA is able to deal with these complex interactions and can successfully solve many MAXSAT problems (Pelikan and Goldberg, 2003)

MAXSAT problems are often expressed with logic formulae in conjunctive normal form with clauses of length at most k; formulae in this form are called k-CNF formulae. A CNF formula is a *logical and* of clauses, where each clause is a *logical or* of literals. Each literal can either be a proposition or a negation of a proposition. An example of a 3-CNF formula with 4 propositions X_1, X_2, X_3, X_4 is

$$(X_4 \vee X_3) \wedge (X_1 \vee \neg X_2) \wedge (\neg X_4 \vee X_2 \vee X_3) \tag{4.1}$$

An interpretation of propositions assigns each proposition either true or false. The task in MAXSAT is to find an interpretation that maximizes the number of satisfied clauses in the formula. In the example from equation 4.1, the assignment setting all literals to true would satisfy all the clauses in the formula and therefore it would be one of the global optima of the corresponding MAXSAT problem. hBOA encodes the interpretations with binary strings where each bit specifies an assignment of one proposition (0 for false, 1 for true). Of note is that MAXSAT is NP complete for k-CNF if $k \geq 2$.

In this section we consider instances of combined-graph coloring translated into MAXSAT (Gent et al., 1999), which are especially interesting because they are often difficult for standard MAXSAT heuristics such as WalkSAT (Pelikan, 2005) and because they represent a class of problems where randomness is combined with structure (Gent et al., 1999). In graph coloring, the task is to color the vertices's of a given graph so that no connected vertices's share the same color, with the number of colors bounded by a constant. Any given graph-coloring instance can be mapped to a MAXSAT instance by having one proposition for each pair (color, vertex) and creating a formula that is satisfiable if and only if exactly one color is chosen for each vertex and the colors of vertices's corresponding to each edge are different.

The graph-coloring problem instances were created by combining regular ring lattices and random graphs with a specified number of neighbors (Gent et al., 1999). Combining two graphs consists of selecting (1) all edges that overlap in the two graphs, (2) a random fraction (1-p) of the remaining edges from the first graph, and (3) a random fraction p of the remaining edges from the second graph. By combining regular graphs in this way, the amount of structure and randomness in the resulting graph can be controlled, since as p decreases the graphs become more regular (for $p = 0$, we are left with a regular ring lattice). This allows us to test methods against MAXSAT instances with varying amounts of structure. The ring lattice is constructed by ordering the vertices cyclically and connecting each vertex to the eight closest neighbors. All tested instances of combined-graph coloring were downloaded from the Satisfiability Library SATLIB (Hoos and Stutzle, 2000) and more details on these instances can be found elsewhere (Gent et al., 1999).

3D Ising Spin Glass with $\pm J$ couplings

While 2D Ising spin glasses are described in section 3.4.1, in this chapter we will also consider 3D Ising spin glasses where each spin has 6 neighbors. 3D Ising spin glasses have a much higher connectivity than 2D Ising spin glasses and are NP-Complete, making them a much tougher optimization problem than their 2D counterparts.

Minimum Vertex Cover

A vertex cover of an undirected graph $G = (V, E)$ is a subset $S \subseteq V$ such that if $(u, v) \in$ E then either $u \in S$ or $v \in S$ or both. The minimum vertex cover problem is to find the vertex cover of a graph G with the smallest number of vertices. This problem is considered because much like MAXSAT, the problem is NP-Complete (Karp, 1972) and can be mapped to other hard graph problems.

In this thesis we will study $G(n, m)$ graphs (Bollobas, 2001; Weigt and Hartmann, 2001), which is a random graph model from graph theory. The $G(n, m)$ model consists of all graphs with n vertices and m

edges such that m and n are related by $m = nc$, where $c > 0$ is a constant. To generate a random $G(n, m)$ graph we start with a graph with no edges. Then cn edges are generated randomly over all possible graphs with cn edges. Each vertex is then expected to connect to $2c$ other vertices on average.

4.1.2 Biasing Models Using the Probability Coincidence Matrix

In order to use previous model information to intelligently restrict model building in the future, we must first store the model information generated by the early runs of the EDA. To do this, we will use what we call a *probability coincidence matrix* (PCM). The PCM is of size $n \times n$, where n is the size of the candidate solutions. In this chapter, we will denote the PCM by P and the elements of the matrix P as P_{ij} where $i, j \in \{1 \dots n\}$. The value of P_{ij} is defined as the proportion of probabilistic models in which the ith and jth string positions are connected in either direction, with the elements of P calculated by examining all available probabilistic models in the trial runs. For example, if for any $i \neq j$, 25% of the available probabilistic models contain a connection between i and j, then $P_{ij} = 0.25$.

In this work we do not consider the time t at which any particular model is discovered and instead all models are considered equally important. However, for some problems it might be beneficial to consider this time. In this case, we could generate a sequence of PCM's P^t_{ij}, with P^t_{ij} storing the proportion of probabilistic models in generation t where i and j were connected in either direction. We would expect this to be required if models changed significantly over time. It would also be possible to weigh some models more heavily then anothers. However, if the models do not change significantly over time it would be better to ignore the time component so that we have a larger set of sample runs to use to restrict hBOA model building.

Another thing to remember with this method is that all generations are treated equally. This means that longer runs will be more strongly represented in the PCM. While it is certainly possible to implement a method that weighs all runs equally, we have not done so. Indeed, not all EDA runs are equal. Some similar problems are harder than others and it is possible that it is beneficial to have these harder runs represented more strongly. By weighing all models equally and ignoring the time component t, P_{ij} represents the actual percentage of models that connected i and j. Adding additional components would make intuitively understanding the meaning of P_{ij} more difficult.

To demonstrate the method used to generate the PCM, figure 4.1 shows 3 example coincidence graphs for a 4-bit problem and the resulting PCM generated. In each of the coincidence graphs, an entry of 1 indicates that the decision variables represented by that row and column are connected in the Bayesian network. As we see, the PCM represents the percentage of models that had an edge between decision variables.

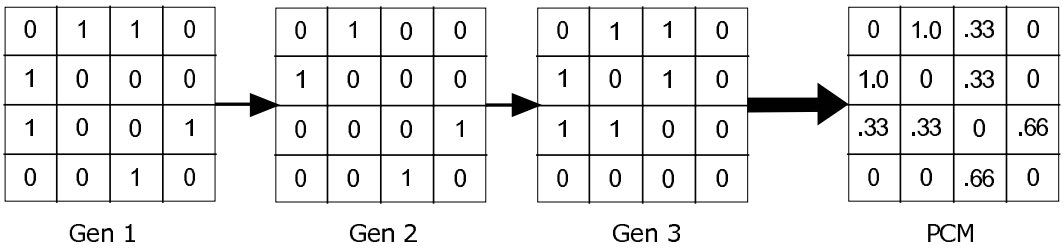


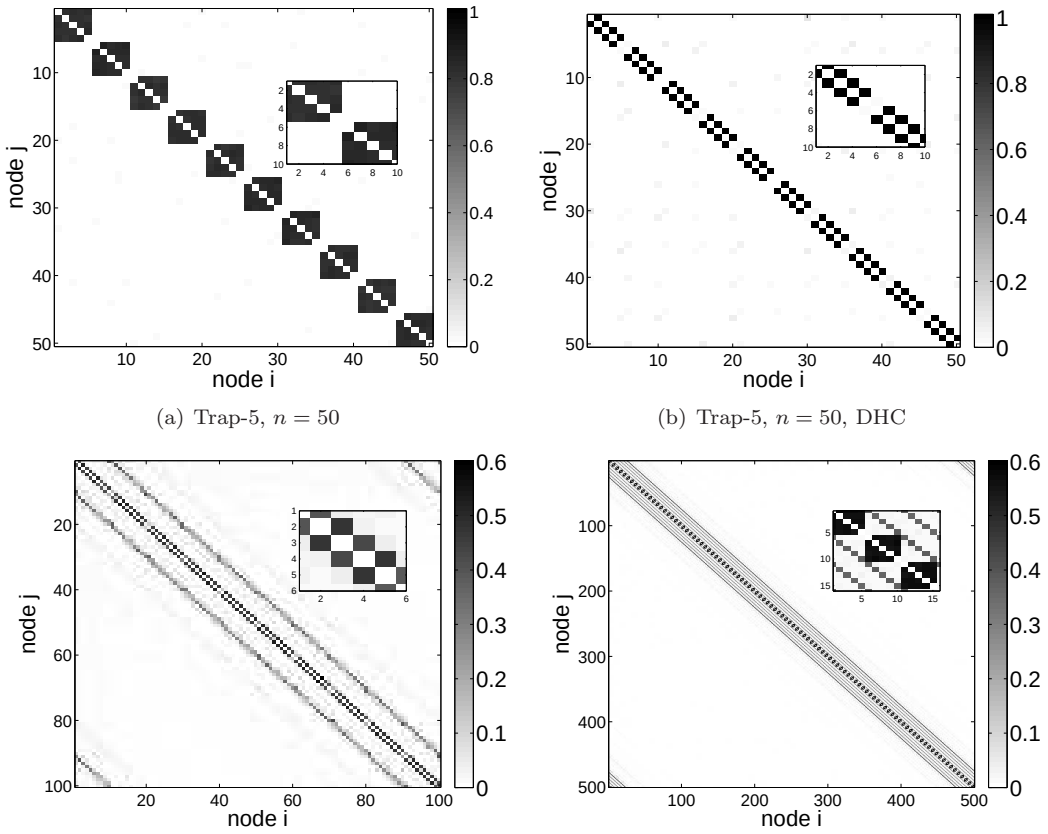
Figure 4.1: Example coincidence graphs for a 4 variable problem and the resulting PCM generated.

The previous paragraph only showed a toy example. How does the PCM look in practice on more complicated problems? To examine this question, figure 4.2a shows the PCM generated from the probabilistic models generated from hBOA when used to solve trap-5 of size $n = 50$. As discussed in section 3.1.4, in order for hBOA to solve trap-5 effectively, it is necessary for hBOA models to connect the decision variables in each trap partition and this should be obvious in the PCM. It is easy to see from figure 4.2a that this is the case. As expected, variables in the same trap partition are connected with a very high frequency and variables in different trap partitions are very rarely connected.

In the experiments later on in this chapter, we will be using DHC on many of the problems. To examine the effect of DHC on trap-5, we also look at the PCM generated from 30 runs of trap-5 using DHC on a trap-5 of size $n = 50$ in figure 4.2b. The results clearly show that when using DHC on trap-5, most of the dependencies discovered by hBOA are simple chain dependencies in the trap partitions. This is because with DHC the majority of trap partitions in each candidate solution are represented as strings of 1's and 0's. This dramatically increases hBOA performance on trap-5 (Radetic et al., 2009), since hBOA is able to discover these patterns with a smaller population and the chain dependencies are sufficient to model each trap partition accurately.

Figure 4.2c shows a PCM generated from 100 random instances of 10×10 2D Ising spin glass, with bisection used to determine population size in 5 out of 5 independent runs. From the results in section 3.4.2, we would expect the majority of dependencies between variables to be those between variables close to each other in the underlying spin glass structure. The PCM in figure 4.2c clearly shows this, with particular neighbor variables connected with almost a 50% probability and variables at a distance of 2 being connected more than higher order distances.

Once we have computed the PCM from sample runs, it is relatively straightforward to use it to perform a hard bias on future runs. This is done by only allowing an edge between variables in the Bayesian network model if that edge is contained in some percentage p_{min} of the sample runs represented in the PCM. For example, if $p_{min} = 0.5$, then during hBOA model building we will only allow a conditional dependency



(a) Trap-5, $n = 50$

(b) Trap-5, $n = 50$, DHC

(c) Ising Spin Glass, $n = 100$

(d) MAXSAT,$p = 2^{-8}$

Figure 4.2: Probability Coincidence Matrices generated from 4 different types of problems. Closeups of areas of high interaction are shown in each.

between variables i and j if $P_{ij} \geq 0.5$. Using the PCM in figure 4.1, if p_{min} was set to 0.5, only conditional dependencies from $0 \leftrightarrow 1$ and $2 \leftrightarrow 3$ would be allowed. If this $p_{min} = 0.5$ was then used on the trap-5 example in figure 4.2a, it is easy to see that hBOA would only consider edges between variables that were in the same trap partition and would not consider any edges outside of a trap partition, which should greatly speed up the performance of hBOA model building. Examining figure 4.2c, it is also easy to see that it would be possible to set p_{min} in such a way as to restrict hBOA to only promising connections for 2D Ising spin glasses. However, in this example a good example of p_{min} that would result in speedups is not as clear.

The primary advantage that using PCM to restrict model building has is that this approach is applicable automatically and only has one additional parameter, the threshold p_{min}. However, the PCM method only works when the interactions between specific variables in one instance corresponds directly to the interactions between equivalent variables in another instance. To show an example when this is not the case, figure 4.2d shows a PCM gathered from hBOA models when used to solve 100 instances of combined-graph coloring

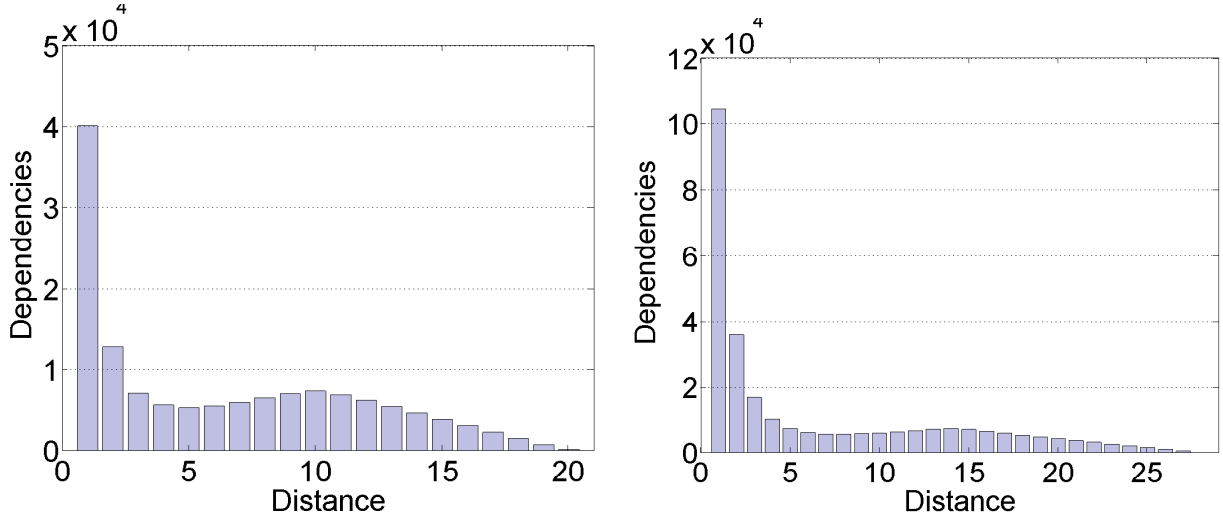
MAXSAT problems with $p = 2^{-8}$. In this example, the only visible structure shown in the PCM is that corresponding to the regular ring lattice, with all other variables being connected much less often. This is due to there being no guarantee that the relationship between two specific propositions in an instance will correspond to the relationship between the same propositions in another instance. It is clear from this example that restricting hBOA model building using the PCM method would not be effective on MAXSAT. For these types of problems, another method is required and this method is described in the next section.

4.1.3 Biasing Models Using Distance

The PCM bias approach allows us to specify one parameter, p_{min}, and then restrict model building directly based on information gathered from previous runs. However, this method suffers from a limitation: namely that it is only applicable when the structure of the underlying problem does not change dramatically between similar problems. As shown in figure 4.2, this is true for trap-5 and 2D Ising spin glass, and is also true for many other problems. However, it does not hold in general for all important classes of problems. Indeed, as shown in MAXSAT in figure 4.2d, often the relative level of interaction between specific problem variables changes between runs.

However, for many problems it is possible to define a distance metric between decision variables that corresponds to the strength of dependency between variables. For example, when looking at 2D and 3D Ising spin glasses, the shorter the path between two variables in the underlying problem structure the more likely a dependency should exist between these variables. As such, it should be beneficial to bias hBOA model building to more strongly or only consider those dependencies connecting variables that are close to each other in the underlying problem structure.

Defining a distance metric corresponding to the strength of interaction between variables is easy for many problems. For rADPS and trap-5, we can define the distance between variables in the same subproblem as being neighbor variables of distance 1. Variables that are not in the same subproblem but which share neighbor variables in another problem are of distance two, with the distance for any other variables defined as the minimum distance between variables passing through other variables. Similarly, in MAXSAT, the distance of variables in the same clause is of distance 1 and for any other pair of variables their distance is recursively defined as the minimum distance between these variables passing through any other variable. Since it is more likely that variables located in the same clause interact more strongly than variables that do not share a clause, we would expect that those variables of short distance in our distance metric should be connected more strongly in any model generated by hBOA. It is relatively trivial to define this distance metric representing the strength of dependency between variables on a broad variety of problems. Many



(a) $n = 400$ (b) $n = 784$

Figure 4.3: Histograms of the total dependencies discovered at each distance during 5 runs of two different spin glass instances of various size. Note the hump at the middle distance.

problems, such as the quadratic assignment problem, facility location problem, scheduling and others can easily be mapped to a distance metric. For example, for MVC the distance metric between variables is simply equal to the minimum number of edge traversals between the vertices represented by these variables in the underlying graph.

Once a distance metric is defined that represents the strength of dependency between variables, it is possible to restrict hBOA model building to only consider those dependencies equal to or below a distance threshold q_{max}. For example, if $q_{max} = 2$, then we would only consider dependencies between variables in hBOA model building if their distance in the distance metric is 2 or less. To examine the effects that such distance restrictions might have on hBOA model building, consider figure 4.3 which shows the number of dependencies discovered by distance discovered during 5 runs on two different instances of 2D Ising spin glass. As expected from section 3.4, most dependencies of shorter distances are discovered than longer ones. While it is not clear exactly what threshold q_{max} would lead to a speedup, it is intuitively clear from figure 4.3 and from section 3.4.7 that there should exist a q_{max} that leads to a speedup by focusing hBOA model building on promising dependencies only. However, we would expect that as problem size increases, that q_{max} would also have to increase.

4.1.4 Discussion on Model Restrictions

The two main benefits to restricting model building in hBOA and other EDAs to only consider the most promising dependencies is that (1) the model building becomes significantly faster because the algorithm

examines much fewer dependencies and (2) the search is more effective because simpler models are discovered which results in better mixing and model sampling. In section 4.2 we will show that these two factors can lead to dramatic performance gains.

While using the correct bias can lead to an improvement, care must be made that the correct method is chosen. While both of the methods discussed in this chapter restrict hBOA model building to only consider those dependencies strongly favored in successful runs, the PCM method should only be used when the absolute positions of problem variables determines their relative level of interaction between each other. While this is true for 2D Ising spin glass and trap-5, it does not hold for MAXSAT, MVC and rADPS. However, for these problems and many other graph based problems, it is relatively straightforward to define a distance metric on the relative strength of variable interaction. In these problems, the distance metric should be used instead.

4.2 Experiments

This section covers the experiments using the two proposed approaches to biasing hBOA model building on 2D and 3D Ising spin glasses, rADPs, MVC and MAXSAT. The PCM method is only used on 2D Ising spin glasses. The distance metric method is tried on all problem types. First the parameter settings and the experimental setup are discussed. The results are then presented.

To rank the computational performance increase from using the hard bias restrictions, we will use the term *speedup*, with speedup defined as the ratio of the original execution time and the execution time after restricting model building. For example, if the original execution time was 10 seconds and after restricting model building the execution time dropped to 5 seconds, we will say the speedup was 2. That is, hBOA with model restrictions was able to solve the problem twice as fast as the base hBOA.

4.2.1 Experimental Setup

To select promising candidate solutions, truncation selection will be used with threshold $\tau = 50\%$, which selects the best half of the current population based on the objective function to be used to generate models. To improve the performance of hBOA, DHC is used to improve the quality of each evaluated solution.

For all problem instances, bisection (Sastry, 2001b; Pelikan, 2005) is used to determine the minimum population size to ensure convergence to the global optimum in 5 out of 5 independent runs, with the results averaged over the 5 runs. The number of generations is upper bounded according to preliminary experiments and hBOA scalability theory (Pelikan et al., 2002b) by $n/4$ where n is the number of bits in the problem.

Each run of hBOA is terminated when the global optimum has been found (success) or when the upper bound on the number of generations has been reached without discovering the global optimum (failure).

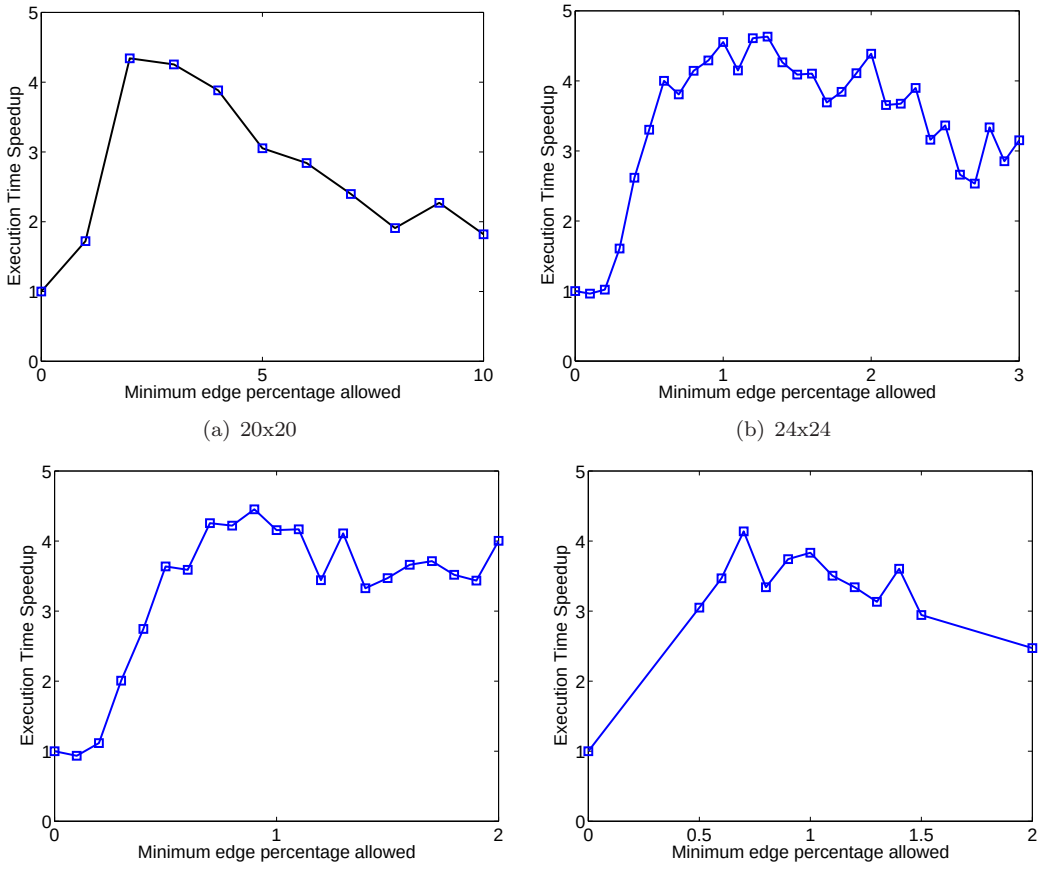
4.2.2 Experiments with PCM Model Bias on 2D Ising Spin Glasses

The most important question when using the PCM method is determining the threshold p_{min} for determining which edges to be excluded during model building. If the threshold is too large, hBOA will not be allowed to add necessary edges which could result in hBOA being unable to build an adequate probabilistic model. On the othe rhand, if the restriction is too small, then very few edges will be excluded and we would expect relatively little benefit from using the PCM.

To examine the influence of p_{min} on hBOA performance on 2D Ising spin glass, we consider problem sizes 16×16 to 32×32. For each of these problem sizes, 100 random instances were examined. In order to ensure that the same problem instances were not used to both learn the PCM as well as test the resulting bias on model structure, 10-fold crossvalidation was used. In 10-fold cross validation, the problem instances are divided up into 10 equally sized subsets and in each step, 9 of these subsets are used to learn the PCM and then the resulting PCM is used to bias hBOA model building on the remaining subset. This process is repeated 10 times, once for each subset.

To start the examination of hBOA performance with model restrictions, we first consider execution time. Figure 4.4 shows the average execution-time speedup for our four different problem sizes with varying amounts of model restriction p_{min}. The threshold for disallowing model dependencies was varied from $p_{min} = 0$ (no restrictions) to some maximum value, where the maximum value was set in order to ensure that no instances in the validation set become infeasible within reasonable computational time (this would indicate too severe model restrictions). We see that for all problem sizes, execution-time speedups of around 4–4.5 were obtained. Note that this execution time is the total execution time of the entire hBOA run, not simply the time spent in model building. We also see that the value of p_{min} that gives the largest speedup decreases as problem size increases. This is to be expected, as we would expect that the restrictions would need to allow more dependencies as problem size increases. However, even for large variations of p_{min}, speedups still result. It is also of note that as problem size increases, the optimal speedup stays nearly constant, indicating good scalability.

In the previous paragraph we examined the effects on overall execution time. However, also of importance is the effect on model-building efficiency alone. To attempt to quantify the effects of PCM model restrictions on hBOA model building, we recorded the number of bits that must be checked to update model paremters during the entire model building procedure, as this is the primary factor affecting the time complexity of



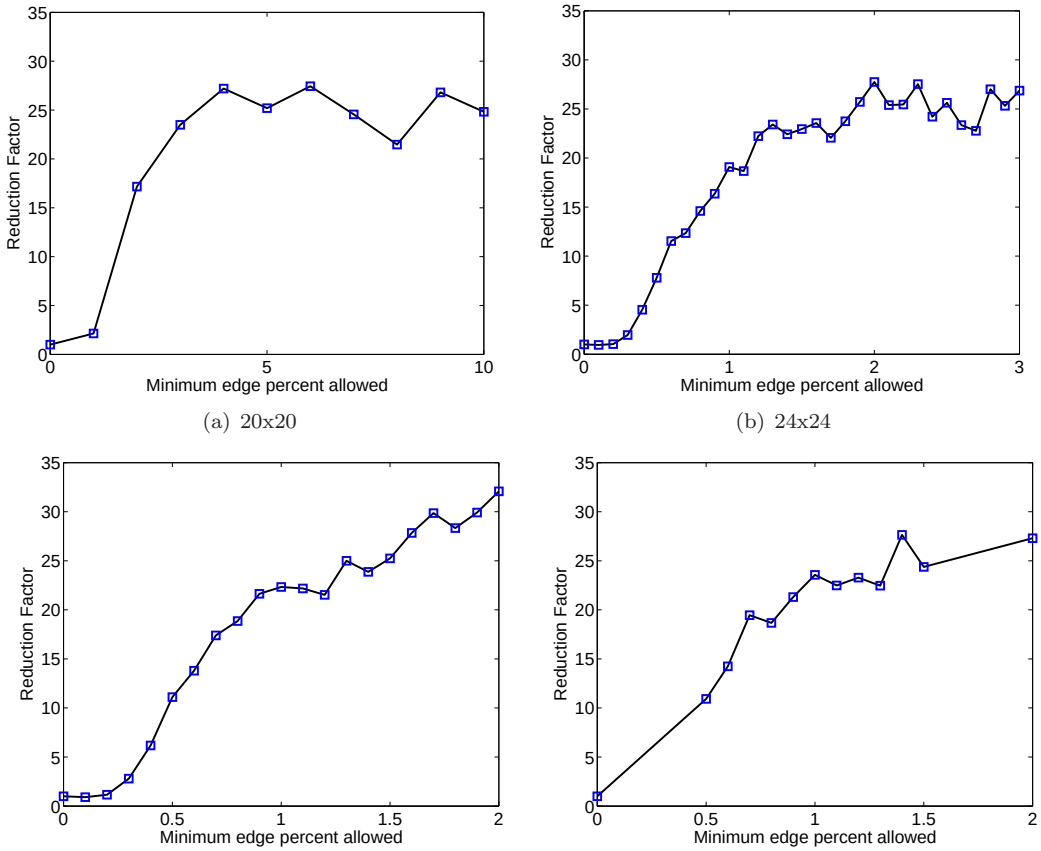
(a) 20x20 (b) 24x24

(c) 28x28 (d) 32x32

Figure 4.4: Execution time speedup for various PCM restrictions on model building for 100 different instances of 2D Ising spin glasses of various sizes.

hBOA model building. By restricting the number of potential edges ending in a particular node, after adding an edge into this node, the number of examined bits is reduced by the same factor. For Bayesian networks with local structures, limiting the model structure leads to a decrease in the number of potential splits on decision variables in the network.

Figure 4.5 shows the reduction factor by model restriction, where the reduction factor is the average reduction in the number of bits examined during model building. For example, if the reduction factor is 2 for a particular p_{min}, that means that only half as many bits were examined during model building. The results show a dramatic decrease in the number of bits examined during model building using the PCM method. We see that as p_{min} increases, the reduction factor increases. This reduction factor even keeps increasing past the optimal p_{min} in regards to minimal execution time. This seems to indicate that after a certain point, the restriction on model building is too severe and other factors instead of model building start to weigh down the overall execution time.



(a) 20x20 (b) 24x24

(c) 28x28 (d) 32x32

Figure 4.5: Factor of reduction in the number of bits examined during model building based on model restriction garnered from the PCM.

Table 4.1 shows the best speedups obtained for all problem sizes examined, the percentage of total possible dependencies considered by hBOA, and the corresponding threshold p_{min} used. We clearly see that for all problem sizes considered, nearly the same maximum speedup of 4–4.5 was found. We also see that as problem size increases that p_{min} must be decreased. This is most likely because as the problem becomes larger, the total number of dependencies increases and so hBOA needs to consider a larger set of dependencies in order to build a sufficient model. However, even as p_{min} increases, the percentage of total dependencies considered is very similar.

4.2.3 Experiments with Distance-Based Bias on 2D Ising Spin Glasses

In the previous section we examined the effects of PCM bias on 2D Ising spin glass. In this section we will examine the effects of distance-based bias. For 2D Ising spin glass, the distance metric will be defined as the minimum number of spin couplings that must be passed to travel from one spin to the other.

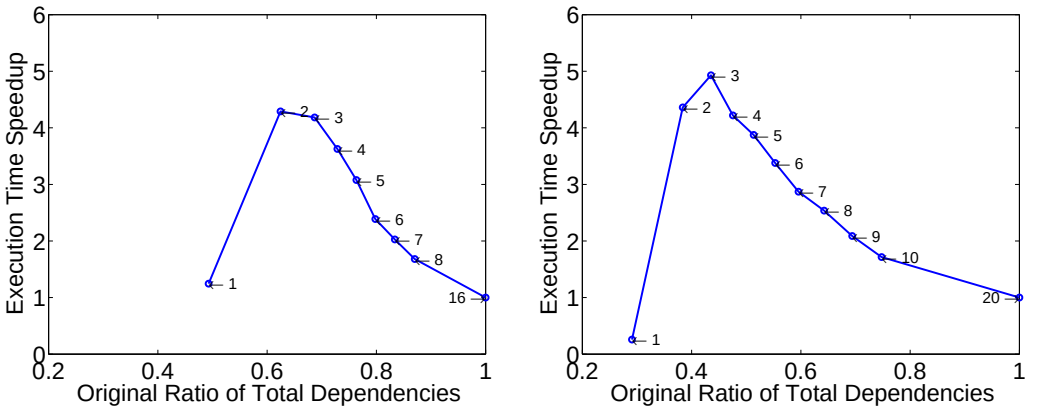
Size	Execution-time speedup	p_{min}	% Total Dep.
256 (16×16)	3.89	0.020	6.4%
324 (18×18)	4.37	0.011	8.7%
400 (20×20)	4.34	0.020	7.0%
484 (22×22)	4.61	0.010	6.3%
576 (24×24)	4.63	0.013	4.6%
676 (26×26)	4.62	0.011	4.7%
784 (28×28)	4.45	0.009	5.4%
900 (30×30)	4.93	0.005	8.1%
1024 (32×32)	4.14	0.007	5.5%

Table 4.1: Optimal speedups found and the corresponding PCM threshold p_{min} as well as the percentage of total possible dependencies that were considered for 2D Ising spin glass.

While it is clear that the probabilistic models discovered by hBOA contain mostly dependencies at shorter distances (see figure 4.3), as we discussed in section 3.4.7, setting an appropriate threshold for the maximum distance of dependencies is not trivial. Much as with the PCM method, if the restrictions are too severe the bias on model building towards simpler models will be too strong. On the other hand, if the distances are not restricted sufficiently, then we can expect little gain in performance.

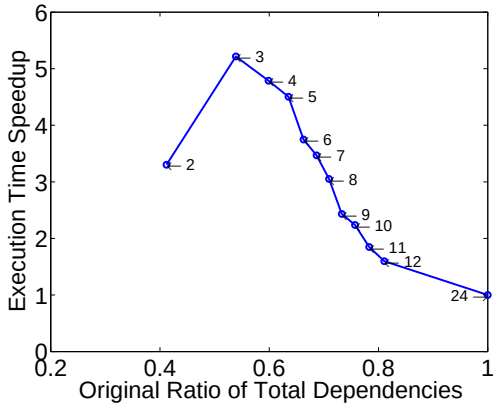
To examine the effects of distance-based restrictions on hBOA model building, problem sizes of 16×16 to 28×28 were considered, using 100 random instances for each problem size. Then hBOA was used to solve each instance, with dependencies restricted by a given maximum distance, which was varied from 1 to half the maximum achievable distance. So, for a 20×20 spin glass, experiments were run that only allowed dependencies between spins of a maximum distance from 1 to 10. For some of the larger problems and more severe restrictions, hBOA did not converge to a solution even for extremely large population sizes ($N = 512000$). This indicated cases where the restrictions on model structure were too strong and the results in those cases are omitted.

Figure 4.6 shows the execution-time speedup by model restriction based on the maximum dependency distance allowed. The x-axis is the ratio of the number of dependencies in the base runs (no restrictions) compared to the number of dependencies discovered during the restricted runs. The maximum distance allowed during model building is shown as labels for each point in the graph. For example, in figure 4.6a we see that in the original runs of spin glasses of size 16x16, about 50% of the dependencies were neighbor dependencies and more than 80% were dependencies of distance 7 or less. The results strongly show that restricting hBOA model building by maximum dependency distance results in significant speedups of from 4.3–5.2. These speedups are even slightly better than those obtained using the PCM method. We also see that as problem size increases, it is necessary to increase the maximum distance of allowed dependencies. This matches those results shown in section 3.4.7. However, even though it is necessary to allow longer

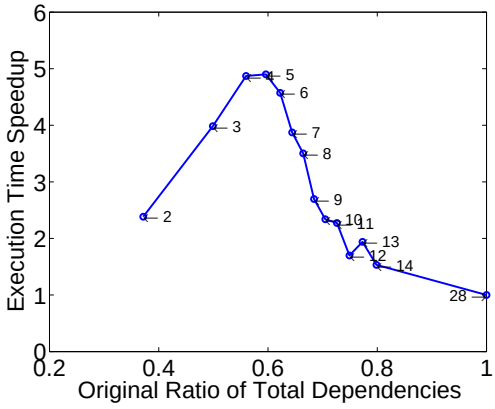


(a) 16×16

(b) 20×20



(c) 24×24



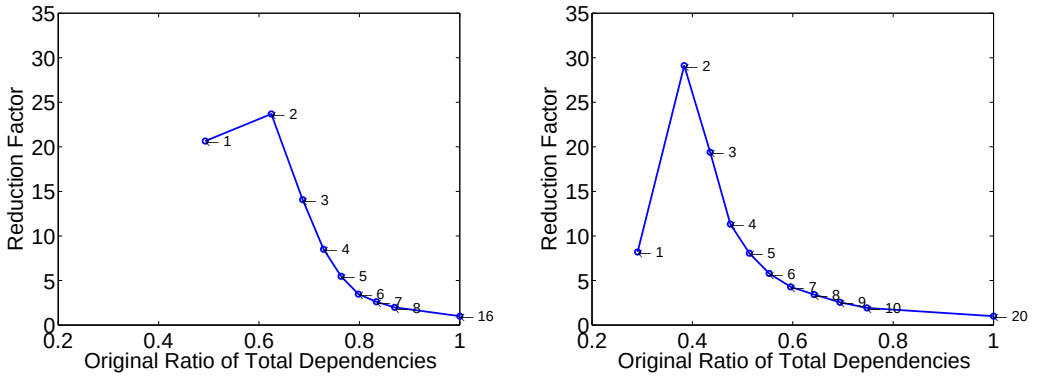
(d) 28×28

Figure 4.6: Execution time speedup obtained by distance-based model restrictions on the 2D Ising spin glass.

dependencies as problem size increases, the maximum execution time speedup seems to remain nearly the same as problem size increases.

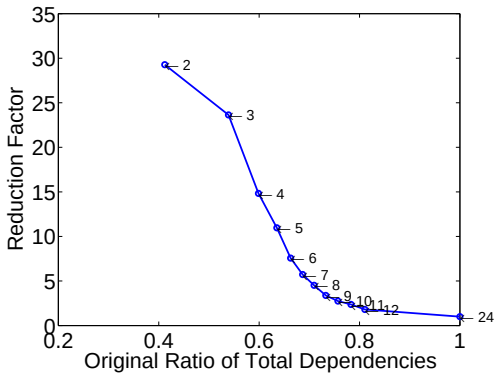
Similarly to the PCM method, we also want to look at the effect on the number of bits examined during model building based on different distance restrictions. Figure 4.7 shows the reduction factor in bits examined during model building when restricting hBOA model building by maximum dependency distance, with the distance restrictions labeled with arrows. The results show that restricting by distance can have a dramatic effect on the number of bits examined, in some cases resulting in a decrease by a factor of 30, with the maximum decrease in bits examining being relatively static over the range of problem sizes considered.

Table 4.2 shows the best speedups, the corresponding maximum distance threshold and q_{min}, and the percentage of total possible dependencies that were considered by hBOA for all problem sizes. This table shows that it is only necessary for hBOA to consider a relatively small percentage of total dependencies discovered during the base runs to solve the problem. The results also show that the maximum speedup

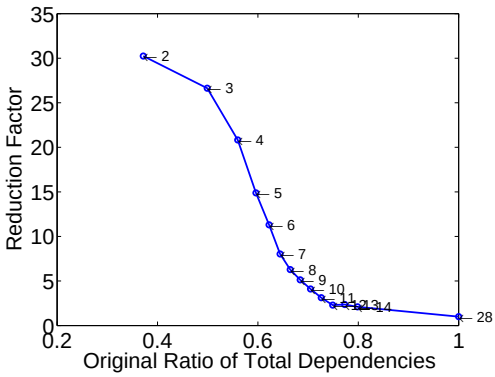


(a) 16×16

(b) 20×20



(c) 24×24



(d) 28×28

Figure 4.7: Reduction factor in number of bits examined during model building based on maximum distance restriction on 2D Ising spin glass instances.

obtained stays nearly the same as problem size increases, indicating that the scalability of this method is strong on 2D Ising spin glass.

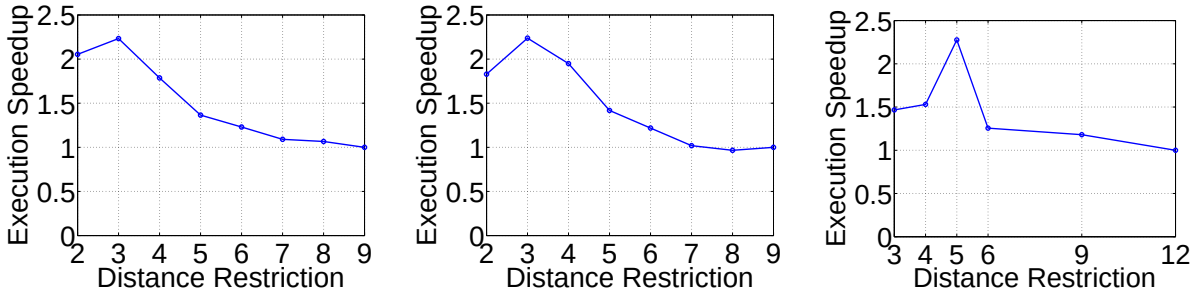
Comparing the PCM method in table 4.1 and the distance-based restrictions in table 4.2, we see that the speedups between the two methods are very similar, with the distance-based bias method being slightly superior. We also see in both cases that hBOA considers a similar percentage of total dependencies in each of the methods.

4.2.4 Experiments with Distance-Based Bias on 3D Ising Spin Glasses

To examine the performance gains from restricting by a distance metric in 3D spin glasses, we looked at three different sizes of instances, 6x6x6, 7x7x7 and 8x8x8. Since spin glasses vary in difficulty depending on the instance, we looked at 1000 different instances for the two smaller sizes. Due to the increased difficulty of the 8x8x8 instances we only examined 100 different instances. For each instance we ran experiments varying the distance restrictions. When the restrictions on model structure became too strong, some of the instances

Size	Execution-time speedup	Max Dist Allowed	q_{min}	% Total Dep.
256 (16×16)	4.2901	2	0.62	4.7%
400 (20×20)	4.9288	3	0.64	6.0%
576 (24×24)	5.2156	3	0.60	4.1%
784 (28×28)	4.9007	5	0.63	7.6%

Table 4.2: The best speedups and their associated distance cutoffs as well as the percentage of total possible dependencies that were considered for 2D Ising spin glass



(a) 6x6x6 (b) 7x7x7 (c) 8x8x8

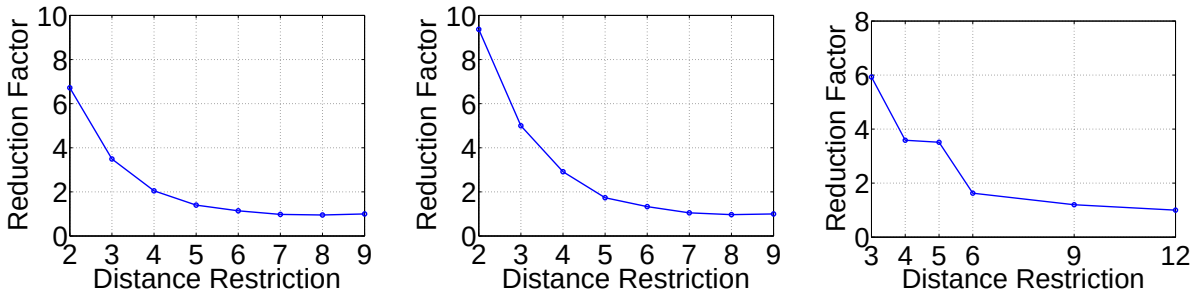
Figure 4.8: Execution time speedups by distance restriction on 3D Ising spin glass instances.

would not converge even for extremely large population sizes; in these cases the results were omitted.

Figure 4.8 shows the execution time speedup with model complexity restricted by various distances for all our three problem sizes. The results show that by restricting the models by distance we can gain speedups of between 2.2 to 1.8. We also see that, as expected, as the problem becomes more difficult we must relax the restrictions to allow higher-order dependencies. Note also that we can restrict too much. Restricting hBOA to only consider those dependencies of distance 2 or less resulted in a loss in performance on the 7x7x7 spin glass. In the same way, restricting below distance five resulted in a loss in performance on the 8x8x8 spin glass.

Another thing of note from these graphs is that we see for the two smaller sizes an almost identical maximum speedup, yet our speedup drops when we move to the 8x8x8. We believe this may be due to only testing 100 instances in the 8x8x8 case which might not have given enough samples to accurately reflect the changes in difficulty between spin glass instances.

Figure 4.9 shows the reduction in bits examined during the entire model-building procedure. We can see from these graphs how we are speeding up our model-building, as we are dramatically reducing the number of bits we must examine in this phase. Note also that regardless of the various problem sizes, the reduction in bits examined is very similar. We also see that this decreases even when our overall execution time is going up. This indicates that at some point other factors start to slow down our execution time more heavily; the model-building bias is too restrictive in these cases.



(a) 6x6x6 (b) 7x7x7 (c) 8x8x8

Figure 4.9: Reduction in bits examined by distance restriction on 3D Ising spin glass instances.

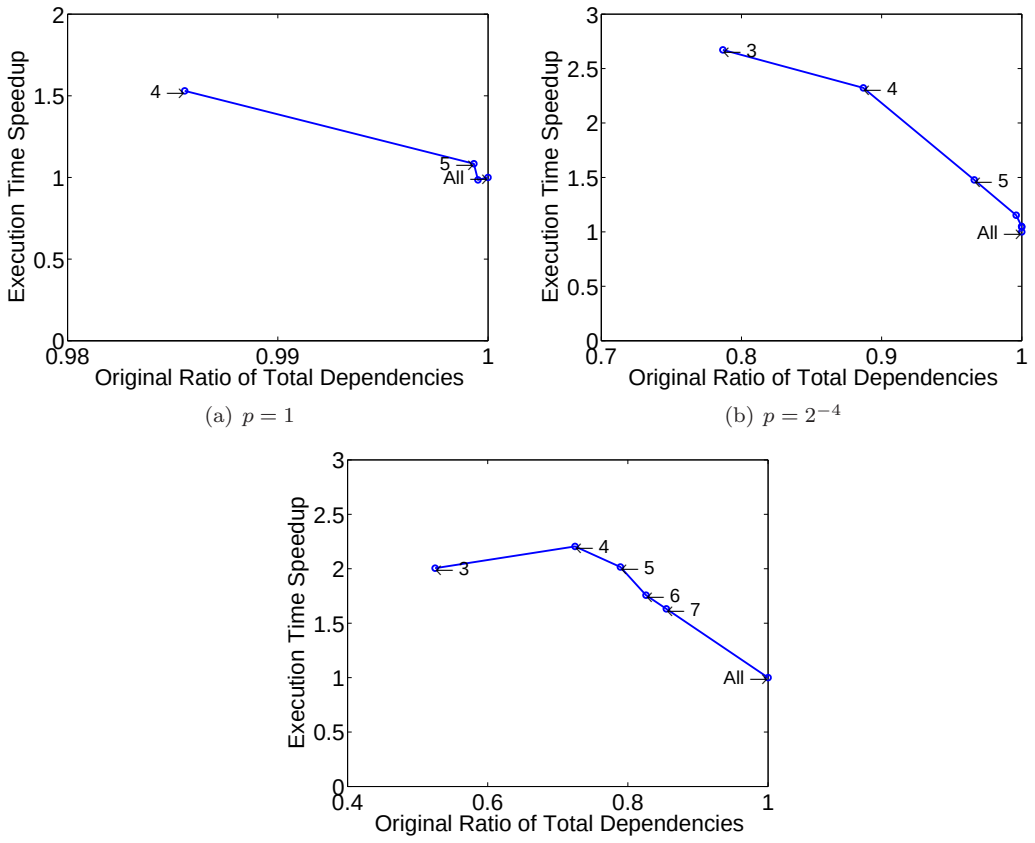
4.2.5 Experiments with Distance-Based Bias on MAXSAT

In the previous sections we saw that restricting model structure by distance leads to significant speedups of hBOA on 2D Ising spin glasses. Can this approach be applied to MAXSAT? This section attempts to answer this question by looking at combined-graph coloring problems encoded as instances of the MAXSAT problem. The distance metric for MAXSAT used in the experiments is described in section 4.1.3.

The distance metric for MAXSAT, like many graph-based problems, is relatively straightforward to implement. What is not so easy is to predict what threshold on distances will lead to optimal or even good performance. This is because while it is easy to agree on the necessity that many of the dependencies that hBOA should consider would be between propositions that share a clause, it isn't so clear what dependencies past this might be necessary. It would most likely be the case that considering only dependencies between propositions in the same clause would be a too severe restriction. However, in order to gain substantially, we would need to restrict as many dependencies as possible.

To examine the trade-off in gains from adding additional restrictions, we considered combined graph-coloring instances of MAXSAT (Gent et al., 1999), with $p = 0$, $p = 2^{-4}$ and $p = 2^{-8}$. For each value of p, 100 random instances were considered where all 100 instances were 5-colorable, and contained 500 propositions and 3100 clauses. This set of instances were used because as p varies, the amount of structure in the problem varies. For example, for $p = 1$, sometimes there was no path between various propositions. For other values of p, the maximum distance between propositions varied from 1 to 9. Given these instances, we can now test various levels of restrictions on maximum distances considered in the underlying problem structure. On some tested problems, small distance restrictions were sometimes too restrictive (for example, restricting to only allowing distance 1 dependencies) and these instances could not be solved even for extremely large population sizes. In these cases the results were omitted.

In Figure 4.10 we show the execution-time speedup of hBOA on MAXSAT when we restrict hBOA model



(a) $p = 1$

(b) $p = 2^{-4}$

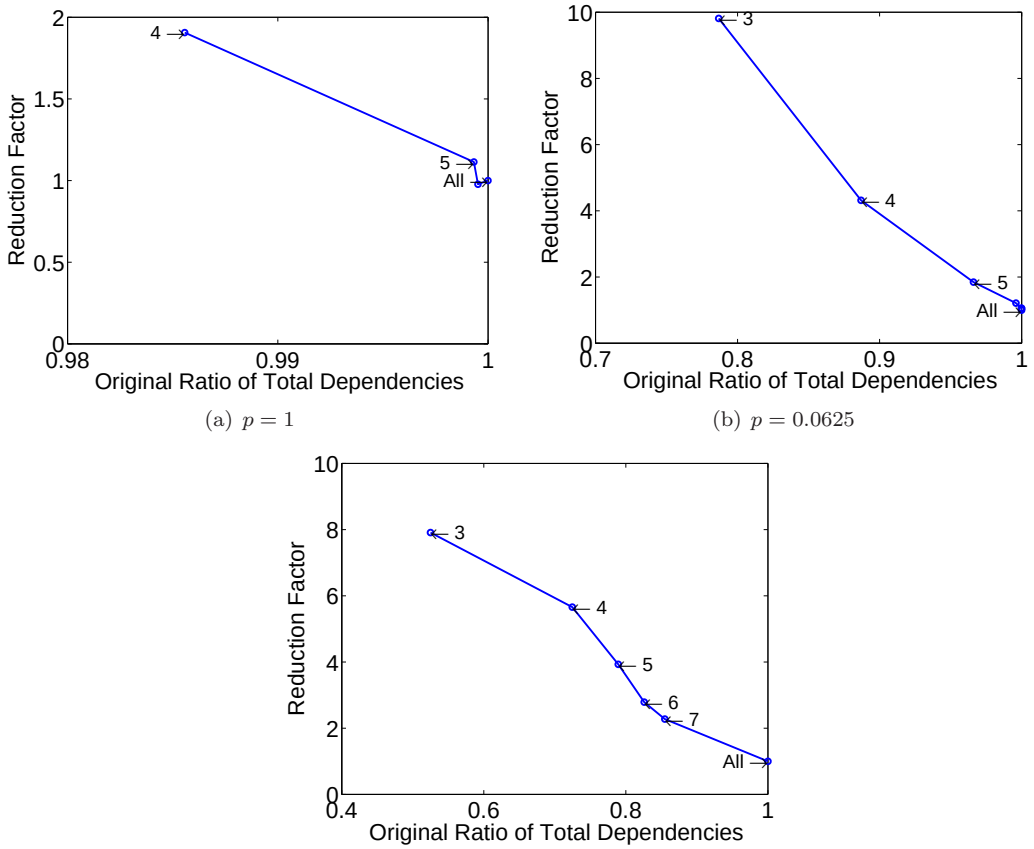
(c) $p = 2^{-4}$

Figure 4.10: Execution time speedup with model restrictions based on the maximum distance allowed on MAXSAT for different values of structure(p).

building to only consider a subset of dependencies based on distance. Similarly to the results with distance restrictions, the horizontal axis stands for the ratio of the number of dependencies with a specific distance disallowed compared to the original unrestricted runs. The maximum distance allowed is shown as a label on each point in the graph.

We see that the maximum speedups obtained varied by the amount of structure in the underlying problem. For $p = 1$, problem instances had very little structure and the maximum speedup obtained was just a bit over 1.5. Even this is somewhat misleading, as all other restrictions led to negligible speedups. On the other hand, as we move to more structure with $p = 2^{-4}$, we see speedups of over 2.5 in the best case, but other restrictions also led to speedups. The most structured problem, with $p = 2^{-8}$, we see that the maximum speedup was 2.2 but many possible restrictions led to improvements.

In general we see that as the mount of structure in the problem increased (by decreasing p), the more possible restrictions (distance thresholds) could be tried that still led to speedups. We must keep in mind



(a) $p = 1$

(b) $p = 0.0625$

(c) $p = 0.0039$

Figure 4.11: Factor by which the number of bits in model building decreases with the model restrictions based on maximum distance on MAXSAT for different values of structure(p).

also that as p increases the number of dependencies at small distances increases rapidly—for example, for $p = 1$ we see that over 98% dependencies were of distance 4 or less, while for $p = 2^{-8}$, only 72% dependencies were at distance 4 or less.

We examine the effects of distance-based restrictions on MAXSAT with regard to the number of bits examined during model building in Figure 4.11. The reduction factor examined in this graph is the reduction factor of total bits examined. We can see that as the amount of structure increases, the reduction factor in bits examined tends to increase also, as the gains are smallest for the case of $p = 1$.

To summarize the results on MAXSAT using distance-based restrictions, Table 4.3 shows the best speedups, the corresponding maximum distance threshold and q_{min} and the percentage of total possible dependencies that were considered by hBOA for all values of p. While these results are not as striking as for 2D Ising spin glasses, nevertheless it was possible to substantially speed up all problem types if the correct restriction was chosen. It is also easy to see why the speedups were not as great as with 2D Ising spin

p	Execution-time speedup	$MaxDistAllowed$	q_{min}	% Total Dep.
$p = 1$	1.53	4	0.99	97.7%
$p = 2^{-4}$	2.67	3	0.79	29.6%
$p = 2^{-8}$	2.20	4	0.83	28.4%

Table 4.3: Best speedups obtained and their associated distance cutoff as well as the percentage of total possible dependencies that were considered for MAXSAT

glasses. A much larger proportion of possible dependencies had to be considered for MAXSAT problems to be solved. Indeed, for $p = 1$, almost all dependencies were considered. Yet even given this, a speedup of almost 50% was obtained.

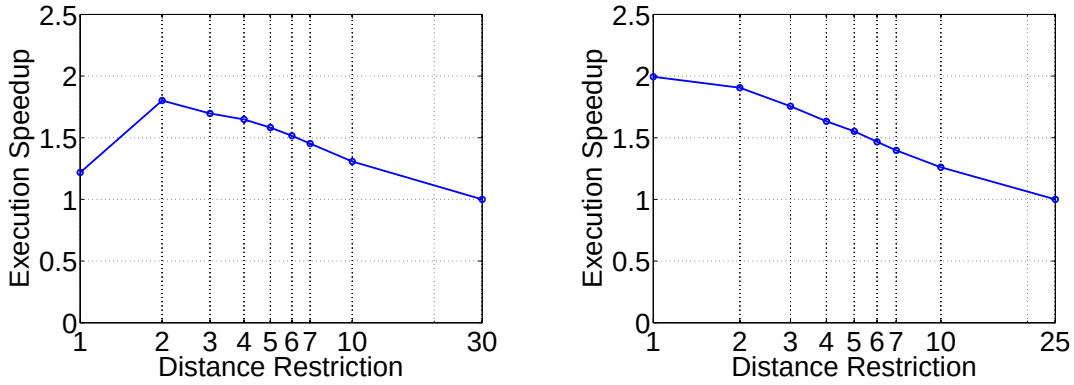
4.2.6 Experiments with Distance-Based Bias on Nearest Neighbor NK Landscapes

Much as we did with MAXSAT, when examining the performance gained from restricting by distance, we want to try considering instances of different amounts of structure. To do this, we considered two different types of instances. The first instances considered were those of nearest neighbor NK landscapes of $n = 92$ with $k = 4$. In these instances, all but one bit of the problem overlapped and so the interaction between subproblems is quite strong. We then contrast those instances with those of $n = 101$ and $k = 4$. These instances only overlap in one bit and so have considerably less interaction between subproblems.

In Figure 4.12a we show the execution time speedup by distance restriction on our nearest neighbor instances with the most overlap. The optimal speedup is seen when we only allow dependencies between bits in the same subproblem or those that are part of overlapping subproblems.

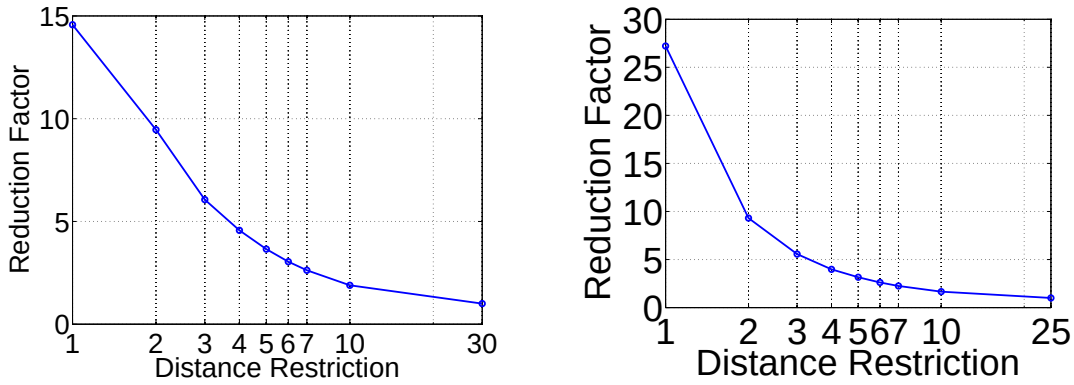
In Figure 4.12b we show the execution time speedup by distance restriction on our nearest neighbor NK landscapes with the least overlap possible. Unlike the instances with more overlap, in this case all restrictions led to speedups, with the heaviest possible distance restriction (to ony consider direct neighbor dependencies, ie: only those dependencies between bits in the same problem) leading to the best speedup. This is a bit counter-intuitive, as these problems overlap and so it would seem beneficial to have included adjacent subproblem dependencies. We would conjecture that this dependency between adjacent subproblems could be taken into account in themodel by the use of the common bit in these problems.

To attempt to examine the direct effects on model building complexity given by different restrictions, Figure 4.13 shows the reduction in bits examined during model building for both types of instances of our nearest neighbor NK landscapes. Both instances show similar behavior with respect to the different restrictions. As restrictions increase, a dramatic reduction in bits examined during model building is show. So while restricting can dramatically reduce overal model building complexity, this does not guarantee us



(a) $n = 92, k = 4, o = 2$ (b) $n = 101, k = 4, o = 1$

Figure 4.12: Execution time speedup by distance restriction on nearest neighbor NK landscape instances.



(a) $n = 92, k = 4, o = 2$ (b) $n = 101, k = 4, o = 1$

Figure 4.13: Reduction in bits examined by distance restriction on nearest neighbor NK landscape instances.

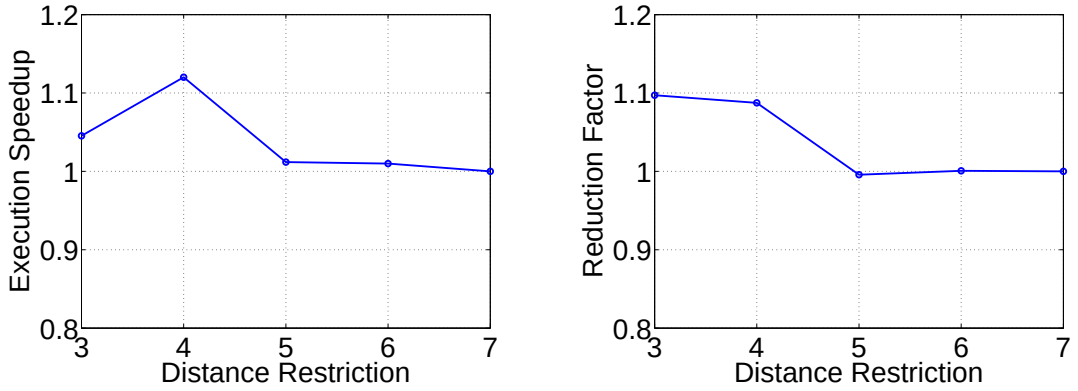
speedup, as other factors besides model building can lead to an increase in execution time for overly harsh restrictions.

4.2.7 Experiments with Distance-Based Bias on Minimum Vertex Cover

For our last examination of the effects of distance-based bias, we want to examine the effects on minimum vertex cover. Figure 4.14a shows the execution speedup by distance restriction on minimum vertex cover instances of $G(n, m)$ graphs, with 300 edges and an average of 4 edges per node.

We see that for MVC, the speedups are much less impressive than our previous examples, showing even in the best case only an improvement of 1.1%. This can be further explained when we consider Figure 4.14b, which shows the reduction in bits examined during model building by distance restriction. We see that unlike all previous experiments, the value does not increase as the restriction increases and there seems to be a transition of some sort occurring between distances 3 and 4.

To attempt to see why this anomaly might be occurring, we looked at the distribution of distances in this problem. Table 4.4 shows the distribution of distances in the 1000 instances we examined. We see that



(a) Execution Speedup

(b) Reduction in bits examined

Figure 4.14: Execution time speedup and reduction in bits examined on a uniformly random MVC instance with 300 edges and an average of 4 edges per node. Note the different pattern from all previous reduction factor graphs examined.

dist	1	2	3	4	5	6	none
total	1200000	8388911	26212460	8883471	137371	580	151960
prob	2.7%	21.3%	79.6%	99.4%	99.7%	99.7%	100%

Table 4.4: Distribution of distances between nodes in all 1000 instances of MVC with 300 nodes and an average of 4 edges per node. The first row is the total number of dependencies at that distance. The second row is the complete total of dependencies at that distance or less. The last row is the cumulative probability of connections equal to or below that columns distance.

by far the most dependencies are of length 3, so this helps to explain why any restriction below 3 leads to the problem being unsolvable. On the other hand, any restriction of 4 or greater will have little effect, as the increase in number of dependencies restricted ends up being quite small. The detrimental effect of cutting off some needed dependencies at these distances outweighs the marginal benefit of cutting so few unnecessary ones.

4.3 Summary

This chapter applied two different approaches to restricting probabilistic models in hBOA based on previous runs. The first approach, creating a PCM and a cutoff threshold to bias model building hBOA, was tested on 2D Ising spin glass. The second approach, using a distance-metric to bias model building in hBOA, was then tested on 2D and 3D Ising spin glass, rADPs, MVC and on morphed graph-coloring instances of MAXSAT. A summary of the key points of this chapter follows:

- For 2D Ising spin glasses, restricting the model building led to a speedup of a factor of 4 to 5.
- Both methods of restricting, PCM and distance-based, led to approximately the same speedups on 2D Ising spin glasses.

- On MAXSAT, depending on the amount of structure in the problem, speedups of 1.5 to 2.5 were observed.
- For 3D Ising spin glasses and nearest neighbor NK landscapes, a speed up of a factor of 2 was observed.
- The results on MVC showed only a marginal improvement due to most of the dependencies being of distance 3.
- The proposed approaches can be adapted to any problem where either (1) problem structure does not vary significantly between different problem instances or (2) one can impose a distance metric on problem variables so that variables located at greater distances are less likely to be connected in the probabilistic model.
- The proposed approaches provide a principled way to control model bias based on previously discovered probabilistic models without requiring the user to manually design such a bias.

Chapter 5

Soft Biasing hBOA Model Building[4]

Chapter 4 showed that it possible to speed up model building in hBOA by restricting hBOA to only consider the most promising dependencies. Chapter 4 also showed that this could be done in several ways, using either information gathered from the absolute bit positions or from distance-based information gathered from the graph-based nature of certain problems.

However, the methods proposed, distance-based bias and PCM, come with the downside that they required the user to specify a threshold parameter to determine the level of restriction used. If this threshold was set incorrectly such that hBOA was not allowed to consider important dependencies, it was possible for hBOA to take longer to solve the problem or be unable to solve the problem at all. One way to avoid this problem would be to use "soft" bias. Rather than restricting hBOA to only consider a certain set of dependencies, we instead bias hBOA to consider some dependencies more strongly than others. In this way, hBOA will never be completely dissallowed from considering some dependencies that might be necessary to solve the problem.

This chapter describes a technique to perform a "soft" bias of hBOA model building by modifying the hBOA model building metric itself to more strongly prefer promising dependencies that were often discovered in previous runs of hBOA on similar problems. To accomplish this, first data from previous runs of the EDA is gathered in a split probability matrix (SPM) and then this data is used to bias the structural priors used by the model building metric. An additional parameter, κ, is required to determine the strength of these priors. This is usually significant problem with black-box optimization, for if a parameter is set incorrectly this can lead to a loss of performance or even an inability to solve the problem. However, the results will show that the same parameter setting leads to speedups in all tested problems.

The chapter is organized as follows. Section 4.1 starts by explaining how hBOA uses the prior probability of network structure in its Bayesian metric and then describes the split probability matrix (SPM) and how it is used to bias model building in hBOA. Section 5.2 presents the test problems and experimental results.

[4]This chapter is based in part on work previously published in Hauschild, M. & Pelikan, M. (2009) Intelligent bias of network structures in the hierarchical Boa. *ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2009)*, 413–420

Finally, section 5.3 summarizes the chapter.

5.1 Soft Bias in Model Building

In section 4 we performed a "hard" bias of hBOA model building by strictly disallowing certain dependencies based on previous runs. This had a direct connection to the speed of hBOA model building: The more dependencies excluded, the less dependencies hBOA had to consider during each model building step.

With "soft" bias, this excluding of dependencies does not happen. hBOA will still be examining the same number of dependencies each step. Instead, performance should be gained by promoting important dependencies found in models in previous runs on similar problems. This is done by modifying the prior probability of network structure in the metric used to judge the quality of our Bayesian networks in hBOA model building. If done correctly, this should result in the metric giving a bonus to those dependencies often found in prior runs of hBOA and penalizing those dependencies that are rare in the prior runs. By making the model building more quickly discover the important dependencies (those found more often in prior runs), this should lead to more accurate models even with a smaller population size. In addition, by penalizing dependencies that were not discovered often in the trial runs, we would expect a reduction in the number of bad or spurious dependencies discovered.

In the next section we will discuss how hBOA incorporates the prior probability of a network structure and how we will modify this prior probability to perform the "soft" bias.

5.1.1 Structural Priors in Bayesian Metrics

The prior probability of network structure is given in Bayesian-Dirichlet metrics and other Bayesian metrics by using structural priors. The basic form of Bayesian-Dirichlet metric for network B and data set D is

$$p(B|D,\xi) = \frac{p(B|\xi)p(D|B,\xi)}{p(D|\xi)}. \tag{5.1}$$

where ξ denotes the background knowledge. the prior probability of a network structure is represented by the term $p(B|\xi)$, which is the important term for this section. Since the Bayesian-Dirichlet metric often generates overly complex models (Friedman and Goldszmidt, 1999; Pelikan, 2005), this prior probability of network structure is set to penalize overly commplex networks. In hBOA, the bias toward simple models is typically introduced as (Friedman and Goldszmidt, 1999; Pelikan, 2005)

$$p(B|\xi) = c2^{-0.5(\sum_i |L_i|)log_2 N}, \tag{5.2}$$

where N is the size of the population, $\sum_i |L_i|$ is the total number of leaves in all decision trees representing parameters of the local conditional distributions, and c is a normalization constant so that the prior probabilities of all network structures sum to 1.

What is important to note is that this prior probability of network structure $p(B)$ must be included in standard hBOA as we have no information about previous networks, and such must assume a uniform prior network or we are lead to overly complex models. However, if we have prior information about models used to solve similar problems, it should be advantageous to use this information to add a different complexity penalty based on this knowledge. Essentially, instead of using this prior probability of network structure to prefer simpler models, we will be modifying it to prefer the networks generated during our prior runs.

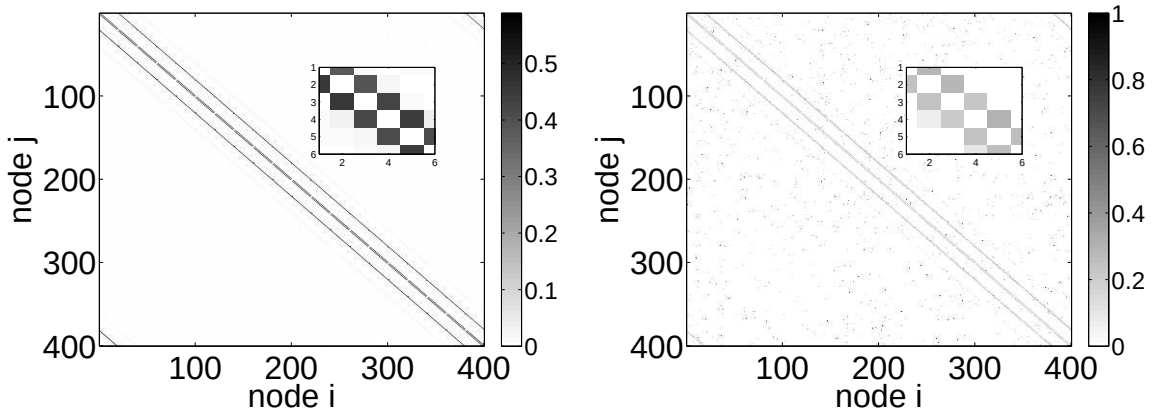
5.1.2 Split Probability Matrix

Before we can bias the prior probabilities of network structure in future runs, we must first gather the statisticis of prior network structure. This is done with the *split probability matrix* (SPM), which is built from the Bayesian networks generated from previous runs of hBOA on similar problems. The SPM stores the probability of network splits on different variables in the prior runs and will be used to modify the prior probability of network structure in the hBOA model building metric.

The SPM is a four-dimensional matrix of size $n \times n \times d \times e$ where n is the number of variables (problem size), d is the maximum number of splits in any decision graph for which we gather the data for and e is the number of generations from which we collect data. Let us denote the matrix by S and the elements of S by $S_{i,j,k,g}$ where $i, j \in \{1 \dots n\}$, $k \in \{1 \dots d\}$ and $g \in \{1 \dots e\}$. The value $S_{i,j,k,g}$ is defined as, during the generation g of the sample data, the conditional probability of a kth split on the variable X_j in the decision graph for X_i given that there were $k-1$ such splits performed already; if there are no models with at least $k-1$ such splits, we define $S_{i,j,k,g} = 0$.

Before computing S, we first collect all models we want to use to bias model building in future hBOA runs. Then we examine all the runs collected and set an upper threshold e for the number of generates to store in the SPM. This threshold e needs to be set high enough so that the sample size of runs used to gather data is sufficient to lead to good results. In this work e will be set to the number of generations that at least 90% of the runs reached. So for example, if 100 runs of hBOA are used to generate the SPM and 90% of the runs contained at least 19 generations, e would be set to 19.

Then all probabilistic models of generatation e or less are examined, incrementing the element $S_{i,j,k,g}$ if there were at least k splits on X_j in the decision tree for X_i during generation g. Once this is complete, for all $k \neq 1$, we divide all elements $S_{i,j,k}$ by $S_{i,j,k-1}$. Lastly, we divide all elements S_{ij1} by the total number



(a) No splits

(b) One split

Figure 5.1: First two splits of the SPM gathered from the first generation of 90 instances of 20×20 2D Ising spin glasses. A closeup of an area with strong interaction is also included.

of probabilistic models that were parsed. Note that the SPM gathers information up to some maximum generation e. However, it is possible when using the SPM to bias model building that hBOA might exceed that generation, leading us to not having an equivalent generation of SPM data. In this case the data from the SPM corresponding to the last generation e is used to bias all subsequent hBOA model building.

Since it is possible that runs of hBOA that use the SPM could exceed the number of generations e stored in the SPM, if it is necessary to use the SPM at that point, the prior probability of network structure from generation e is used.

Let us give a simple example to illustrate this. Suppose that during generation g of the trial runs of hBOA, for a particular $i \neq j$, 30 of the 100 models examined had at least one split on variable X_j in the decision tree for X_i, and 15 of those models had at least two such splits. Then $S_{i,j,1,g} = 30/100 = 0.3$ and $S_{i,j,2,g} = 15/30 = 0.2$.

$S_{i,j,1,g} = 40/100 = 0.4$ and $S_{i,j,2,g} = 10/40 = 0.25$.

To examine what the SPM looks like on an actual problem instance, figure 5.1 shows a SPM gathered from the first generation of 90 instances of 20×20 2D Ising spin glasses. Figure 5.1a shows the probability that there was at least one split between two variables in a decision tree during the first generation. Figure 5.1b then shows the conditional probability of a second split between these two variables given that there is already a split between them. As expected, there is much less of a probability of a second split on each variable. However, it is clear that the strongest expected dependencies in 2D Ising spin glass, the neighbor dependencies, are strongly represented in both these figures.

5.1.3 SPM-Based Model-Building Bias

Now that the SPM has gathered the probability of prior network structure in our sample runs, we now need to use it to modify the metric itself. In order to tune the effects of prior bias, we will use a parameter κ. When κ is set to 0, prior model information will be ignored for learning future models, with the larger the value of κ, the greater the influence the prior models will have on future models. While it is not clear exactly what the optimal value of κ is, our results will show that $\kappa = 1$ leads to improvements on all tested problems.

In the Bayesian-Dirichlet metric, the prior probability of network structure is set to the product of prior probabilities for all the decision trees in the network:

$$p(B|\xi) = \prod_{i=1}^{n} p(T_i). \tag{5.3}$$

For each decision tree T_i, $p(T_i)$ is set to the following product:

$$p(T_i) = \prod_{j \neq i} q_{i,j,k(i,j)}^{\kappa}, \tag{5.4}$$

where $q_{i,j,k(i,j)}$ denotes the probability that there are at least $k(i,j)$ splits on X_j in decision trees for X_i in the set of models used to bias model building, $k(i,j)$ denotes the number of splits on X_j in T_i.

Since hBOA builds its models greedily one split at a time, the conditional probabilities stored in SPM can be used to compute the gains in the log-likelihood of the model after any admissible split. More specifically, let's consider evaluation of the split on X_j in the tree T_i during generation g, let's denote the network before performing this split by B and the network after performing this split by B', and let's assume that there were $k(i,j)-1$ splits on X_j in T_i before this split. Then, the gain in the log-likelihood corresponding to this split is defined as

$$\delta_{i,j} = \log_2 p(D|B',\xi) - \log_2 p(D|B,\xi) + \kappa \log_2 S_{i,j,k(i,j),g}.$$

For comparison, hBOA without considering prior information only uses a complexity term which is defined as

$$\delta_{i,j} = \log_2 p(D|B',\xi) - \log_2 p(D|B,\xi) - 0.5 log_2 N.$$

As we discussed in section 5.1.2, if the SPM is not defined for a specific generation, the last generation defined by the SPM (e) is used.

5.2 Experiments

To examine the effect of soft bias, we consider two test problems, trap-5 and 2D Ising spin glasses. In this section we cover our experiments on these two test problems using the SPM to bias hBOA model building. First the parameter settings and the experimental setup are discussed. The results are then presented.

5.2.1 Parameter Settings

For trap-5, bisection (Sastry, 2001b; Pelikan, 2005) was used to determine the minimum population size to ensure convergence to the global optimum in 10 out of 10 independent runs. For more reliable results, 10 independent bisection trials were run for each problem size, for a total of 100 runs for each problem size.

For each spin glass instance, bisection was used to determine the minimum population size to ensure convergence to the global optimum in 5 out of 5 independent runs, with the results averaged over all problem instances.

The number of generations was upper bounded according to preliminary experiments and hBOA scalability theory (Pelikan et al., 2002b) by $n/4$ where n is the number of bits in the problem. The results were obtained by averaging over all 100 runs. Each run of hBOA is terminated when the global optimum has been found (success) or when the upper bound on the number of generations has been reached without discovering the global optimum (failure). In those problems using SPM bias, the maximum generational SPM data stored is set to the number of generations reached by at least 90% of the sample runs.

5.2.2 SPM Model Bias on Trap-5

For trap-5, 10 bisection runs of 10 runs each were first performed with standard model building bias toward simple networks used in the original hBOA. The models discovered in these 100 runs were then used to compute the SPM matrices, which were consequently used to bias model building in the next 10 rounds of bisection.

In figure 5.2a we show the average execution-time speedup obtained with the SPM bias over all runs using $\kappa = 1$. Again, this is the total execution time required to solve the problem, not just the time necessary for model building. We see that a speedup by a factor of 5 was quite common across a broad variety of problem sizes, meaning that hBOA with these restrictions was solving the problems about 5 times faster. The maximum speedup achieved was a factor of 6, with a minimum speedup on the smallest problem size being 3.5. In Figure 5.2b we examine the speedup in terms of the number of evaluations. Again in this case, we see speedups across the entire range of problems. In some cases, a speedup of 3 was achieved, but even

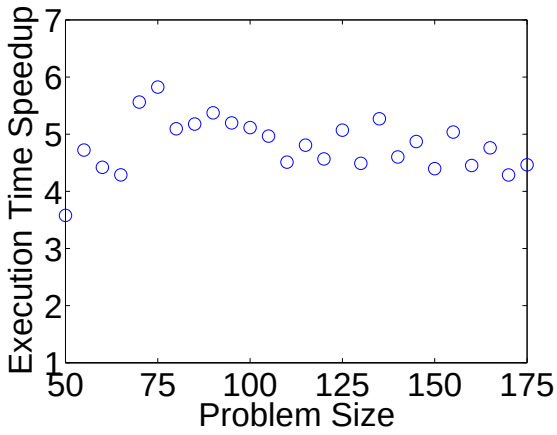
in the worst case SPM bias halved the number of evaluations.

It should be immediately clear from the previous paragraph that the overall execution time was improved using SPM bias, what was the effect on model building speed itself? This is not as straightforward to examine as the number of evaluations, but can be done by considering one of the most common operation done in model building. The number of bits examined to update model parameters is one of the primary factors affecting time complexity of model building. To help isolate the affect of SPM bias on model building, we examined the average factor of decrease in these bits being examined. Figure 5.2c shows the average factor of decrease in bits examined during model building with respect to problem size. We can see that for the smallest problem size, $n = 50$, the number of bits examined during model building was reduced by a factor of about 14. This reduction factor of bits examined continues to increase almost linearly as the problem size increases. For the largest problem size considered, that is, $n = 175$, the reduction factor is as high as 67. This constantly increasing performance is in marked contrast to the speedup in evaluations, where the reduction in evaluations was fairly constant. This shows that the main contributor to the speedup in overall execution time when using SPM bias is due to the reduction in work during model building.

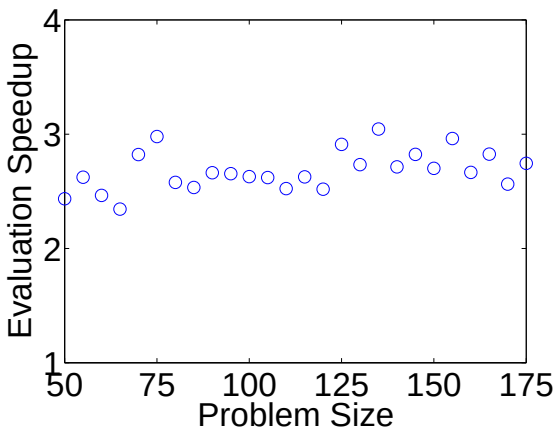
Another factor to consider is the effects of our parameter κ. It is possible that by varying κ, we could increase the performance gains obtained with SPM bias, giving us a way to tune SPM bias for better performance. On the other hand, if SPM bias is too sensitive to changes in κ, it is possible that we have added an additional parameter that must be set correctly in order for hBOA to solve the problem. Figure 5.3 shows the effects of κ on hBOA performance in terms of the overall execution time in seconds, the number of evaluations and the number of bits examined on trap-5 of size $n = 100$. We see in Figure 5.3a that in general, the execution time decreases as κ is decreased. On the other hand, the performance is only marginally improved when $\kappa < 1$ compared to $\kappa = 1$. The effects of κ in terms of the number of evaluations and the number of bits examined in model building are quite similar as those measured in terms of the overall execution time (see Figure 5.3b and Figure 5.3c). We can see that while increasing κ does indeed improve performance, the performance is quite good when κ is close to 1. On the other hand, setting $\kappa = 0$ resulted in very poor performance and the results were omitted. This is to be expected as this is equivalent to completely removing the complexity penalty in hBOA, which would result in overly complex models (Friedman and Goldszmidt, 1999; Pelikan, 2005).

5.2.3 SPM Model Bias on 2D Ising Spin Glass

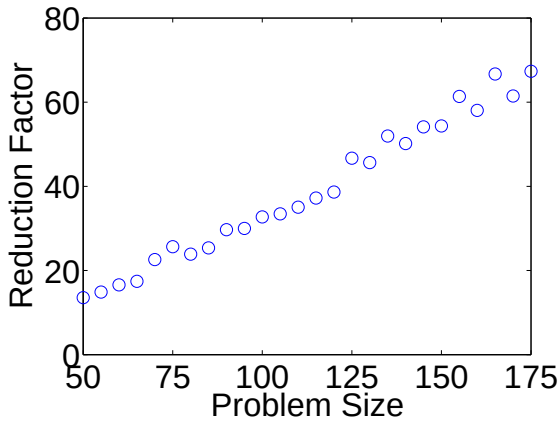
We saw in the previous section that we were able to gain significant speedups of hBOA on trap-5 by using SPM bias. This was due to the SPM bias leading the model building towards more promising structures.



(a) Execution Speedup



(b) Evaluation Speedup



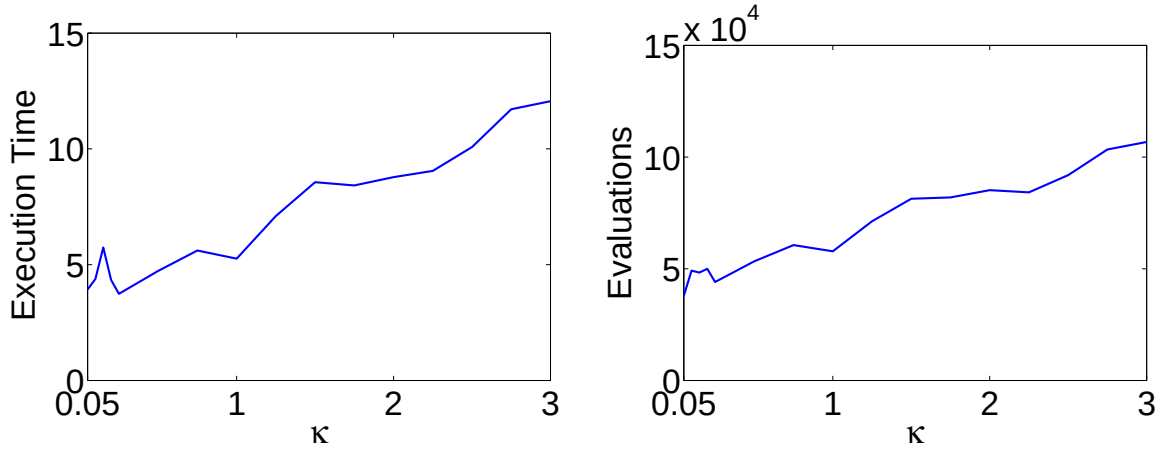
(c) Reduction in Bits Examined

Figure 5.2: Speedups obtained using SPM bias for $\kappa = 1$ on trap-5 in terms of overall execution time, number of evaluations, and number of bits examined in model building.

However, as we know from chapter 3, it is relatively easy to define a perfect model for trap-5. On the other hand, for 2D Ising spin glasses, even defining an adequate model can be extremely difficult (Mühlenbein et al., 1998; Hauschild et al., 2007). In this section we will examine whether SPM bias is effective even in a case where defining an exact model is difficult.

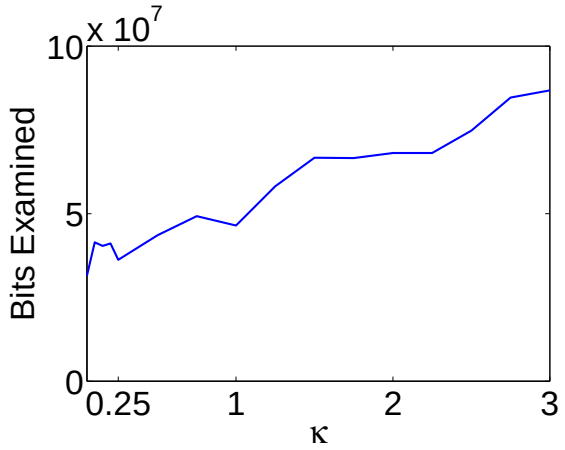
In order to explore the effects of SPM bias on 2D Ising spin glasses, we examined three different problem sizes: 16×16 (256 spins), 20×20 (400 spins) and 24×24 (576 spins). For each of these problem sizes, we examined 100 random instances with 10-fold crossvalidation, similar to how we examined the effects of PCM in chapter 4. This ensures that the validation was done on different instances than those used to learn the SPM.

In Table 5.1 we show the effect of SPM bias for $\kappa = 1$. For the problem size of 16×16, the smallest problem size, a speedup of about 1.16 was obtained on hBOA performance. As the problem size increased,



(a) Execution Time

(b) Evaluations



(c) Bits Examined

Figure 5.3: The effects of changing κ on the execution time, number of evaluations, and number of bits examined in model building using SPM bias on trap-5 for size $n = 100$.

so did the execution time speedup, with a speedup of 1.56 being observed for the problems of size 24×24. The factor of decrease in bits examined during model building also decreased, with the benefits growing as problem size increased. On the other hand, unlike for trap-5, we actually see a small increase in the number of evaluations. This increase does drop off as problem size increases, but this is still in contrast to the results for trap-5. For trap-5 the performance gained by hBOA was both a factor of the reduction in model building complexity and the number of evaluations. For 2D Ising spin glasses, the speedups obtained using SPM bias seem to be the result of reducing the complexity of model building.

We now examine the effects of modifying κ on the speedups obtained using SPM bias. In figure 5.4 we show the execution times in seconds for hBOA with the SPM bias for a range of values of κ on the 2D Ising spin glass. In trap-5 we saw that decreasing κ below 1 led to better performance (except in the most

size	Exec. speedup	Eval. Speedup	Bits Exam.
16×16	1.16	0.87	1.5
20×20	1.42	0.96	1.84
24×24	1.56	0.98	2.03

Table 5.1: Speedups obtained using SPM bias for $\kappa = 1$ on 2D spin glasses in terms of overall execution time, number of evaluations, and number of bits examined in model building.

size	κ	Exec. speedup	Eval. Speedup	Bits Exam.
16×16	0.75	1.24	0.96	1.66
20×20	1.25	1.44	0.94	1.85
24×24	1	1.56	0.98	2.03

Table 5.2: The optimal value of κ that led to the maximum execution-time speedup of hBOA using SPM bias on the 2D Ising spin glass of 3 sizes, and the accompanying speedups in evaluations and number of bits examined in model building.

extreme case of $\kappa = 0$. However, for 2D Ising spin glasses, $\kappa < 1$ led to much worse performance in almost all cases. Also, as κ increases past 1, the execution times continue to increase, with the area of maximum speedup occurring when κ is approximately 1.

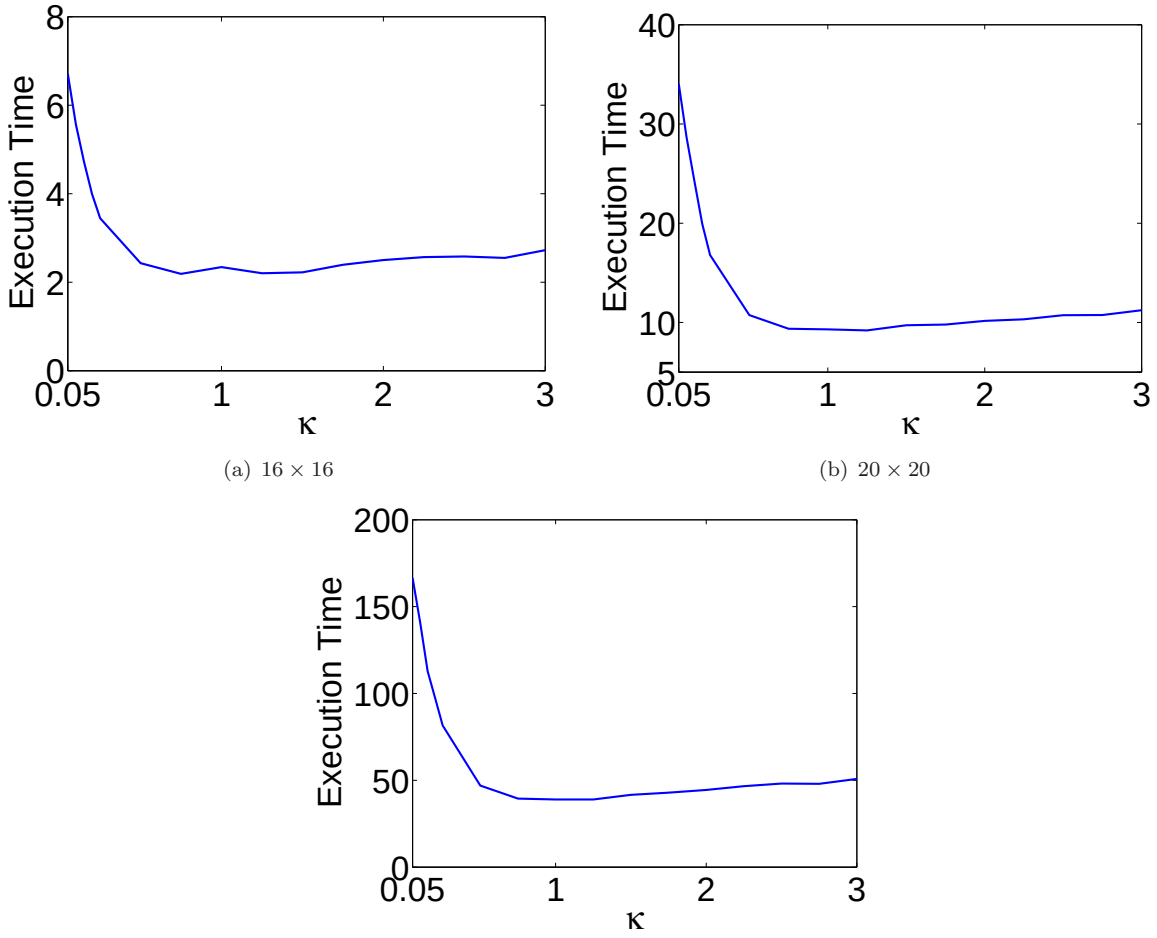
Of interest to us is the exact values of κ that led to the maximum speedup of execution time. Table 5.2 shows these results, as well as the accompanying speedups of the number of evaluations and the factor of decrease in bits examined during model building. We see that We see for all problem sizes examined, having a κ close to one led to the maximum speedup.

We also look at the effects onf κ on the number of evaluations for 2D spin glass in figure 5.4. We can see that the minimum number of evaluations necessary to solve all three problem sizes occurred when $\kappa \approx 0.5$. The number of evaluations gets much worse as κ approaches 0 and gradually gets worse as κ grows beyond the optimal value. This same pattern is observed in the results for the factor of decrease in the number of bits examined during model building in figure 5.6. For all κ considered, the best performance was gained when $\kappa \in [0.5, 1]$.

5.3 Summary

This chapter proposed an approach to bias the model building metric in hBOA based on the probabilistic models generated from previous runs on similar problems. The bias was introduced through structural priors of Bayesian metrics using the split probability matrix (SPM). A summary of the key points of this chapter follows:

- The split probability matrix (SPM) was proposed as a way to bias hBOA model building through structural priors of Bayesian metrics.



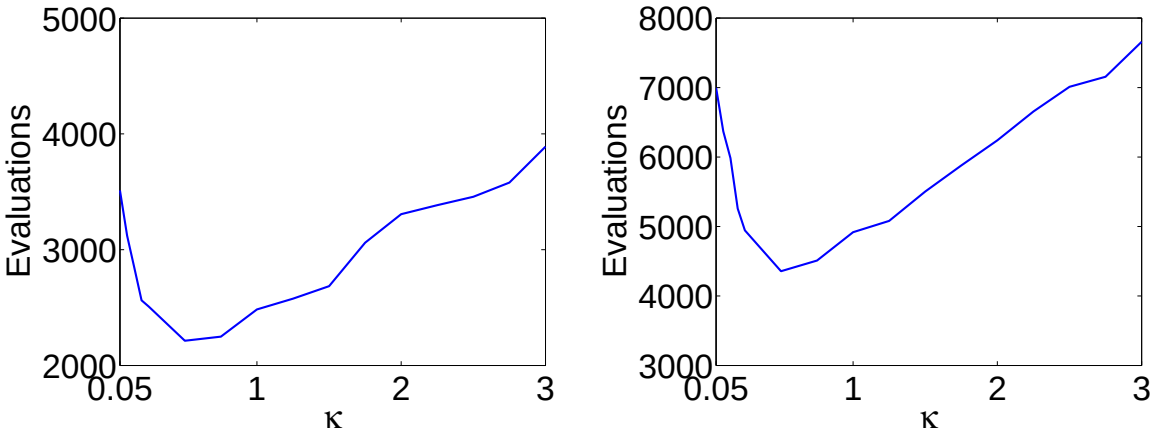
(a) 16×16

(b) 20×20

(c) 24×24

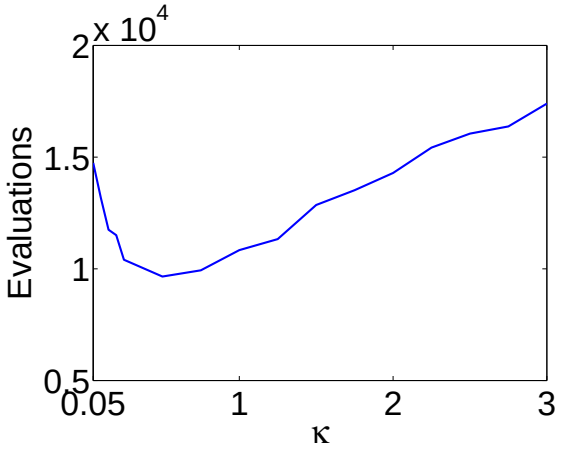
Figure 5.4: The effects of changing κ on the execution time of hBOA using SPM bias on 2D Ising spin glasses.

- On trap-5, SPM bias led to substantial speedups in regards to the number of evaluations and bits examined during model building. For overall execution time, speedups of about 3.5 to 6 were obtained.
- On 2D Ising spin glass the SPM bias led to a slight increase in the number of evaluations, but in terms of the number of bits examined during model building as well as the overall execution time substantial speedups were obtained. Execution times were reduced by a factor of 1.5 or more, with the speedups increasing with problem size.
- While the method in this chapter requires a parameter κ, the results show that $\kappa = 1$ led to execution-time speedups in all tested problems. More importantly, those speedups were close to the speedups obtained with the optimal value of κ. This is important as this method, unlike all previous methods, relied on parameters that if set incorrectly could either slow down performance or make the problem



(a) 16×16

(b) 20×20

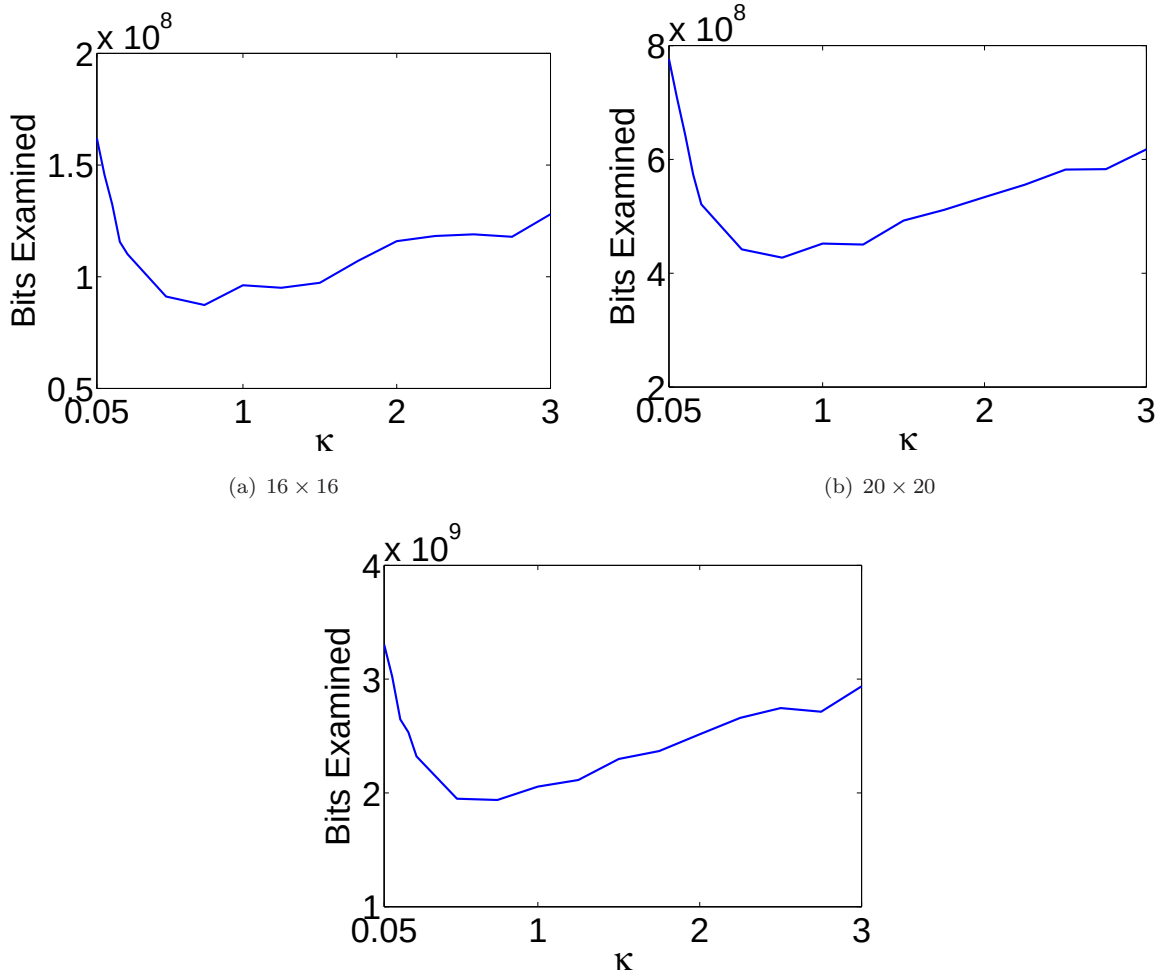


(c) 24×24

Figure 5.5: The effects of changing κ on the number of evaluations for hBOA using SPM bias on 2D Ising spin glasses.

intractible.

- While this chapter only considered two test problems, the proposed methods should work on any problem that contains significant structural similarities between problem instances. This structural similarity could also take into account a distance metric, such as in chapter 4.



(a) 16×16

(b) 20×20

(c) 24×24

Figure 5.6: The effects of changing κ on the number of bits examined in model building for hBOA using SPM bias on 2D Ising spin glasses.

Chapter 6

Network Crossover[5]

As we have discussed previously in this thesis, practitioners often have prior information about the problem they are trying to solve. This type of information could range from a simple understanding of which variables tend to depend on others, to more explicit information such as that given when examining graph based problems. In addition, we have shown in chapter 4 and chapter 5 that it is also possible to learn valuable problem information from examining models generated from trial runs of an EDA.

In chapter 4 and chapter 5 this information was used to speed up model building on similar problems. We saw that this led to speedups on a broad range of test problems. However, this came at a cost. If this information was used incorrectly, for example by overly biasing the models towards only the very strongest dependencies discovered, it was possible that the EDAs performance could be hurt or it could even be unable to solve the problem.

In this chapter, we will attempt to exploit this information by modifying the crossover operator in a simple genetic algorithm to respect the underlying problem structure. This *network crossover* will use either a user-specified network or a network discovered by trial runs of an EDA to determine which variables will be exchanged during crossover. As long as this network represents good information about the problem structure, it should be superior to the standard GA crossover operators as it will respect the linkages between important variables. In addition, since it will not require an expensive model building phase, it might be able to outperform EDAs. We will then examine the effects of this network crossover operator against uniform and two-point crossover on several test problems known to be hard for standard evolutionary algorithms. The proposed GA with network crossover is then tested against hBOA.

The chapter is organized as follows. Section 6.1 briefly outlines the simple genetic algorithm (GA). Section 6.2 describes the network crossover operator used in this chapter. Section 6.3 outlines the test problems used in this chapter. Section 6.4 then presents the experimental results. Finally, section 6.5 summarizes the chapter.

[5]This chapter is based in part on work previously published in Hauschild, M. & Pelikan, M. (2010) Network crossover performance on NK landscapes and deceptive problems. *ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2010)*, 713–720 and Hauschild, M. & Pelikan, M. (2010) Performance of Network Crossover on NK Landscapes and Spin Glasses. *Proceedings of the Parallel Problem Solving from Nature Conference (PPSN-2010)*, *2* 462–471

6.1 Genetic Algorithm

The genetic algorithm(GA) (Holland, 1975; Goldberg, 1989) evolves a population of candidate solutions typically represented by binary strings of fixed length. The starting population is generated at random according to a uniform distribution over all binary strings. Each iteration starts by selecting promising solutions from the current population; in this work we use binary tournament selection without replacement. New solutions are created by applying variation operators to the population of selected solutions. These new candidate solutions are then incorporated into the population using a replacement operator. The run is terminated when some termination criteria has been met. In this work, we terminate each run either when the global optimum has been found or when a maximum number of iterations have been reached.

The most common variation operators are crossover and bit-flip mutation (Goldberg, 1989). In this work we use three different crossover operators, both that take two candidate solutions and generate two new candidate solutions. The first, network crossover, is explained in section 6.2. Uniform and two-point crossover are also used.

6.2 Network Crossover

The standard uniform and one-point crossover operators are effective for many problems (Goldberg, 1989). However, if the underlying problem has tightly linked variables, these operators can often too heavily disrupt candidate solutions (Thierens, 1999). The main idea behind network crossover is to solve this problem by developing a special crossover that attempts to minimize the disruption of tightly linked variables.

For example, any two-parent crossover works by selecting variables from each parent to develop new candidate solutions. In the uniform crossover operator, each bit is used with a 50% probability to create a new candidate solution. More formally, for each crossover operation on a solution of length n, a crossover mask of length n bits is generated, such that if the k^{th} bit is a 1, the child's k^{th} bit is set to the first parent, otherwise the child's k^{th} bit is set to the second parents k^{th} bit. In the case of uniform crossover, the probability of a particular bit being one in any position is 50%.

In this section, we will develop a crossover that attempts to maximize the probability that we take strongly linked variables from the same parent together. If done correctly, this should minimize the disruption of the crossover. To do this, we first require a user-specified matrix G of size $n \times n$ that specifies the strongest dependencies between variables. While this is information that is not normally given when using black box optimization, this also does not require in-depth knowledge of the exact strength of the interactions. Instead, it should simply specify the very strongest of dependencies. This network is easy to develop for graph-based

problems, as this G should correspond directly to the underlying graph in these problems. However, not all problems have an implicit graph structure. In those cases, it should be possible to use trial runs of an EDA to learn the strongest connections between variables and use this to generate our G.

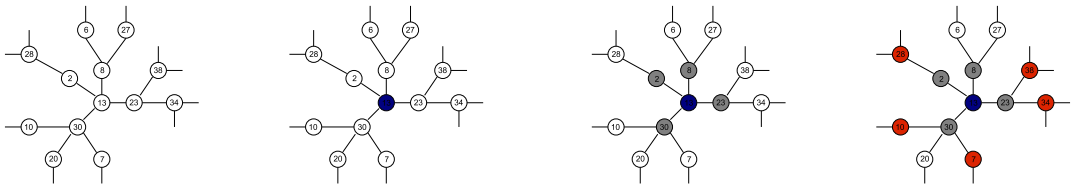
Note also that this network G can be quite sparse or uniform. If the network itself is sparse, indicating that little information is known about the strength of interactions, then our crossover operator should act like uniform crossover except in those cases where we do have interactions. In the case of a uniform matrix, where all bits in the underlying problem are specified as having important connections, then we would again expect it to act much like uniform crossover. This is to be expected, as in both these cases the network G is giving us relatively little information about the underlying problem structure.

Once this network G is given, the network crossover mask can be generated. To start a random bit position in the solution string is added to the crossover mask by setting $m_i = 1$. A randomized breadth-first search is then used on the network to set bits in m to 1 until m reaches the correct size. If the breadth-first search ends early due to the network having a dead end, the process is repeated from another random bit position.

More formally, given a $n \times n$ network G and an empty crossover mask vector M:

1. While $|M| \leq n/2$
   (a) Select a random starting variable in the problem p.
   (b) Add p to set C.
   (c) Repeat
      i. Add to C the set of all x, such that $G_{i \in M, x} = 1$
      ii. While $C \neq \emptyset$ AND $|M| \leq n/2$
         A. Remove an element i randomly from C.
         B. Add i to M.
   (d) Until $C \equiv \emptyset$ or $|M| > n/2$

We can see an example of a crossover mask M being created in figure 6.1. First we start with an empty crossover mask and a network G as in figure 6.1a. Then we select a random starting point p, which in this example is node variable 13, in figure 6.1b. Then we add all the neighbors of variable 13 to our set C and add each one in turn to our crossover mask, ending with a crossover mask containing the variables colored in figure 6.1c. We then add all neighbors of the newly added members of the crossover mask to our set C



(a) Blank crossover mask (b) Initial random variable (c) Expansion (d) Final stage

Figure 6.1: Building a crossover mask M from a network G in 4 steps.

and attempt to add as many of these as we can to our network crossover mask. However, in this case we reach our limit and can only add five additional members, as show in figure 6.1d.

Using this crossover, variables connected strongly in G should most likely come from the same parent when creating new children, resulting in a crossover that strongly respects the underlying dependencies in our problem.

Requiring the network G is a significant additional step. However, as we discussed earlier, it is not necessary for the practitioner to have total problem understanding. The network G must only contain the strongest linkages, and for many problems this is quite easy to specify. For graph-based problems, G can be constructed by connecting those variables that have an edge between them. In the NK landscape instances used in this chapter, any neighbor bits are connected in G, with bits that are not neighbors remaining unconnected in G. For Trap-5, those bits in the same partition are connected in G. In cases where we do not have relevant network information to build a graph G, it is possible to do so with an EDA. For example, in figure 4.2c from chapter 4 we see that the strongest connections in 2D Ising spin glass are between neighbor connections. We could set a threshold for the importance of a connection between bits, for example if an edge is seen in the models more than 5% of the time, and connect those edges in our graph G. This way the network crossover mask would connect those bits that showed the strongest linkages during EDA model building.

Using a user-specified network to modify crossover operators has been done before and our work was inspired by others. (Drezner and Salhi, 2002; Drezner, 2003; Stonedahl et al., 2008). Our version of this crossover operator most closely resembles the work by Stonedahl (Stonedahl et al., 2008). However, in Stonedahl's work a random walk through the network structure was used to generate the crossover mask. In this work a randomized breadth-first search was used. This is because we wanted to emphasize the short

range strong dependencies, since these were found to be the strongest in chapter 4. However, we are not attempting to improve this operator but instead test this operator against the standard GA operators and hBOA when using various ways to generate G.

6.3 Test Problems

This section describes NK landscapes, one of the two test problems used in this chapter. The other test problem, trap-5, is described in section 2.2.2.

6.3.1 NK Landscapes

In this chapter we consider two classes of instances, nearest neighbor NK landscapes (described in section 3.2.1) and unrestricted NK landscapes with $k = 5$.

In unrestricted NK landscapes, for each string position X_i, we generate a random set of k neighbors where each string position except for X_i is selected with equal probability. While nearest neighbor NK landscapes are solvable in polynomial time (Wright et al., 2000), unrestricted NK landscape instances are NP-complete. To solve the instances of unrestricted NK landscapes, a branch and bound algorithm is used based on ref.(Pelikan, 2008). While this branch and bound technique is complete and thus guaranteed to find the optimum, its complexity grows explonentially fast and solving large NK landscapes becomes intractable with this algorithm.

6.4 Experiments

6.4.1 Experimental Setup and Parameters

In order to examine the effects of network crossover on Trap-5, we examined problem sizes of $n = 100$ to $n = 300$. Bisection was used to determine the minimum population size to ensure convergence to the global optimum in 10 out of 10 independent runs.

For NK landscapes with nearest neighbor interactions, we considered problem sizes of $n \in \{30, 60, 90, 120, 150, 180, 210\}$, with $k = 5$. The two boundary cases of step sizes were also considered, namely $step \in \{1, 5\}$ so that we could look at the two extreme cases (most and least overlap possible between subproblems without considering those cases where the subproblems were completely disjoint). Since NK landscape instances vary in difficult, we considered 1000 random problem instances for each combination of n, k, and $step$, 1000 random problem instances were generated. In the case of unrestricted NK landscapes, problem sizes of

$n \in 20, 22, 24, 26, 28, 30, 32, 34, 36, 38$ were considered. Again, since these instances vary in difficulty, 1000 random problem instances were examined for each problem size.

One of the variables when comparing GAs to EDAs is that they often use different replacement operators. To ensure that our tests were not biased based simply on the replace operator used, new solutions were incorporated into the old population using two replacement techniques, RTR and elitism. RTR (Pelikan, 2005) is used by many EDAs as a niching method that helps to ensure diversity in a population by having new candidate solutions replace solutions that are similar to themselves in the population. Elitist replacement on the other hand works to ensure that some portion of the population's most fit solutions are maintained over time. In this work, the 50% most fit solutions are kept each generation.

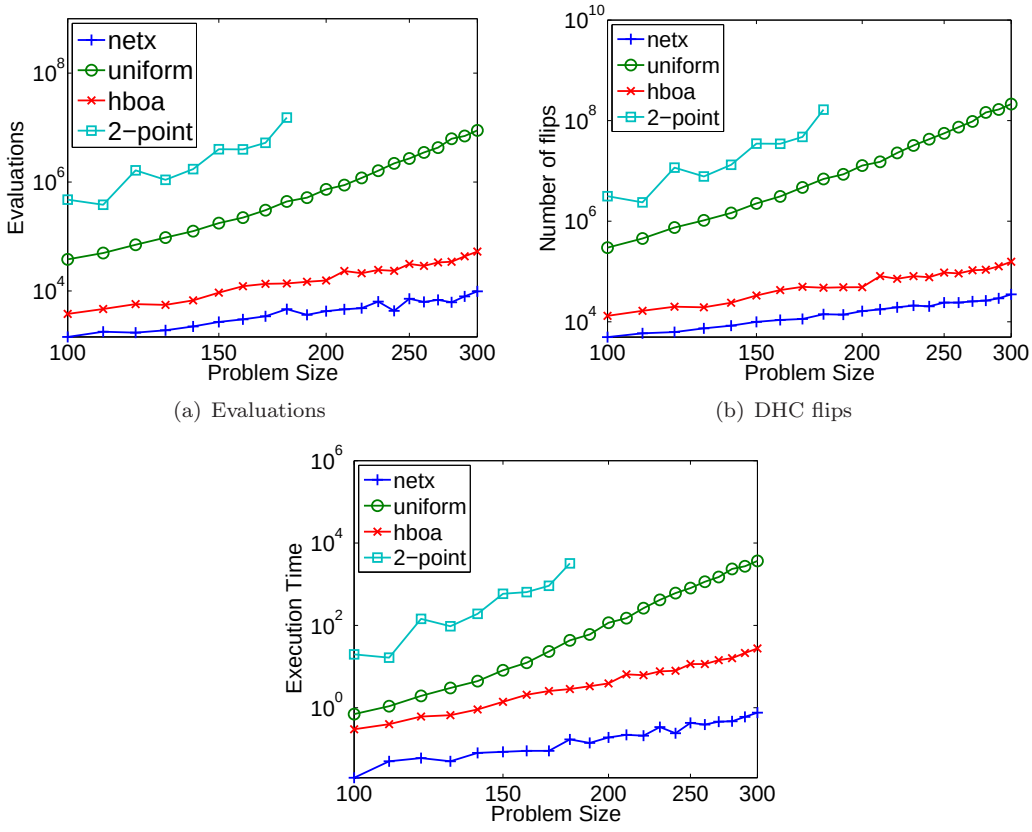
For all problem instances, GA with network, uniform and two-point crossover and hBOA were applied, with both replacement operators. For all GA runs, bit-flip mutation was used with a probability of flipping each bit of $p_m = 1/n$. For all crossover operators, the probability of crossover was set to 0.6 as per (Thierens, 1999).

For each problem instance, bisection was used to determine the minimum population size to ensure convergence to the global optimum in 10 out of 10 independent runs. Each run was terminated when the global optimum had been found (success) or when the maximum number of generations $n \times 4$ had been reached (failure). In some cases the worst performing algorithms experiments were cut short when the problem sizes necessary to solve the instances became extremely large. Typically when comparing algorithms, the number of evaluations is considered the key statistic. However, for hybrid evolutionary algorithms, often a great deal of time is spent in local search, so the number of DHC flips is also examined.

In order to use network crossover on trap-5 and NK landscapes, it is necessary to create our network G. For trap-5, this is done by connecting all bits that are in the same trap partition, with no overlap between partitions and so the graph is disjoint. For nearest neighbor NK landscapes, all bits in the same subproblem are connected. Since in all partitions in our NK landscape instances had some overlap, this resulted in a fully connected (though not complete) graph.

6.4.2 Trap-5, RTR

In figure 6.2a we examine the number of evaluations necessary to solve Trap-5 using RTR for the selected algorithms. It is clear that in regards to evaluations, the best performing algorithm is the GA with network crossover for all problem sizes. This is not a surprise, as it should be quite uncommon for network crossover to disrupt bits in the same trap partition. hBOA scales similarly to the GA with network crossover as problem size increases. On the other hand, both two-point crossover and uniform crossover perform very



(a) Evaluations

(b) DHC flips

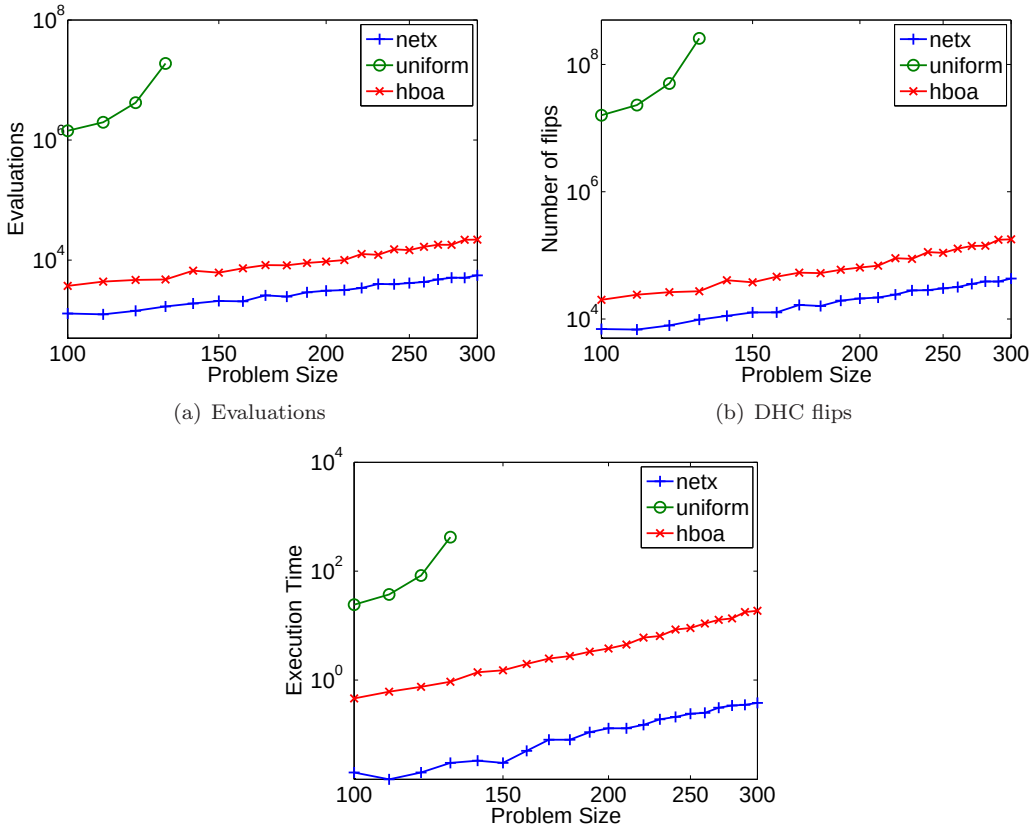
(c) Execution Time

Figure 6.2: Performance of the tested algorithms on Trap5 using RTR replacement

poorly on Trap-5 as expected. After $n = 190$, the experiments with two-point crossover were cut off due to the extreme population sizes required to solve the problem.

In figure 6.2b we examine the number of local search steps necessary to solve each problem instance for the respective algorithms. We see again that the GA with network crossover is the best performing algorithm, requiring the least DHC flips. hBOA has similar scaling as the GA with network crossover. GA with two-point crossover and uniform crossover performed quite poorly.

The previous results are emphasized when we examine the different algorithms performance on execution time in Figure 6.2c. The GA with network crossover is clearly the best performing algorithm, but again hBOA scales similarly. Yet again, uniform and 2-point again perform poorly, showing exponential scaling as problem size increases.



(a) Evaluations

(b) DHC flips

(c) Execution Time

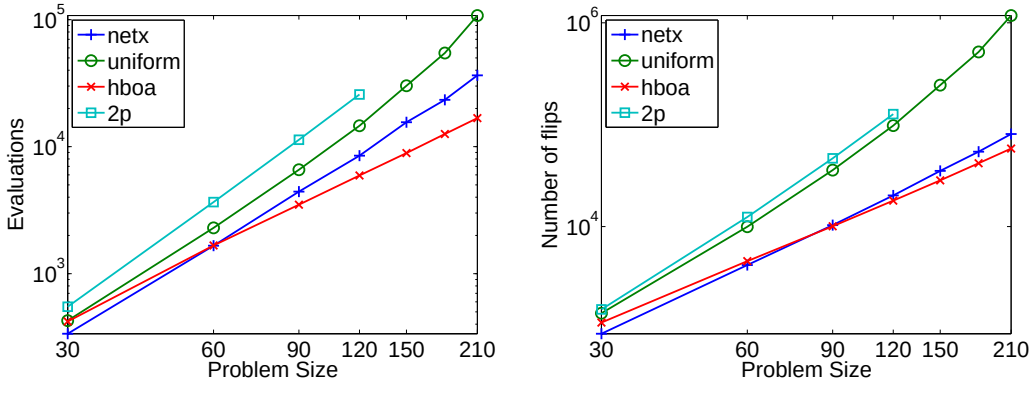
Figure 6.3: Performance of the tested algorithms on Trap-5 using elitist replacement

6.4.3 Trap-5, Elitism

The previosu results showed that with RTR, the GA with network crossover and hBOA scaled similarly on Trap-5. However, does changing the replacement method effort the relative performance of the algorithms? This is an important question, as evolutionary algorithms are often only compared with one replacement operator used.

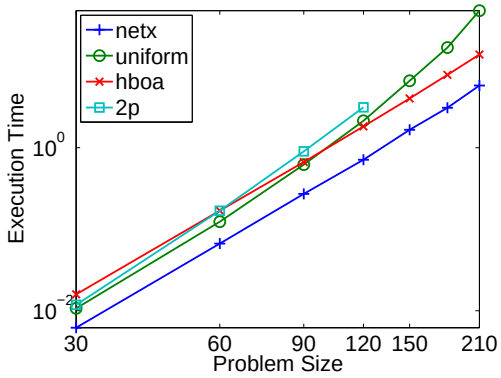
In Figure 6.3a we examine the relative performance of the algorithms on Trap-5 using elitism as the eplacement operator. Similary to RTR, the GA with network crossover is the best performing algorithm. As before, hBOA again shows similar scaling to the GA with network crossover. Both other crossover operators perform very poorly however. Indeed, two-point crossover performed so poorly that even for our smallest problem size, $n = 100$, the GA was not able to solve all 10 independent bisection runs even when using extremely large population sizes. Uniform crossover did not perform much better, and problem sizes greater than $n = 130$ were terminated.

In Figure 6.3b and Figure 6.3 we looked at the number of local search steps and the execution times



(a) Evaluations

(b) DHC flips



(c) Execution Time

Figure 6.4: Performance of the tested algorithms on nearest neighor NK landscapes using $step = 5$ and RTR replacement

necessary to solve the Trap-5 problems with elitism. The GA with network crossver and hBOA show similar scaling, but again the GA has the best performance. Uniform crossover has extremely poor performance in both metrics, showing exponential scaling of execution time and number of local search steps required.

6.4.4 Nearest Neighbor NK Landscapes, RTR

In the previous section it was shown that at least on Trap-5, the GA with network crossover outperformed all other algorithms using all of our metrics. However, hBOA did show similar scaling as problem size increased. In this section, we examine the algorithms on nearest neighbor NK landscapes.

First we examine the algorithms on the number of evaluations necessary to solve the NK landscape instances with neighbor interactions of $step = 5$ using RTR. With $step = 5$, these are the instances with the least possible interaction between subproblems without making them separable. In Figure 6.4a we see that hBOA has the best performance in regards to number of evaluations, with the performance of hBOA relative to the other algorithms actually increasing as problem since increased. Two-point crossover performs

the worst for all problem sizes. The GA with network crossover performs better than the other crossover methods, but only beats out hBOA on the very smallest sizes.

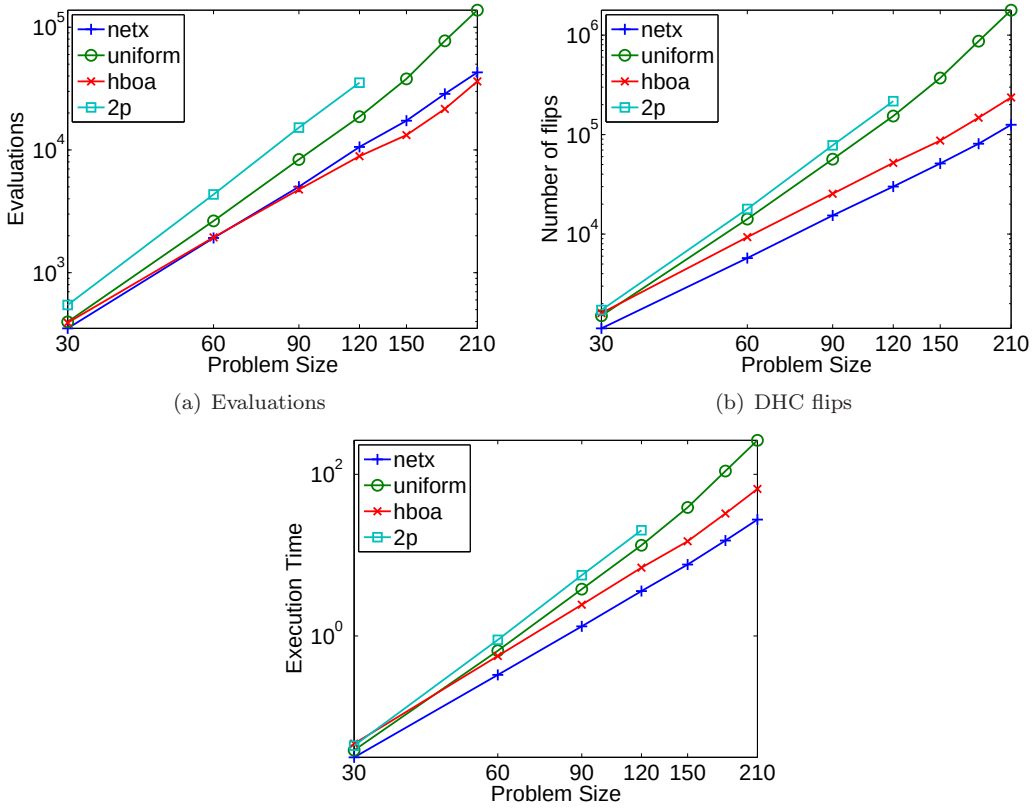
We see a slightly different story when we examine the number of local search steps required. Figure 6.4b shows the number of local search steps required to find the optimums. We see that the GA with two-point and uniform crossover perform quite poorly in relation to the other two algorithms. The GA with network crossover requires the lowest number of local search steps when the problem size is small. However, hBOA shows superior scaling and takes the lead in performance as problem size increases.

The previous results showed that in regards to evaluations and DHC flips, hBOA was the superior algorithm compared to all GA operators, with network crossover being a close second. However, when we examine the average execution time for all instauces using the various algorithms in Figure 6.4c, we see a different story. In regards to evaluation time, the GA with network crossover shows superior performance to hBOA for all problem sizes tested, hBOA showing similar scaling performance. hBOA being superior in evaluations and local search steps while having a worse execution time than the GA with network crossover is most likely due to the increased operator load in hBOA, which must build a new model each generation.

For nearest neighbor NK landscapes with as little as possible interaction between subproblems, the GA with network crossover showed the best performance. Is this same pattern repeated when the interactions between subproblems increase? In Figure 6.5a we examine the average number of evaluations required by our algorithms to solve nearest neighbor NK landscapes for $step = 1$, which gives us the maximum possible overlap between subproblems. The worst performing algorithm is again two-point crossver. hBOA slightly edges out the GA with network crossover as the best performing algorithm. However, the GA with network crossover does show similar scaling for all problem sizes tested.

We look at the local search steps required for our algorithms in Figure 6.5b. Unlike the previous results for $step = 5$, GA with network crossover had the least required local search steps when the interaction between subproblems was high. hBOA shows similar scaling as problem size increases. Only for the smallest problem sizes do the other two crossover operators perform well, but both two-point and uniform crossover show worse scaling than the other two algorithms.

When we examine execution time in Figure 6.5c, the GA with network crossover is the clear winner. hBOA again scales in a similar fashion, taking about twice as long to solve each instance on average than the GA with network crossover. The worst performing algorithm is again two-point crossover, with uniform crossover also performing poorly and showing worse scaling than either the GA with network crossover or hBOA.



(a) Evaluations

(b) DHC flips

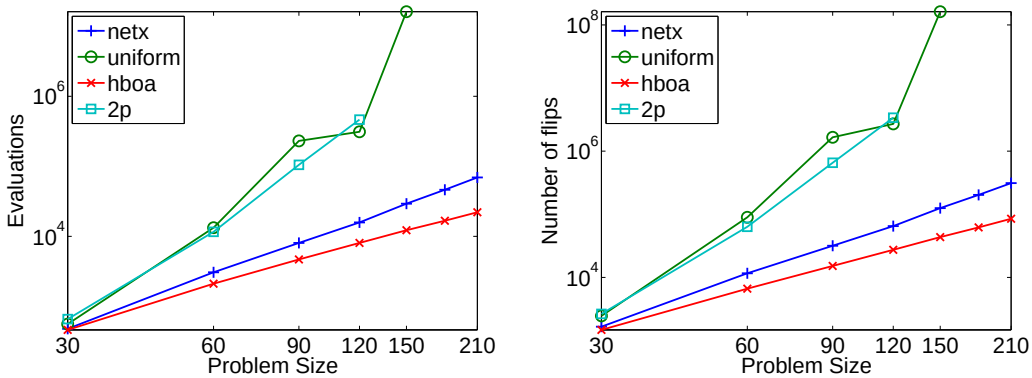
(c) Execution Time

Figure 6.5: Performance of the tested algorithms on nearest neighor NK landscapes using $step = 1$ and RTR replacement

6.4.5 Nearest Neighor NK Landscapes, Elitism

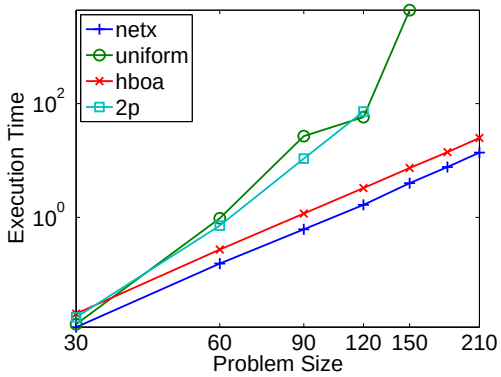
The results in the previous section showed that GA with network crossover and hBOA both perform quite well on nearest neighbor NK landscapes with RTR. On the other hand, the other GA operators performed badly. Was this due to the replacement operator chosen? In this section we examine the effects of changing the replacement operator to elitism, with the worst 50% of the population replaced with new children each generation.

Again we start by examining the average number of evaluations required to solve each problem szie for our algorithms using elitism. In Figure 6.6a we show these results using $step = 5$. We immediately see that uniform and two-point crossover show poor performance. As problem size increases, the performance gets so poor that the algorithms were unable to find the solution for even very large problem sizes and so the experiments were cut short at $n = 150$ for uniform and $n = 120$ for two-point crossover. hBOA performed the best in both overall evaluations as well as scaling. GA with network crossover performed slightly worse and scaled faster. However, it still only scaled polynomially fast.



(a) Evaluations

(b) DHC flips



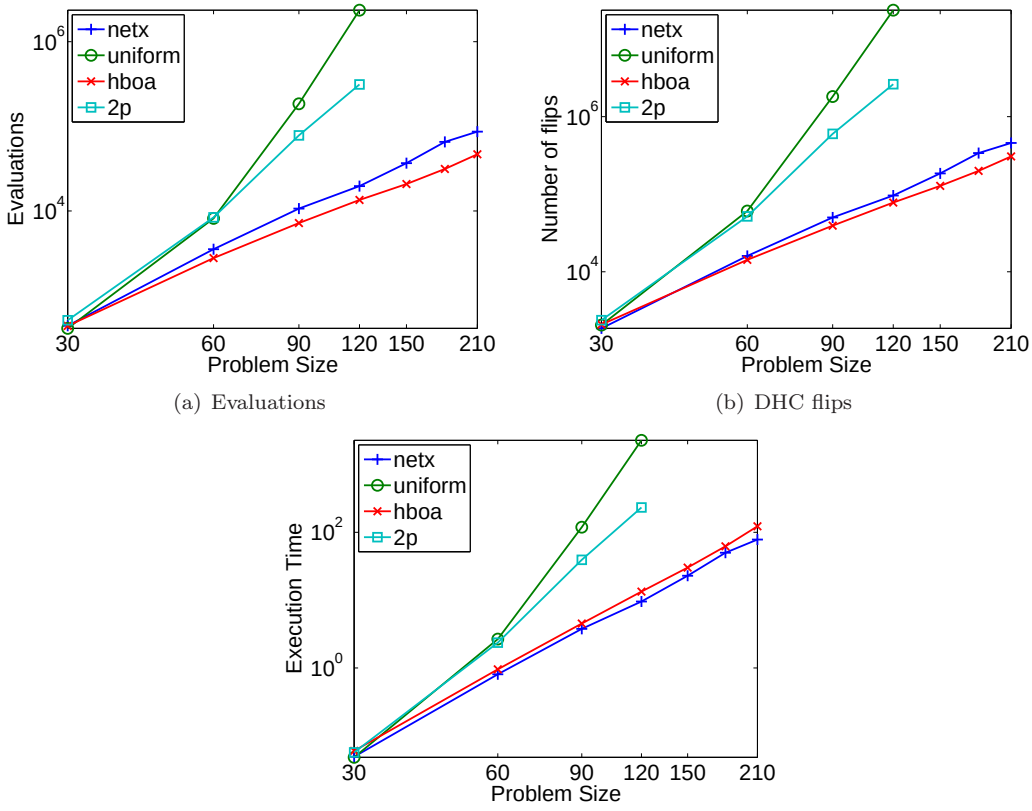
(c) Execution Time

Figure 6.6: Performance of the tested algorithms on nearest neighor NK landscapes using $step = 5$ and elitist replacement

A similar pattern is shown when we examine the number of local search steps. In Figure 6.6b we look at the number of local search steps required for our algorithms. The GAs with uniform and two-point crossover again perform quite poorly. hBOA is again the best performing algorithm. The GA with network crossover does not scale as well, but also does not show exponential scaling like the other two crossover operators.

When we examine the overall execution time for $step = 5$ and elitist replacement in Figure 6.6c, the GA with network crossover takes the lead. While the GA with network crossover required more evaluations and more local search steps, due to its lower overhead operator, it is able to outperform hBOA overall in execution time. hBOA's execution time is still close, taking about twice as long to solve the each instance than the GA with network crossover. The other algorithms again show exponential scaling.

Now we must consider elitist replacement with the greatest overlap between subproblems, that is when $step = 1$. In Figure 6.7a we show the number of evaluations required for our tested algorithms to solve the instances nearest neighbor NK landscapes with the greatest possible overlap between subproblems. Immediately we see exponential scaling for both GA with uniform and two-point crossover. This is even



(a) Evaluations

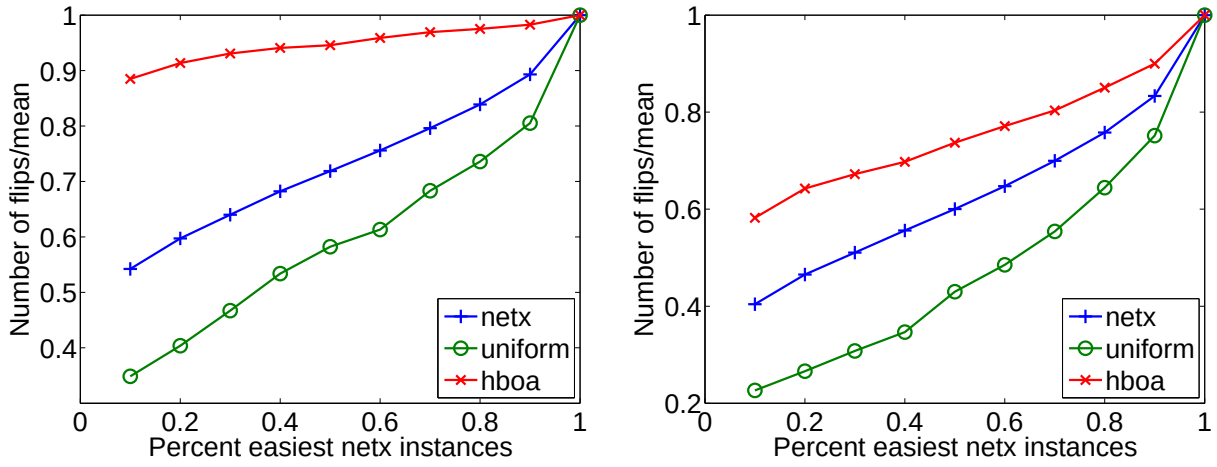
(b) DHC flips

(c) Execution Time

Figure 6.7: Performance of the tested algorithms on nearest neighor NK landscapes using $step = 1$ and elitist replacement

more severe than when RTS was used, indicating that without the niching offered by RTS, these operators have trouble solving the largest instances. Even with the greater number of interactions, hBOA again showed the best performance in regards to evaluations. The GA with network crossover took more evaluations than hBOA and this gap grows slightly as problem size increases.

We see a similar pattern in regards to DHC flips when we examine the local search steps required to solve the instances with elitism and $step = 1$ in Figure 6.7b. The GA with network crossover performs slightly more local search steps than hBOA, with the other two algorithms again showing exponential scaling. On the other hand, when we consider the overall execution time to solve the problems in Figure 6.7c, the GA with network crossover takes a slight lead over hBOA. hBOA shows very similar performance however, only being about 30% slower for even the largest problem size.



(a) $step = 5$ (b) $step = 1$

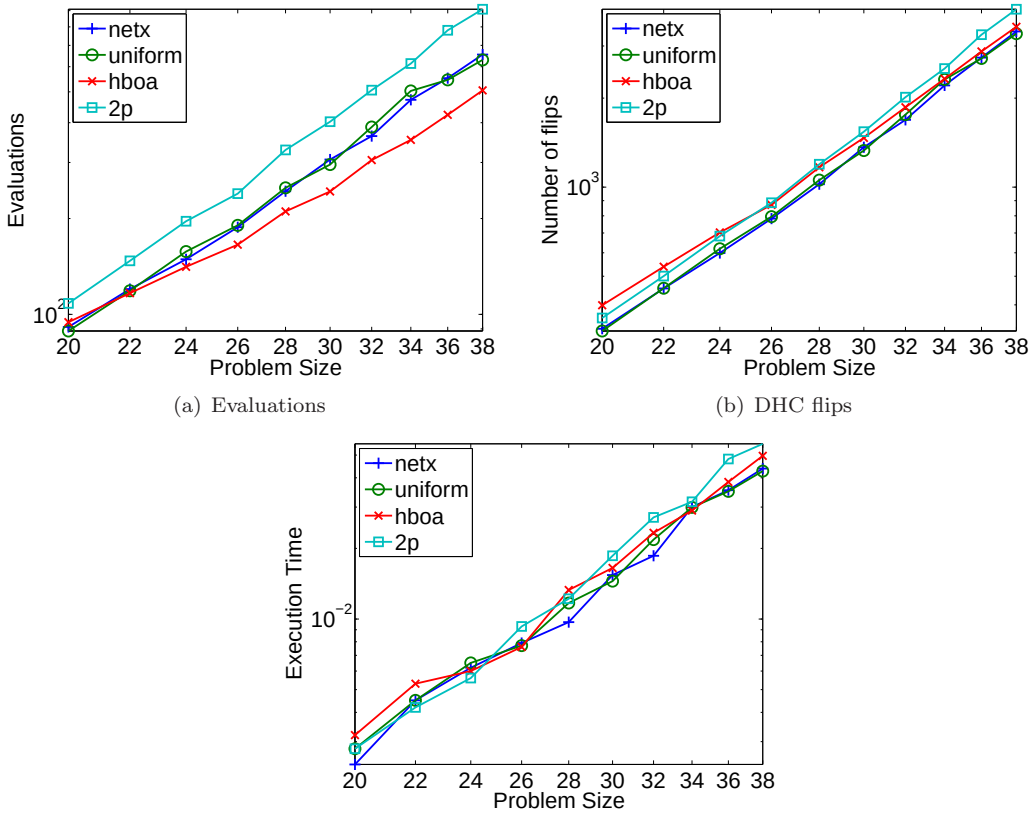
Figure 6.8: Performance of the tested algorithms using RTR replacement on nearest neighbor NK landscapes as a function of instance difficulty ranked by network crossover execution time, $n = 210$

6.4.6 Nearest Neighbor NK Instance Difficulty

In the previous results, 1000 instances of each problem size and setting were averaged. However, we know that the difficulty of any particular random nearest neighbor NK landscape can vary quite dramatically. Averaging the overall results on all instances could obscure the relative performance of the algorithms on the easiest or the hardest problems. For example, it is possible that an algorithm might have performed worse than another algorithm overall, but was superior when solving the hardest instances.

To examine the relative performance of our algorithms as instance difficulty varies, we selected a particular problem size of $n = 210$ and $step = 5$ and ranked the instances by their execution time when solved by the GA with network crossover. This was picked to rank our instances as it was the best performing algorithm. We then examined the three best performing algorithms on these instances (two-point crossover was ommitted), with the results ranked by difficulty.

In Figure 6.8 we examine the average number of local search steps for a percentage of the easiest isntances divided by the mean of all the instances for both step sizes. We see that hBOA shows the least variance in local search steps required between the easiest and the hardest instances. For $step = 5$, hBOA almost spends the same number of local search steps regardless of instance difficulty. For both step sizes, GA with uniform crossover showed a large variance in local search steps required depending on instance difficulty. GA with network crossover was between the two extremes, showing a variance in local search steps required but not as extreme as that evidenced with uniform crossover.



(a) Evaluations

(b) DHC flips

(c) Execution Time

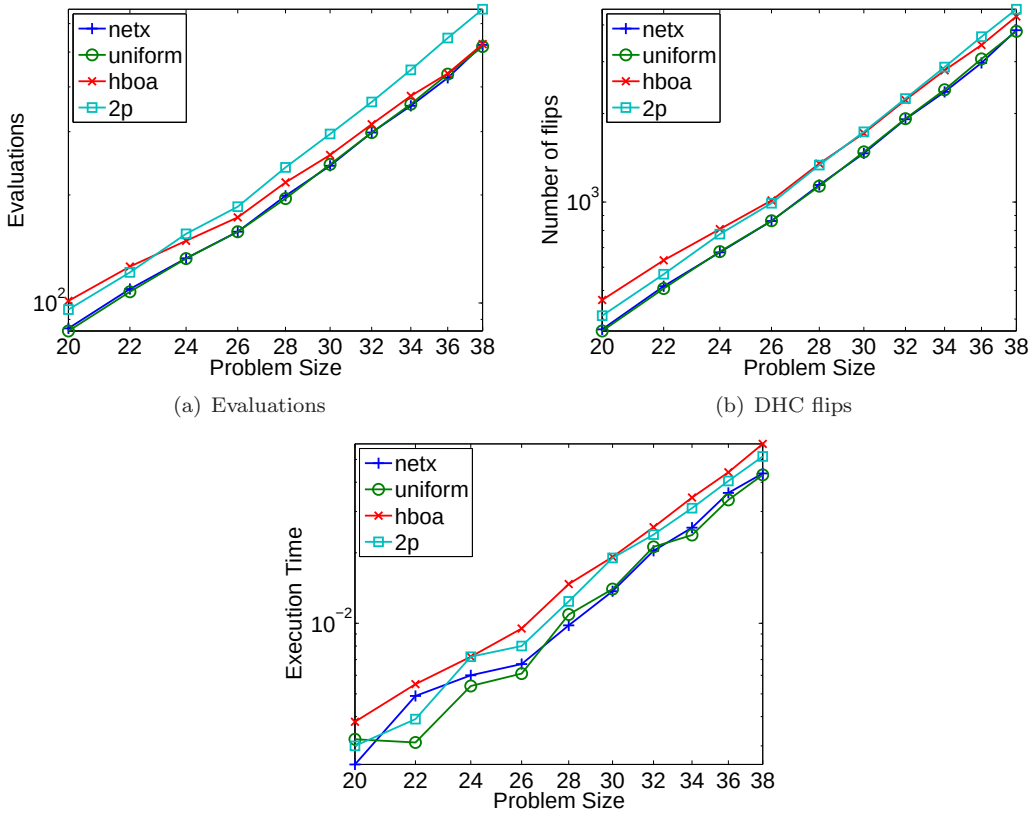
Figure 6.9: Performance of the tested algorithms on unrestricted NK landscapes using RTR replacement

6.4.7 Unrestricted NK Landscapes, RTR

The previous sections showed that the GA with network crossover was the best performing algorithm for nearest neighbor NK landscapes in regards to overall execution time. Unrestricted NK landscapes have much more complicated interacts between bits.

In Figure 6.9a we examine the average number of evaluations required for our algorithms to solve unrestricted NK landscapes using RTR. The algorithm with the worst performance is clearly two-point crossover. Uniform and network crossover have nearly the same performance as each other, only showing some slight variance between different problem sizes. hBOA on the other hand shows the best performance, scaling better than the other examined algorithms.

When we examine the number of local search steps required for the compared algorithms in Figure 6.9b, the relative performance is less clear than for evaluations. hBOA required the most local search steps for small problem sizes, but shows superior scaling as problem size increases. The GAs with uniform and network crossover required the least local search steps for all problem sizes. Two-point crossover shows the worst



(a) Evaluations

(b) DHC flips

(c) Execution Time

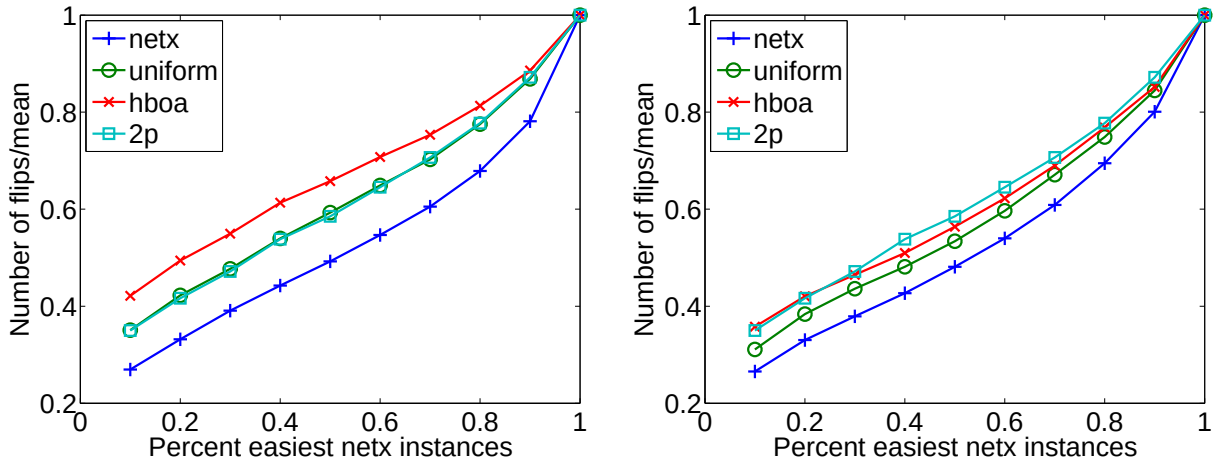
Figure 6.10: Performance of the tested algorithms on unrestricted NK landscapes using elitist replacement

scaling of the algorithms, starting off superior to hBOA in regards to local search steps, but getting worse as problem size increases.

When we examine the overall execution time for our tested algorithms in Figure 6.9c for unrestricted NK landscapes using RTR, we see that hBOA scales the best of all examined algorithms, with the GA with two-poitn crossover having the worst performance. Again we see that uniform and network crossover perform almost identically.

6.4.8 Unrestricted NK Landscapes, Elitism

To test the effect of changing our replacement technique on solving unrestricted NK landscapes, we examine the average number of evaluations required to solve the instances using elitist replacement in Figure 6.10a. The GA with uniform and network crossover performed the best on all tested instances. However, hBOA scales the best of all compared algorithms, starting off showing the most evaluations required for the smallest problem size but matching the best performance when we examine the largest problem size. Two-point



(a) RTR (b) elitism

Figure 6.11: Performance of the tested algorithms on unrestricted NK landscape instances as a function of instance difficulty ranked by network crossover execution time for $n = 38$ and RTR replacement

crossover was the worst performing operator.

When we examine the results for unrestricted NK landscapes and elitism on local search steps in Figure 6.10b, we see that again uniform and network crossover show identical performance, outperforming all other algorithms. hBOA however does show good scaling as problem size increases. Two-point crossover performs slightly better than hBOA on the smallest problem sizes, but scales the worst as problem size increases.

In Figure 6.10c we show the average execution time required by the algorithms to solve the unrestricted landscape instances with elitism. Network crossover starts off poorly but scales quite well and shows nearly the best performance on the largest problem size.

6.4.9 Unrestricted NK Instance Difficulty

As with nearest neighbor NK landscapes, unrestricted NK landscape instances vary in difficulty. To examine the effects of instance difficulty on each algorithm, we again ranked the largest problem size instances by the overall execution time required by the GA with network crossover using RTR for replacement.

In Figure 6.11 we show the average number of flips for a percentage of the easiest instances divided by the mean of all the instances using both replacement operators. We see that for all compared algorithms, there is a large variance in the amount of local search steps as the difficulty of the instances changes. This shows a difference from the restricted NK landscapes, where hBOA showed very similar performance across different difficulties. Network crossover has the most variance by difficulty, hBOA shows the least variance in difficulty when RTR is used. When using RTR, two-point and uniform crossover have nearly identical

performance. However, for elitism, two-point crossover actually has the least variance in local search steps required.

6.5 Summary

This chapter proposed a method to create a specialized network crossover operator that can be used in a GA to incorporate problem-specific knowledge. This network crossover was then used in a GA and compared to a GA using other common crossover operators and hBOA. A summary of the key points of this chapter follows:

- Proposed a network crossover that can be used in a GA to take into account previous knowledge about problem structure
- Tested the GA with the network crossover operator against the hBOA on two test problems, trap5 and NK landscapes.
- For NK landscapes with nearest neighbor interactions and trap-5, the network crossover GA required less running time than hBOA or other common GA crossover operators for all instances and parameter settings. In addition, it was shown that RTR was superior to elitist replacement on these problems.
- For unrestricted NK landscapes the results showed that hBOA scaled the best out of all tested algorithms, with network crossover performing very similarly to uniform crossover. Also, the replacement method had less effect on the results, showing very similar results for both elitist replacement and RTR.
- While the improvements over hBOA required some user-specified information, it is important to note that this is often not a difficult task. There are many classes of problems where defining such a structure is trivial, from problems in graph theory (such as graph coloring and graph bipartitioning) to Ising Spin glasses and MAXSAT. In addition, as shown in ref. (Hauschild and Pelikan, 2009) and chapter 4, it is also possible to use runs of an EDA to learn about the structure of a problem and use this information to bias EDAs.

Chapter 7

Future Work

While this thesis has presented several techniques to incorporate prior knowledge into hBOA, it should be understood that these techniques are only the first step in exploiting prior knowledge in EDAs in general. The goal was not to outline the only methods to be used in using learning from experience in EDAs, but instead to motivate future research in the exploitation of this knowledge to speed up EDAs. In the past, most researchers have not tried to take into account the ability of EDAs to incorporate prior knowledge in their work. This thesis attempts to change that mindset to one of actively attempting to exploit this information whenever possible.

In the rest of this chapter, we will outline many avenues of possible future work on the subject of learning from experience in EDAs. However, much as this thesis is not a definitive list of all methods to use prior knowledge in hBOA, this section should not be seen as the definitive list of possible avenues for future exploration of prior knowledge in EDAs.

7.1 Examination of Real-World Applications

Most of the methods in this thesis were tested against either artificial test problems like trap-5 or against classical problems in computer science such as MAXSAT or MVC. In addition, we tested these methods against 2D and 3D Ising spin glasses. This was done to provide a baseline of performance for these methods and these functions often test the design envelope of the algorithms. However, this is only the first step in using these methods. The next step is testing these methods on important real-world application problems. Testing real-world application problems, where noise and other difficulties are present, is important to show the general applicability of these methods.

7.2 Exploitation of Different Statistics

In this thesis we primarily gathered structural information to bias future model building. This was done first with the PCM that gathered the percentage of dependencies found in prior runs. Then the SPM was used

to gather the conditional probability of splits between variables in prior runs. Of course there are also other ways to store information on the structure as well. One disadvantage of these methods is that they only take into account whether or not a dependency was found during a run, not the strength of those dependencies.

However, structural information is not the only information that can possibly be exploited. In addition to the structural information, which stores how often variable i is directly dependent on j, the Bayesian networks also hold information on the strength of this actual dependency in the probability of different partial solutions being generated during sampling. While gathering statistics on the probability of different partial solutions generated during sampling might be more complicated than gathering the structural information, it has the advantage that it would directly take into account the strength of dependencies and not simply the fact that they exist. By broadening the range of possible statistics gathered from previous runs, we would expect that the types of problems these methods would apply to would increase.

7.3 Applicability to other EDAs

hBOA was the primary algorithm tested in this thesis. However, the methods outlined are not simply limited to hBOA. Many of the methods can directly be applied to other EDAs that use Bayesian network models, for example the estimation of Bayesian network algorithm (EBNA) (Etxeberria and Larrañaga, 1999) and the learning factorized distribution algorithm (LFDA) (Mühlenbein and Mahnig, 1999). However, for other EDAs with different types of models, the idea of storing information about prior runs should still be possible.

For example, with the ECGA it should be possible to store information about the different linkage groups discovered during prior runs to bias future runs of the ECGA. EDAs such as the Linkage Tree Genetic Algorithm (LTGA) could also be modified to take into account information gathered during previous runs. It should also be possible to bias the model building in Markov network model EDAs such as the Markovianity based optimization algorithm (MOA) (Shakya and Santana, 2008). The methods examined in this thesis should be tried on other EDAs and their benefits examined.

7.4 Other types of Network Crossovers

In section 6, we covered one method to generate a specialized crossover operator to incorporate prior knowledge into a genetic algorithm. This information was gathered either from a user specified network or gathered from hBOA. However, other methods of modifying crossover to take into account prior information should be explored. It is possible that using other methods to generate the crossover masks could be advantageous. For example, using a random walk through the network G could be more beneficial than the randomized

breadth-first search used in this work. Additional methods for generating a crossover mask based on networks gathered from hBOA should be attempted in the future, or even using more than one type of network mask in a run. It should also be possible to explore using specialized mutation operators that could take advantage of our network information.

7.5 Using hBOA to generate Specialized Operators

In the previous section we talked about creating specialized crossover operators based on knowledge gained by prior runs of an EDA. However, this is not the only type of specialized operator that it is possible to develop. Sometimes the way a problem is formulated can change the difficulty of the problem. Since we know it is possible to use hBOA to learn a great deal about a problem, in the future it should be possible to exploit that information to reformulate problems in different ways. This could allow algorithms that previously were unable to solve the old problem, to be more successful.

Chapter 8

Conclusions

The purpose of this chapter is to provide the summary and main conclusions of this dissertation. First, the contributions of the dissertation are summarized. Next, the main conclusions of the dissertation are provided.

8.1 What Has Been Done

A summary of the major results of this thesis follows:

Model quality in hBOA We have performed an indepth analysis of the accuracy of hBOA models generated when solving problems. The models generated from four fundamentally different sets of test problem types were examined: (1) concatenated traps, (2) randomly decomposable problems, (3) hierarchical traps and (4) two-dimensional $\pm J$ Ising spin glasses with periodic boundary conditions. The results showed that the models closely corresponded to the structure of the underlying optimization problem. Also, in many cases the models did not vary much from generation to generation, staying relatively stable through the lifetime of the problem. In addition, it was shown that while it was possible to restrict hBOA model building significantly on Ising spin glasses without affecting performance, that doing so by hand may not lead to good scalability.

Hard restrictions on hBOA model building From the results on model quality, it was clear that it was possible to restrict hBOA model building to speed up performance. Two methods were described to restrict hBOA model building based on information gathered from models in previous runs: (1) The probability coincidience matrix (PCM) method and (2) the distance threshold method. The first method was used on 2D Ising spin glasses and the second method was used on 2D and 3D Ising spin glasses, rADPs, MAXSAT and MVC. The results showed that it was possible to substantially speed up hBOA on these problems based on information garnered from previous runs.

Soft bias of hBOA model building While the results on hard restrictions were good, they had a signif-

icant downside in that it has a strong reliance on the parameter that determines which dependencies to ignore. A method was proposed to bias the metric used by hBOA to determine the quality of the dependencies, thereby avoiding the inability to add some possible connections which was a downside with hard restrictions. Using this method, substantial speedups on 2D Ising spin glasses and trap-5 were obtained. While this method required specifying a parameter, it was shown that this parameter can be set to one value and still obtain speedups in all cases.

Network crossover While information garnered from subsequent runs of an EDA can be used to speed up problem solving in other EDA runs, this information can be used in other ways to speed up problem solving. A network crossover operator was presented that could be used in a GA to take into account problem specific knowledge either specified by a user or garnered by trial runs of an EDA. The resulting GA with network crossover was shown to outperform hBOA and GAs with other crossover operators on nearest neighbor NK landscapes and trap-5. However, hBOA was able to outperform the GA with network crossover on unrestricted NK landscapes.

8.2 Main Conclusions

As was mentioned in the beginning of this thesis, one of the main advantages of EDAs over many other stochastic optimization techniques is that they supply us with a roadmap of how they solve the problem. This roadmap of probabilistic models contains a wealth of information about the problem being solved. Yet until recently, relatively little work has been done in exploiting this knowledge to speed up problem solving on similar problems.

While much work remains, for practitioners seeking to improve the speed of solving large numbers of similar problems, this thesis should make it clear that it is possible to exploit prior knowledge garnered from prior runs to speed up future problem solving. Equally important is the fact that the speedups gained from exploiting prior knowledge *stacks* with gains from parallelism, local search or other methods of efficiency enhancements. In order to fully maximize your efficiency when solving problems with EDAs, the exploitation of prior knowledge must be considered.

References

Ackley, D. H. (1987). An empirical study of bit vector function optimization. *Genetic Algorithms and Simulated Annealing*, pages 170–204.

Arst, R., Minsker, B. S., and Goldberg, D. E. (2002). Comparing advanced genetic algorithms and simple genetic algorithms for groundwater management. *Proceedings of the American Society of Civil Engineers (ASCE) Environmental & Water Resources Institute (EWRI) 2002 Water Resources Planning & Management Conference*, pages Roanoke, VA.

Bacardit, J., Stout, M., Hirst, J. D., Sastry, K., Llorà, X., and Krasnogor, N. (2007). Automated alphabet reduction method with evolutionary algorithms for protein structure prediction. *Genetic and Evolutionary Computation Conference (GECCO-2007)*, pages 346–353.

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, Carnegie Mellon University, Pittsburgh, PA.

Baluja, S. (2006). Incorporating a priori knowledge in probabilistic-model based optimization. In Pelikan, M., Sastry, K., and Cantú-Paz, E., editors, *Scalable optimization via probabilistic modeling: From algorithms to applications*, pages 205–219. Springer.

Barahona, F. (1982). On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical, Nuclear and General*, 15(10):3241–3253.

Binder, K. and Young, A. (1986). Spin-glasses: Experimental facts, theoretical concepts and open questions. *Rev. Mod. Phys.*, 58:801.

Bollobas, B. (2001). *Random Graphs*. Cambridge University Press.

Boughaci, D. and Drias, H. (2004). A performance comparison of evolutionary meta-heuristics and solving MAX-SAT problems. In Okatan, A., editor, *International Conference on Computational Intelligence*, pages 379–383. International Computational Intelligence Society.

Brownlee, A. E. I., McCall, J. A. W., and Brown, D. F. (2007). Solving the MAXSAT problem using a multivariate EDA based on markov networks. In Bosman, P. A. N., editor, *Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO'2007)*, pages 2423–2428, London, United Kingdom. ACM Press.

Cantú-Paz, E. (2000). *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer, Boston, MA.

Chen, C.-H. and p. Chen, Y. (2007). Real-coded ECGA for economic dispatch. *Genetic and Evolutionary Computation Conference (GECCO-2007)*, pages 1920–1927.

Chickering, D. M., Geiger, D., and Heckerman, D. (1994). Learning Bayesian networks is NP-hard. Technical Report MSR-TR-94-17, Microsoft Research, Redmond, WA.

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

Dayal, P., Trebst, S., Wessel, S., Würtz, D., Troyer, M., Sabhapandit, S., and Coppersmith, S. (2004). Performance limitations of flat histogram methods and optimality of Wang-Langdau sampling. *Physical Review Letters*, 92(9):097201.

Deb, K. and Goldberg, D. E. (1991). Analyzing deception in trap functions. IlliGAL Report No. 91009, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL.

Drezner, Z. (2003). A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, 15(3):320–330.

Drezner, Z. and Salhi, S. (2002). Using hybrid metaheuristics for the one-day and two-way network design problem. *Naval Research Logistics*, 49(5):449–463.

Ducheyne, E., De Baets, B., and De Wulf, R. (2004). Probabilistic models for linkage learning in forest management. In Jin, Y., editor, *Knowledge incorporation in evolutionary computation*, pages 177–194. Springer.

Etxeberria, R. and Larrañaga, P. (1999). Global optimization using Bayesian networks. In Ochoa, A., Soto, M. R., and Santana, R., editors, *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, pages 151–173. Editorial Academia. Havana, Cuba.

Fischer, K. and Hertz, J. (1991). *Spin Glasses*. Cambridge University Press, Cambridge.

Friedman, N. and Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I., editor, *Graphical models*, pages 421–459. MIT Press.

Gao, Y. and Culberson, J. C. (2002). An analysis of phase transition in NK landscapes. *Journal of Artificial Intelligence Research (JAIR)*, 17:309–332.

Gent, I., Hoos, H. H., Prosser, P., and Walsh, T. (1999). Morphing: Combining structure and randomness. *Proceedings of the American Association of Artificial Intelligence (AAAI-99)*, pages 654–660.

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

Goldberg, D. E. (1999). Using time efficiently: Genetic-evolutionary algorithms and the continuation problem. *Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 212–219. Also IlliGAL Report No. 99002.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Kluwer.

Gottlieb, J., Marchiori, E., and Rossi, C. (2002). Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(1):35–50.

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

Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. *International Conference on Genetic Algorithms (ICGA-95)*, pages 24–31.

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., and Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *International Conference on Evolutionary Computation (ICEC-97)*, pages 7–12, Piscataway, NJ. IEEE Press.

Hartmann, A. K. (1996). Cluster-exact approximation of spin glass ground states. *Physica A*, 224:480.

Hauschild, M. and Pelikan, M. (2011). An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation*, 1(3):111 – 128.

Hauschild, M., Pelikan, M., Lima, C., and Sastry, K. (2007). Analyzing probabilistic models in hierarchical boa on traps and spin glasses. *Genetic and Evolutionary Computation Conference (GECCO-2007)*, I:523–530.

Hauschild, M. W. and Pelikan, M. (2009). Intelligent bias of network structures in the hierarchical boa. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 413–420, New York, NY, USA. ACM.

Hayes, M. S. and Minsker, B. S. (2005). Evaluation of advanced genetic algorithms applied to groundwater remediation design. *Proceedings of the American Society of Civil Engineers (ASCE) Environmental & Water Resources Institute (EWRI) World Water & Environmental Resources Congress 2005 & Related Symposia*, pages Anchorage, AK.

Heckerman, D., Geiger, D., and Chickering, D. M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research, Redmond, WA.

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

Hoos, H. H. and Stutzle, T. (2000). Satlib: An online resource for research on sat. *SAT 2000*, pages 283–292. SATLIB is available online at www.satlib.org.

Howard, R. A. and Matheson, J. E. (1981). Influence diagrams. In Howard, R. A. and Matheson, J. E., editors, *Readings on the principles and applications of decision analysis*, volume II, pages 721–762. Strategic Decisions Group, Menlo Park, CA.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press.

Kauffman, S. (1989). Adaptation on rugged fitness landscapes. In Stein, D. L., editor, *Lecture Notes in the Sciences of Complexity*, pages 527–618. Addison Wesley.

Kauffman, S. A. (1993). *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press.

Kollat, J. B., Reed, P. M., and Kasprzyk, J. R. (2008). A new epsilon-dominance hierarchical Bayesian optimization algorithm for large multi-objective monitoring network design problems. *Advances in Water Resources*, 31(5):828–845.

Larrañaga, P. and Lozano, J. A., editors (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer, Boston, MA.

Lima, C. F., Lobo, F. G., and Pelikan, M. (2008). From mating pool distributions to model overfitting. In *Genetic and Evolutionary Computation Conference (GECCO-2008)*, GECCO '08, pages 431–438, New York, NY, USA. ACM.

Lima, C. F., Pelikan, M., Goldberg, D. E., Lobo, F. G., Sastry, K., and Hauschild, M. (2007). Influence of selection and replacement strategies on linkage learning in BOA. MEDAL Report No. 2007005, Missouri Estimation of Distribution Algorithms Laboratory, University of Missouri–St. Louis, St. Louis, MO.

Lima, C. F., Pelikan, M., Sastry, K., Butz, M. V., Goldberg, D. E., and Lobo, F. G. (2006). Substructural neighborhoods for local search in the Bayesian optimization algorithm. *Parallel Problem Solving from Nature*, pages 232–241.

Lipinski, P. (2007). ECGA vs. BOA in discovering stock market trading experts. *Genetic and Evolutionary Computation Conference (GECCO-2007)*, pages 531–538.

Mendiburu, A., Miguel-Alonso, J., and Lozano, J. A. (2006). Implementation and performance evaluation of a parallelization of estimation of bayesian network algorithms. *Parallel Processing Letters*, 16(1):133–148.

Mezard, M., Parisi, G., and Virasoro, M. (1987). *Spin glass theory and beyond*. World Scientific, Singapore.

Mühlenbein, H. (2008). Convergence of estimation of distribution algorithms for finite samples. Technical report, Fraunhofer Institut Autonomous intelligent Systems, Sankt Augustin, Germany.

Mühlenbein, H. and Mahnig, T. (1999). FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376.

Mühlenbein, H. and Mahnig, T. (2002). Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning. *International Journal on Approximate Reasoning*, 31(3):157–192.

Mühlenbein, H., Mahnig, T., and Ochoa-Rodriguez, A. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5:215–247.

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

Mühlenbein, H. and Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature*, pages 178–187.

Ocenasek, J. (2002). *Parallel Estimation of Distribution Algorithms*. PhD thesis, Faculty of Information Technology, Brno University of Technology, Brno.

Ocenasek, J., Cantú-Paz, E., Pelikan, M., and Schwarz, J. (2006). Design of parallel estimation of distribution algorithms. In Pelikan, M., Sastry, K., and Cantú-Paz, E., editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, pages 187–203. Springer.

Ocenasek, J. and Schwarz, J. (2000). The parallel Bayesian optimization algorithm. In *Proceedings of the European Symposium on Computational Inteligence*, pages 61–67. Physica-Verlag.

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

Pelikan, M. (2005). *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*. Springer-Verlag.

Pelikan, M. (2008). Analysis of estimation of distribution algorithms and genetic algorithms on nk landscapes. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1033–1040.

Pelikan, M. and Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. *Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 511–518.

Pelikan, M. and Goldberg, D. E. (2003). Hierarchical BOA solves Ising spin glasses and MAXSAT. *Genetic and Evolutionary Computation Conference (GECCO-2003)*, II:1275–1286.

Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. *Genetic and Evolutionary Computation Conference (GECCO-99)*, I:525–532.

Pelikan, M., Goldberg, D. E., and Lobo, F. (2002a). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20.

Pelikan, M. and Hartmann, A. K. (2006). Searching for ground states of Ising spin glasses with hierarchical BOA and cluster exact approximation. In Pelikan, M., Sastry, K., and Cantú-Paz, E., editors, *Scalable optimization via probabilistic modeling: From algorithms to applications*, pages 333–349. Springer.

Pelikan, M., Ocenasek, J., Trebst, S., Troyer, M., and Alet, F. (2004). Computational complexity and simulation of rare events of ising spin glasses. *Genetic and Evolutionary Computation Conference (GECCO-2004)*, 2:36–47.

Pelikan, M. and Sastry, K. (2004). Fitness inheritance in the Bayesian optimization algorithm. *Genetic and Evolutionary Computation Conference (GECCO-2004)*, 2:48–59.

Pelikan, M., Sastry, K., Butz, M. V., and Goldberg, D. E. (2006a). Generator and interface for random decomposable problems in c. MEDAL Report No. 2006003, Missouri Estimation of Distribution Algorithms Laboratory, University of Missouri–St. Louis, St. Louis, Mo.

Pelikan, M., Sastry, K., Butz, M. V., and Goldberg, D. E. (2006b). Performance of evolutionary algorithms on random decomposable problems. In *PPSN*, pages 788–797.

Pelikan, M., Sastry, K., and Cantú-Paz, E., editors (2006c). *Scalable optimization via probabilistic modeling: From algorithms to applications*. Springer-Verlag.

Pelikan, M., Sastry, K., and Goldberg, D. E. (2002b). Scalability of the Bayesian optimization algorithm. *International Journal of Approximate Reasoning*, 31(3):221–258.

Pelikan, M., Sastry, K., and Goldberg, D. E. (2006d). Sporadic model building for efficiency enhancement of hierarchical BOA. *Genetic and Evolutionary Computation Conference (GECCO-2006)*, pages 405–412.

Pelikan, M., Sastry, K., and Goldberg, D. E. (2008). Sporadic model building for efficiency enhancement of the hierarchical BOA. *Genetic Programming and Evolvable Machines*, 9(1):53–84.

Pelikan, M., Sastry, K., Goldberg, D. E., Butz, M. V., and Hauschild, M. (2009). Performance of evolutionary algorithms on nk landscapes with nearest neighbor interactions and tunable overlap. MEDAL Report No. 2009002, Missouri Estimation of Distribution Algorithms Laboratory, University of Missour–St. Louis, St. Louis, MO.

Peña, J., Lozano, J., and Larrañaga, P. (2004). Unsupervised learning of Bayesian networks via estimation of distribution algorithms: an application to gene expression data clustering. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 12:63–82.

Petrovski, A., Shakya, S., and Mccall, J. (2006). Optimising cancer chemotherapy using an estimation of distribution algorithm and genetic algorithms. *Genetic and Evolutionary Computation Conference (GECCO-2006)*, pages 413–418.

Radetic, E., Pelikan, M., and Goldberg, D. E. (2009). Effects of a deterministic hill climber on hboa. MEDAL Report No. 2009003, Missouri Estimation of Distribution Algorithms Laboratory, University of Missour–St. Louis, St. Louis, MO.

Rana, S. and Whitley, D. L. (1998). Genetic algorithm behavior in the MAXSAT domain. *Parallel Problem Solving from Nature*, pages 785–794.

Santana, R. (2005). Estimation of distribution algorithms with Kikuchi approximations. *Evolutionary Computation*, 13(1):67–97.

Sastry, K. (2001a). Efficient atomic cluster optimization using a hybrid extended compact genetic algorithm with seeded population. IlliGAL Report No. 2001018, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL.

Sastry, K. (2001b). Evaluation-relaxation schemes for genetic and evolutionary algorithms. Master's thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL.

Sastry, K. and Goldberg, D. E. (2004). Designing competent mutation operators via probabilistic model building of neighborhoods. *Genetic and Evolutionary Computation Conference (GECCO-2004)*, pages 114–125.

Sastry, K., Goldberg, D. E., and Pelikan, M. (2001). Don't evaluate, inherit. *Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 551–558.

Sastry, K., Pelikan, M., and Goldberg, D. E. (2006). Efficiency enhancement of estimation of distribution algorithms. In Pelikan, M., Sastry, K., and Cantú-Paz, E., editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, pages 161–185. Springer.

Schwarz, J. and Ocenasek, J. (2000). A problem-knowledge based evolutionary algorithm KBOA for hypergraph partitioning. Personal communication.

Shah, R. and Reed, P. (2010). Comparative analysis of multiobjective evolutionary algorithms for random and correlated instances of multiobjective d-dimensional knapsack problems. *European Journal of Operations Research*. In revision.

Shakya, S. and Santana, R. (2008). An EDA based on local markov property and gibbs sampling. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 475–476, New York, NY, USA. ACM.

Shakya, S. K., McCall, J. A., and Brown, D. F. (2006). Solving the Ising spin glass problem using a bivariate EDA based on Markov random fields. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2006)*, pages 908– 915.

Simon, H. A. (1968). *The Sciences of the Artificial*. The MIT Press, Cambridge, MA.

Smith, R. E., Dike, B. A., and Stegmann, S. A. (1995). Fitness inheritance in genetic algorithms. *Proceedings of the ACM Symposium on Applied Computing*, pages 345–350.

Spin Glass Ground State Server (2004). `http://www.informatik.uni-koeln.de/ls\_juenger/research/sgs/sgs.html`. University of Köln, Germany.

Srivastava, R. and Goldberg, D. E. (2001). Verification of the theory of genetic and evolutionary continuation. *Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 551–558. Also IlliGAL Report No. 2001007.

Stonedahl, F., Rand, W., and Wilensky, U. (2008). Crossnet: a framework for crossover with network-based chromosomal representations. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1057–1064, New York, NY, USA. ACM.

Thierens, D. (1999). Scalability problems of simple genetic algorithms. *Evolutionary Computation*, 7(4):331–352.

Thierens, D. and Goldberg, D. E. (1993). Mixing in genetic algorithms. *International Conference on Genetic Algorithms (ICGA-93)*, pages 38–45.

Thierens, D., Goldberg, D. E., and Pereira, A. G. (1998). Domino convergence, drift, and the temporal-salience structure of problems. *International Conference on Evolutionary Computation (ICEC-98)*, pages 535–540.

Weigt, M. and Hartmann, A. K. (2001). Minimal vertex covers on finite-connectivity random graphs - a hard-sphere lattice-gas picture. *Physical Review E*, 63:056127.

Wright, A. H., Thompson, R. K., and Zhang, J. (2000). The computational complexity of n-k fitness functions. *IEEE Trans. Evolutionary Computation*, 4(4):373–379.

Wu, H. and Shapiro, J. L. (2006). Does overfitting affect performance in estimation of distribution algorithms. *Genetic and Evolutionary Computation Conference (GECCO-2006)*, pages 433–434.

Young, A., editor (1998). *Spin glasses and random fields*. World Scientific, Singapore.

Yu, T.-L. (2006). *A Matrix Approach for Finding Extreme: Problems with Modularity, Hierarchy, and Overlap*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL.

Yu, T.-L., Santarelli, S., and Goldberg, D. E. (2006). Military antenna design using a simple genetic algorithm and hBOA. In Pelikan, M., Sastry, K., and Cantú-Paz, E., editors, *Scalable optimization via probabilistic modeling: From algorithms to applications*, pages 275–289. Springer.

Vita

Mark Hauschild was born in Alton, Illinois on February 3rd, 1974. From 1992 to 1996, Mark served in the U.S. Navy. He graduated from the University of Missouri–St. Louis in 2005 with a degree in Computer Science and in 2006 with a degree in Mathematics. He also received his Masters in Computer Science in 2009.