Universidade do Algarve

Faculdade de Ciências e Tecnologia

# Application of Substructural Local Search in the MAXSAT problem

Pedro Frazão González

Mestrado em Engenharia Informática

**2011**

Universidade do Algarve

Faculdade de Ciências e Tecnologia

# Application of Substructural Local Search in the MAXSAT problem

Pedro Frazão González

Tese orientada por
Fernando Lobo

Mestrado em Engenharia Informática

**2011**

# Resumo

Os Algoritmos Genéticos (AGs) são optimizadores estocásticos normalmente aplicados a problemas onde o uso de métodos determinísticos é inviável ou quando a informação sobre como resolver o problema é escassa.

Embora os AGs apresentem bons resultados numa ampla quantidade de problemas, não têm em conta as dependências que podem existir entre as variáveis de um determinado problema. Sem respeitar essas ligações, conseguir o óptimo seria muito difícil ou quase impossível.

Os Algoritmos da Estimação da Distribuição (AEDs) são métodos inspirados nos AGs capazes de identificar as ligações existentes entre as variáveis sem necessidade de fornecer nenhuma informação sobre a estrutura do problema. Estes métodos utilizam técnicas de *aprendizagem de máquina* para construir um modelo probabilístico que capture as regularidades presentes na *população* (conjunto de soluções candidatas para o nosso problema). O modelo aprendido é utilizado para gerar novas soluções similares àquelas presentes na população mas também com alguma inovação.

A pesquisa local sub-estrutural (PLS) é um método proposto recentemente que aproveita o modelo construído pelo AED e executa uma pesquisa local em cada uma das sub-estruturas do modelo, gerando no fim uma solução de alta qualidade. Este método mostrou ser capaz de melhorar a eficiência da procura quando aplicado a diferentes EDAs em vários problemas artificias de dificuldade limitada.

Nesta tese, a utilidade da PLS no Algoritmo de Optimização Bayesiana hierárquico (AOBh) (um AED que utiliza redes Bayesianas como modelo probabilístico), é investigada no problema MAXSAT.

Os resultados mostram que os métodos de PLS são capazes de melhorar a eficiência do AOBh, mas apenas nas instâncias MAXSAT com poucas variáveis. Para instâncias maiores esse comportamento não se verifica. Adicionalmente, a execução da PLS é analizada de forma a compreender melhor os resultados obtidos. Por fim, são expostas algumas observações e propostas

para um melhoramento dos métodos de PLS.

**Palavras chave:** Algoritmos genéticos, algoritmos da estimação da distribuição, redes Bayesianas, pesquisa local sub-estrutural, MAXSAT.

# Abstract

Genetic Algorithms (GAs) are stochastic optimizers usually applied to problems where the use of deterministic methods is not practical or when information about how to solve the problem is scarce. Although traditional GAs show good results in a broad range of problems, they do not take into account the dependencies that may exist among the variables of a given problem. Without respecting these *links*, achieving the optimum can be very hard or even impossible.

Estimation of Distribution Algorithms (EDAs) are methods inspired on GAs that are able to learn the linkage between variables without providing any information about the problem structure. These methods use *machine learning* techniques to build a probabilistic model that captures the regularities present in the *population* (a set of candidate solutions for our problem). The learned model is used to generate new solutions similar to those present in the population but also with some innovation.

The Substructural Local Search (SLS) is a method recently proposed that takes advantage from the model built by the EDA and performs local search in each substructure of the model, providing in the end a high quality solution. This method has shown to improve the efficiency of the search when applied to different EDAs in several artificial problems of bounded difficulty.

In this thesis, the utility of SLS in the hierarchical Bayesian Optimization Algorithm (hBOA) (an EDA that uses Bayesian networks as probabilistic model), is investigated in the MAXSAT problem.

Results show that SLS is able to improve the efficiency of hBOA, but only on MAXSAT instances with a small number of variables. For larger instances that behavior is not observed. Additionally, the SLS execution is analyzed in order to better understand the obtained results. Finally, some observations and suggestions are exposed for an improvement of SLS.

**Keywords:** Genetic algorithms, estimation of distribution algorithms, Bayesian networks, substructural local search, MAXSAT.

# Agradecimentos

Gostaria de agradecer ao meu orientador, prof. Dr. Fernando Lobo, por me ter proposto este projecto, ajudado, e por me fornecer todo o material que necessitei para o realizar. Sempre senti curiosidade pela área de Algoritmos Genéticos mas a realização desta tese fez com que sentisse um interesse cada vez maior pela área.

Quero agradecer à minha namorada Catarina por ter estado sempre ao meu lado, ter-me compreendido e apoiado em todas as ocasiões e por me ter dado carinho todos estes anos.

Quero também agradecer aos meus pais Zeca e Beatriz. Sobretudo à minha mãe por me ter ouvido, ajudado e aturado durante esta árdua fase.

Finalmente, gostaria de agradecer aos meus colegas de mestrado e laboratório e aos meus amigos, em especial ao Natanael que me tem acompanhado durante longos anos.

O trabalho desta tese foi inserido no âmbito do projecto *Efficiency Enhancement Techniques for Probabilistic Model Building Genetic Algorithms* com a referência PTDC/EIA/67776/2006, financiado pela Fundação para a Ciência e Tecnologia (FCT/MCTES).

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation and main contributions

Estimation of distribution algorithms (EDAs) are methods that endow traditional Genetic Algorithms (GAs) with a valuable feature. They use machine learning techniques on the population with promising solutions to learn *on-the-fly* the underlying structure of a given problem.

Recently, a local search method that exploit the structure learned by EDAs has been applied successfully in a set of artificial problems [13]. The method is called substructural local search (SLS) and it is able to reduce the number of objective function evaluations required by the EDA to find the optimum. The reduction of the number of objective function evaluations is important because many real problems require costly evaluations, e.g., the computation of the objective function may involve an expensive simulation.

The success of SLS in the artificial problems tested leads us now to investigate its utility in the context of real problems. This thesis gives a contribution in that direction by investigating the application of SLS in random instances of the Maximum Satisfiability problem, an NP-complete problem commonly referred as the MAXSAT problem in computer science literature.

## 1.2 Thesis organization

This thesis is divided into six chapters. The first chapter presents the motivation for making this work and an overview of the chapters that composes it.

Chapter 2 reviews the basics of traditional Genetic Algorithms, providing common terminology and the basic procedure. It then introduces the Estimation of Distribution Algorithms and show how they differentiate from traditional GAs. EDAs are then classified into three types that differ in the complexity of the models they use.

Chapter 3 introduces the Bayesian Optimization Algorithm (BOA). It belongs to the third class of EDAs. In order to understand the behavior of BOA, an introduction about Bayesian networks is provided, and all the steps of the algorithm are detailed. Among these, it is explained how to learn a Bayesian network from a dataset and how to sample new individuals from the learned network. This chapter includes several examples to help the reader visualize and better understand the operation of the algorithm. Finally, it presents and extension made to BOA, the hierarchical BOA (hBOA), which is capable of solving a broader class of problems than the original BOA, and is considered to be one of the most powerful EDAs that currently exist.

Chapter 4 presents Substructural Local Search (SLS) and its integration in hBOA. Fitness estimation which is used by SLS is introduced in this chapter as well. Then, loopy SLS is presented. It is another method of performing substructural local search but it is based on loopy Belief Propagation principles.

Chapter 5 investigates the performance of hBOA in the MAXSAT problem when applying different types of local search to the EDA. Additionally, an analysis to the execution of loopy SLS is performed.

Chapter 6 suggests a set of topics for future work and presents the thesis conclusions.

# Chapter 2

# Genetic and Estimation of Distribution Algorithms

## 2.1 Introduction

Genetic Algorithms (GAs) [5, 11] are stochastic methods that are often applied to optimization problems. Their design was inspired by natural evolution ideas. The survival of the fitter individuals and transmission of their genetic information to the offspring are some key concepts that lead to the success of these algorithms. GAs have been applied in many problems of science, business, engineering and even in non-traditional areas such as drug design or composition of music [6].

Like all stochastic optimizers, GAs do not guarantee that the best possible solution for the problem, or *optimum*, will be found. Deterministic methods could be used instead, although this would require either evaluating the entire search space or having problem-specific knowledge. The latter is often hard to find, and even then, the interaction between the decision variables of the problem sometimes is not so clear. GAs present a simple, yet robust, way to find good solutions, needing no information about how to solve the problem, what gives them a wide applicability in the area of computational optimization.


Figure 2.1: Example of an individual with fitness 3 for the 6-bit onemax.

The chapter starts with an introduction to traditional Genetic Algorithms. Some artificial problems that present different difficulties for the GAs are presented along the chapter.

Estimation of Distribution Algorithms (EDAs) are then introduced. These algorithms aim to overcome some of the difficulties present in the traditional GAs like the disruption of building blocks. This task is achieved by building a probabilistic model that is able to identify the linkage between the variables of the problem. The chapter ends with the identification of three classes of EDAs.

## 2.2 The basic operation of GAs

When solving any given problem with a Genetic Algorithm, one must represent somehow the solutions, also called *individuals*. Common representations are binary strings, integer vectors, real vectors, permutations and combinations of these. The representation of the individual is referred to as *chromosome*. Using the analogy from genetics, each string position is called a *gene* and its value is the *allele*. The quality of the individuals is known as *fitness* and is given by a *fitness function*. In this thesis only binary strings of fixed length are considered.

The chromosome representation and the fitness function vary depending on the problem that we are trying to solve. Figure 2.1 shows an example of an individual for the *onemax* problem, an artificial problem defined as the sum of bits present in a binary string and therefore the optimum is a string of all 1s. In the example is shown an individual of length $\ell = 6$ with fitness 3.

The basic operation of a GA is shown in Figure 2.2 and consists in the

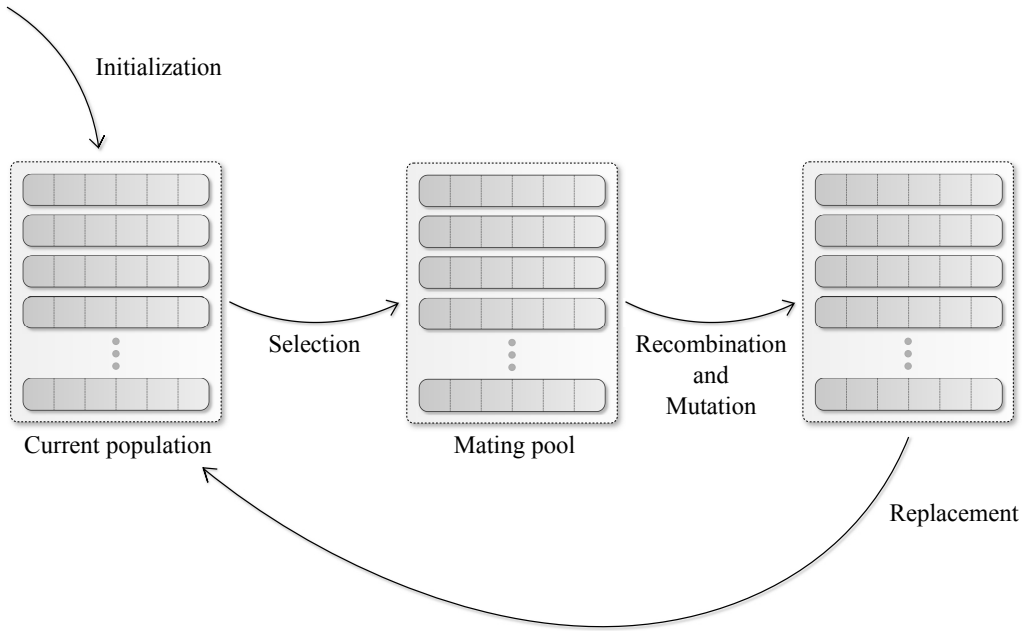



Figure 2.2: Basic GA procedure.

following steps:

1. **Initialization.** A *population* of individuals of size $n$ is created and is generally initialized at random, although good solutions can be injected into the initial population to speedup convergence if we have that problem-specific information.

2. **Selection.** In this step, the best solutions from the current population are identified and placed into the *mating pool*. In order for selection to be performed, each of the individuals in the population must be evaluated using the fitness function. Then, a selection operator is used, basing its decision in the fitness of the individuals. Many selection operators exist in the literature. In this thesis it will be used *tournament selection* without replacement [26] which picks $s$ individuals at random from the current population and copies the one with higher fitness to the mating pool. Since selection is made without replacement, the participants of a tournament are not candidates for another tournament

until all individuals have competed. The process is repeated until the mating pool is full. Using this selection operator it is guaranteed that every individual participates in exactly $s$ tournaments [13].

3. **Recombination.** Now that we have the most promising individuals selected, it's time to create new ones, and if possible better ones. Again, many alternatives exist for the recombination operator. Typical ones are *one-point crossover* which takes two individuals, $a$ and $b$, selects a random position in the chromosome and combines the left part of $a$ with the right part of $b$ and vice-versa, generating two new offspring, and also *uniform crossover* that exchanges each bit of the parents with 50% probability, which mixes completely all the genes of both individuals.

4. **Mutation.** After recombination, *mutation* can be also performed. In binary strings mutation is done by flipping each bit of the string with a very small probability. The purpose of mutation is to explore new areas of the search space, that would be inaccessible by crossover alone.

5. **Replacement.** Finally, the new set of individuals must be re-inserted into the population. Typically, it replaces the entire old population, but other techniques, that will be introduced latter on, can be used as well.

   A new iteration is executed, and the process continues until the termination criteria are met. For example, until a maximum number of generations its attained, the best individual in the population does not improve its fitness for a pre-specified number of generations, or the population reaches a certain level of convergence.

### 2.2.1 The population's role

The onemax problem introduced previously presents no difficulty for a GA, mainly because there is no relation between the genes of an individual; they are independent.

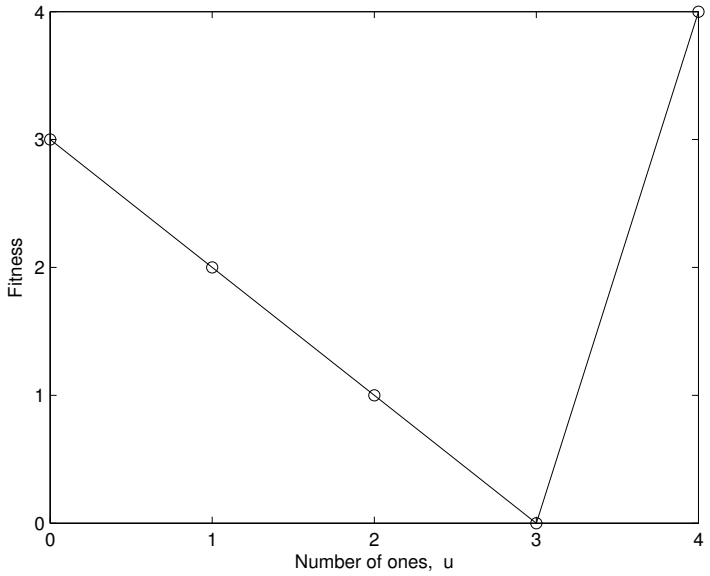



Figure 2.3: Trap function of order 4.

Problems one would like to solve in real life are not so simple, and will likely present some groups of genes dependent between them. Highly fit groups are called *building blocks* (BBs) [11]. As an example of another artificial problem, consider the deceptive trap function [4] defined as:

$$f_{trap}(u) = \begin{cases} k & \text{if } u = k, \\ k - 1 - u & \text{otherwise.} \end{cases} \tag{2.1}$$

where $u$ is the number of ones in the string and $k$ is the size of the trap. Fig. 2.3 depicts the graphical representation for $k = 4$.

Note how the fitness increases as the number of ones decreases. This causes an algorithm with no information about the problem to climb the deceptive peak until it reaches a local optimum with fitness 3, because it is easier, but the global optimum lies in the opposite side with fitness 4. The $m$-$k$ trap problem is a concatenation of $m$ trap functions of order $k$.

As we know from the problem definition, the genes in each partition are dependent between them and each partition is independent from the others.

A 1111 in any partition is a building block. In onemax a building block is simply a 1 in any position.

In [7] the authors point out the importance of a proper supply of BBs in the initial population and of deciding well among competing BBs. These two factors depend on the population size and influence the quality of the solutions found by the GA [7]. Problems with short BBs have higher probability to generate at random more BBs in the initial population, thus needing smaller population sizes than problems with long BBs. Growing and mixing the building blocks are also necessary to obtain reliable convergence to high quality solutions. Is has been shown that if the BBs are tightly linked (i.e., the genes that constitute a BB are located close to each other in the chromosome) then the GA is able to accurately solve problems in sub-quadratic time complexity [7]. However, if the BBs are loosely linked (i.e., the genes that constitute a BB are scaterred along the chromosome), then the population grows exponentially (and so does the overall time complexity of the GA) with increasing number of BBs [29].

The issue is finding the correct population size for problems of which we don't know the structure, thus neither the building blocks. If we use small population sizes, the quality of the solutions will probably not be good enough, and if we use too large ones, we will waste time doing extra processing.

The $m$-$k$ trap problem helps to understand that recombination can easily disrupt some building blocks that were already found, which reduces the chances to achieve the optimum. A good recombination mechanism should be able to combine promising solutions without disrupting too many BBs while selection increases their proportion in the population. Standard recombination operators can only do that when the BBs are tightly linked, or when linkage information is available to allow the utilization of a special purpose operator that takes that information into account. Unfortunately, such information is often unavailable for most real world problems.

In the next section we shall see how the problem of building block identi-

fication (also referred in the literature as *linkage learning*) can be addressed by using probabilistic models.

## 2.3 Estimation of Distribution Algorithms

To overcome the problem of the disruption of building blocks, a growing interest arose in methods that are able to learn the structure of the problem *on the fly* and use this information to ensure an efficient growth and mixing of BBs [24]. EDAs (Estimation of Distribution Algorithms) [17] also known as PMBGAs (Probabilistic Model Building Genetic Algorithms) [24], differ from the simple GA in the way they process the population of promising solutions (mating pool) and generate new individuals [20]. Instead of the traditional recombination and mutation operators, these algorithms use probabilistic modeling of promising solutions to guide the exploration of the search space. The main feature of EDAs is to prevent disruption of important partial solutions, which is done by giving them high probability to be present in the offspring population [13].

### 2.3.1 The basics

The mating pool contains a set of promising solutions identified by the selection operator of the GA. If we knew the probability distribution that is able to generate such set, we could use that distribution to generate new individuals that would be similar to the ones contained in the set.

The procedure of EDAs is very similar to that of GAs. They start by initializing the population at random and selecting in each generation the promising solutions. Then, the true probability distribution of the selected individuals is estimated and this distribution is used to sample new individuals similar to the previous ones but also with some innovation. The new solutions will replace some or all of the old individuals and a new generation will take place until the termination criteria are met. Fig. 2.4 presents the described procedure.

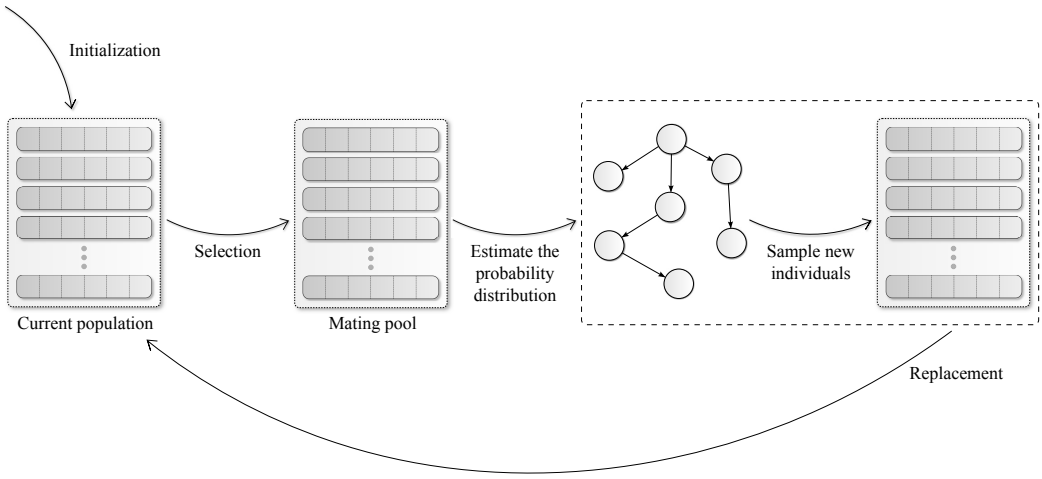



Figure 2.4: Basic EDA procedure.

The recombination and mutation operators of the simple GA were replaced by the following two steps:

1. Estimation of the probability distribution of the promising solutions.

2. Sampling new solutions from the estimated distribution.

However, the estimation of the true probability distribution is not a trivial task. There is a trade-off between the accuracy and the efficiency of the estimate [24].

The next sections present three classes of EDAs that differ in the complexity of the models they use.

### 2.3.2 Univariate models

The simplest model to start with, is one that represents no interactions among variables (see figure 2.5). The compact GA (cGA) [9], the UMDA [17], and PBIL [1], are popular algorithms that use these kind of models.

The cGA replaces the entire population by a *probability vector* $(p_1, p_2, \ldots, p_n)$, where $p_i$ represents the proportion of 1s in the position $i$ of the string. In figure 2.6 are shown two examples of possible populations for a given

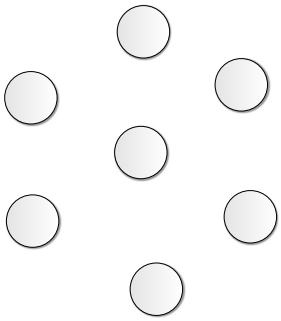
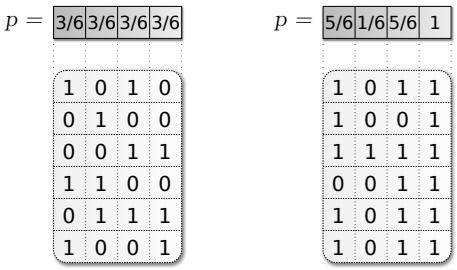


Figure 2.5: Graphical model with no interactions covered. The nodes represent the variables of the problem.




(a) (b)

Figure 2.6: Illustration of what a probability vector represents. In cGA there is no population, only a vector is needed.

probability vector. As stated previously, the population does not exist and only the vector is needed.

Each $p_i$ is initialized to 0.5, which means that the proportion of 1s and 0s is the same in each position. Then, two individuals are sampled from the vector and compete with each other, providing a winner $w$ and a looser $l$. The vector is then updated in the following way:

$$p_i = \begin{cases} p_i + \frac{1}{n} & \text{if } w_i = 1 \text{ and } l_i = 0, \\ p_i - \frac{1}{n} & \text{if } w_i = 0 \text{ and } l_i = 1, \\ p_i & \text{otherwise.} \end{cases} \tag{2.2}$$

This corresponds to picking two individuals at random from a population of size $n$ and replacing the looser by a copy of the winner. The proportion

of the winner's alleles will increase by $1/n$ in the population (except when $w_i = l_i$ where $p_i$ remains unchanged).

The algorithm stops when the vector converges, i.e., when each position of the vector is either 0 or 1. The final solution is represented by $p$.

The univariate marginal distribution algorithm (UMDA) [17] also uses a probability vector, but unlike cGA, it maintains a population of individuals. It starts by performing selection of promising solutions and the probability vector is calculated from the set of selected individuals. Then, new individuals are sampled from the vector replacing the old ones and the process repeats until the termination criteria are met.

These algorithms have similar performance. They have linear or subquadratic performance on linear problems but fail in problems where strong interaction among the variables exist [24].

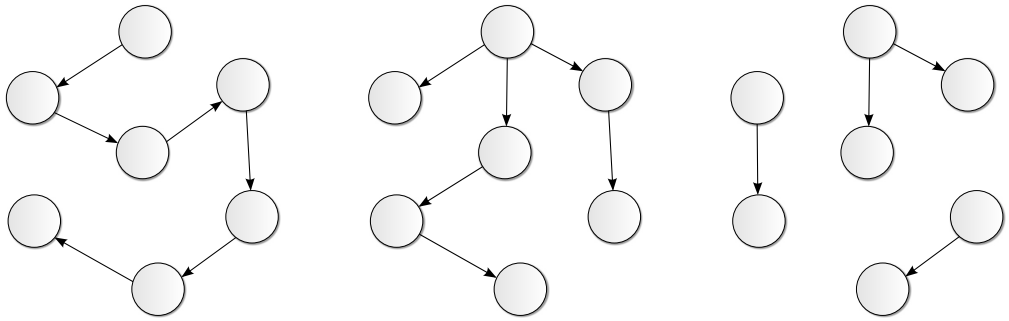
The next section extends the previous methods by considering pairwise interactions.

### 2.3.3 Bivariate models

In order to consider the dependency of the variables of a given problem, some algorithms that could cover pairwise interactions appeared. For example, the mutual-information-maximizing input clustering (MIMIC) algorithm [3] uses a chain where each node corresponds to a variable (see figure 2.7(a)). The chain represents an ordering of the variables were each variable is dependent of the previous one, except for the first, that is independent. Therefore, the first position stores the probability of generating a 1 for the first variable, and every other position stores the conditional probability of generating a 1 given the value of the previous variable.

Examples of other bivariate models are the dependency trees and forest distributions (set of mutually independent dependency trees) shown in figures 2.7(b) and 2.7(c).

This kind of algorithms can reproduce and mix BBs of order two very efficiently, but they are insufficient to solve problems with multivariate or



(a) Chain (b) Dependency tree (c) Forest distribution

Figure 2.7: Graphical models with pairwise interactions covered. The nodes represent the variables of the problem.

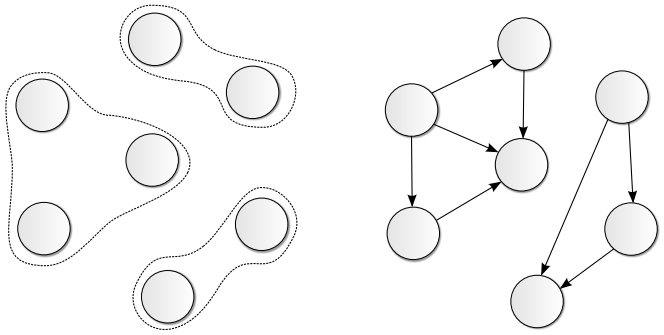
highly-overlapping building blocks [24]. Thus, more complex models are needed to address these more complex problems.

### 2.3.4 Multivariate models

The extended compact genetic algorithm (ECGA) [8] allows to capture higher order interactions by grouping the variables into mutually exclusive sets (see figure 2.8(a)). The algorithm builds the model using a greedy algorithm based on the MDL (minimum description length) metric. This metric prefers models that allow higher compression of data (in this case, the set of promising solutions) but penalizing too complex ones. The model building is an iterative process that starts with all variables separated. Then, in each iteration, it merges the groups that increase the metric the most until no more improvement is possible.

The problem of ECGA is that it is not able to capture interactions between variables of different groups because a variable cannot belong to two or more groups simultaneously. This is an issue when dealing with problems with overlapping building blocks.

Another kind of representation is used in the Bayesian Optimization Algorithm (BOA) [20]. It uses Bayesian networks (figure 2.8(b)) to model the set of promising solutions. Bayesian networks are directed acyclic graphs



(a) ECGA (b) BOA

Figure 2.8: Graphical models with multivariate interactions covered. The nodes represent the variables of the problem and edges represent dependencies. (a) is the model used by ECGA and (b) is a Bayesian network, which is the model used by BOA.

(DAGs) where the nodes represent the variables and the edges represent dependencies. Like the ECGA, the BOA also uses a greedy algorithm, but it is based in metrics for Bayesian networks. Two common metrics are the BIC (Bayesian Information Criterion) [28] and the BD (Bayesian Dirichlet) [10]. The use of Bayesian networks overcomes the problem of overlapping building blocks present in ECGA.

In the next chapter, the BOA and an extension to it, the hierarchical BOA (hBOA) [20], are explained in more detail.

## 2.4 Summary

This chapter gives an introduction about Genetic Algorithms (GAs). Common terminology is described and the procedure of traditional GAs is presented. Two artificial problems, onemax and $m$-$k$ traps, are described as well. These problems help to visualize the concept of *building blocks* (BBs). The importance of a proper supply of BBs is discussed, as well as a proper growing and mixing of them.

Right after, EDAs (Estimation of Distribution Algorithms) are intro-

duced. These kind of algorithms are able to identify BBs and ensure a proper mixing. They replace the recombination and mutation operators of the traditional GA with the estimation of the probability distribution of the promising solutions and sampling of new individuals from the estimated distribution. Three classes of EDAs that differ in the models they use are identified. Those can be univariate, bivariate or multivariate models. The Bayesian Optimization Algorithm (BOA) uses a kind of multivariate model that allows to capture interactions between several variables even if they belong to overlapping building blocks. BOA is the subject of the next chapter.

# Chapter 3

# Bayesian Optimization Algorithm

## 3.1 Introduction

In the previous chapter, methods that are able to capture multivariate interactions were briefly introduced. One of those methods, the Bayesian Optimization Algorithm (BOA), is here explained in more detail. BOA is an EDA that uses Bayesian networks to model the population of promising solutions. By using Bayesian networks it is able to identify interactions among variables, thus identifying building blocks, and furthermore, it is able to capture interactions between variables among overlapping building blocks.

The chapter starts by providing the procedure of BOA. Then, an introduction to Bayesian networks is presented as well as a method to learn Bayesian networks from data and how to sample new individuals from a Bayesian network. The chapter ends with the hierarchical BOA, an extension made to BOA that solves hierarchical problems.

*Bayesian Optimization Algorithm*

1: Generate initial population.
2: Perform selection on the current population.
3: Build a Bayesian network $B$ for the selected solutions.
4: Sample a set of new solutions from $B$.
5: Replace some or all solutions in the current population with the new ones.
6: If the termination criteria are not met, go to step 2.

Figure 3.1: Pseudocode of the Bayesian Optimization Algorithm.

## 3.2 Procedure of BOA

The basic procedure of BOA is described in figure 3.1. As traditional GAs, it starts by generating an initial random population. Then, promising solutions from the current population are selected using any selection operator. After that, a Bayesian network that models the population of promising solutions is built and this network is used to generate new solutions. The new solutions are inserted into the current population replacing some or all of the old ones. The algorithm runs until the termination criteria are met.

In the following sections an introduction to Bayesian networks will be presented and then steps 3, how to learn a Bayesian network from a given dataset, and 4, how to sample solutions from the learned network, will be explained. These are the two steps that differ from traditional genetic algorithms.

## 3.3 Bayesian networks

A Bayesian network (BN) is a probabilistic graphical model that allows to represent and reason about an uncertain domain [12].

It is composed by:

- **A structure:** a directed acyclic graph (DAG), where the nodes rep-

resent the variables of our problem and the edges represent the conditional *dependencies* between them.

- **Parameters:** a set of conditional probability tables (CPTs). Each variable will have one table which contains the conditional probabilities given any instance of its parents.

Consider the example[1] in figure 3.2. That Bayesian network encodes the relationships between having a history of smoking ($H$), bronchitis ($B$), lung cancer ($L$), fatigue ($F$) and the result of a chest X-ray ($C$). Table 3.1 identifies the possible instances of each variable. The edges represent direct influences. For example, having a history of smoking influences directly the presence of a lung cancer, and the presence of a lung cancer also has a direct influence in the result of a chest X-ray. There is no edge from $H$ to $C$ because having a history of smoking does not influence directly the result of the chest X-ray, e.g., if a person has already a lung cancer, the result of the chest X-ray is likely to be positive and is not going to be affected by the fact of the person having a smoking history or not. In this case, we say that $C$ is conditionally *independent* of $H$ given $L$. More formally, a variable is conditionally independent of the set of all its nondescents [2] given the set of all its parents.

In the context of PMBGAs, conditional dependencies will cause the involved variables to maintain the configuration seen in the population, whereas conditional independencies lead to the mixing of partial solutions [20].

Each variable in the figure has one CPT. It contains the conditional probability of the variable given its parents. For example, in the table for bronchitis, the probability to have bronchitis given that there is a history of smoking is 0.25 and the probability to have bronchitis given that there is no history of smoking is 0.05. There is no need to store the conditional probabilities for $b_2$ because, since the variables are binary, $p(b_2|H) = 1 - p(b_1|H)$.

[1] Example adapted from [18].

[2] Given two variables $X$ and $Y$, $Y$ is nondescent of $X$ if there is no path from $X$ to $Y$.

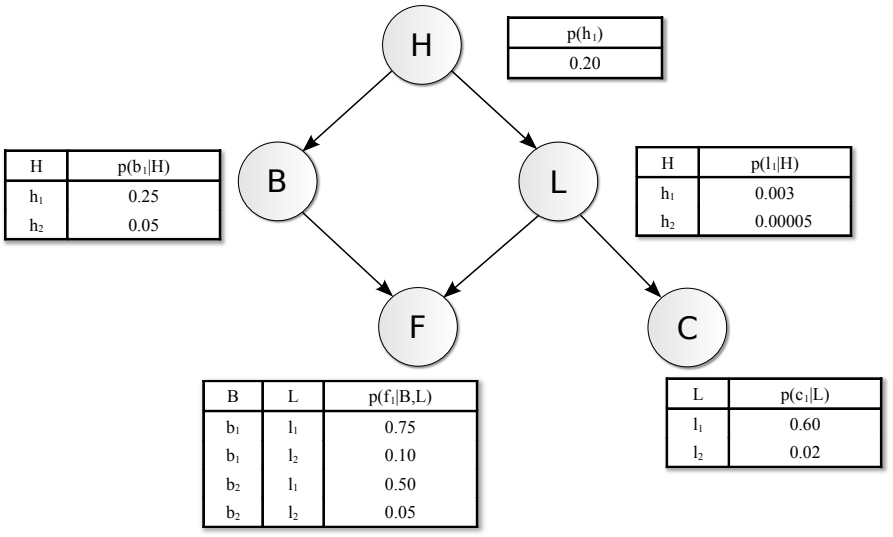



Figure 3.2: A Bayesian network and the corresponding Conditional Probability Tables (CPTs).

Table 3.1: Possible instances for each variable in the Bayesian network of figure 3.2.

| Feature | Value | When the feature takes this value |
|---|---|---|
| H | $h_1$ | There is a history of smoking |
| | $h_2$ | There is no history of smoking |
| B | $b_1$ | Bronchitis is present |
| | $b_2$ | Bronchitis is absent |
| L | $l_1$ | Lung cancer is present |
| | $l_2$ | Lung cancer is absent |
| F | $f_1$ | Fatigue is present |
| | $f_2$ | Fatigue is absent |
| C | $c_1$ | Chest X-ray is positive |
| | $c_2$ | Chest X-ray is negative |

A Bayesian network encodes a joint probability distribution, which can be written as:

$$p(X) = \prod_{i=1}^{\ell} p(X_i|\Pi_i), \tag{3.1}$$

where $X = (X_1, X_2, \ldots, X_\ell)$ is a vector with all the variables of the problem, $\Pi_i$ is the set of parents of the variable $X_i$ (set of nodes from which there exists an edge to $X_i$), and $p(X_i|\Pi_i)$ is the conditional probability of $X_i$ given its parents $\Pi_i$.

In the previous example this would be written as follows:

$$p(f, c, b, l, h) = p(f|b, l)p(c|l)p(b|h)p(l|h)p(h). \tag{3.2}$$

The next section discusses alternative representations for the CPTs that reduce the memory requirements while storing the same information.

## 3.4 Using local structures

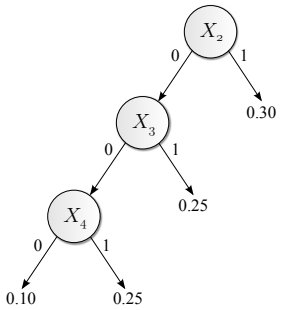
The parameters of a Bayesian network can be stored in a more efficient way than using CPTs. Local structures allow a compact representation. This is useful for large problems with many interactions between variables because the number of conditional probabilities that need to be specified in a Bayesian network grows exponentially with the order of interactions encoded by the BN [20].

A motivating example[3] to use local structures is shown in figure 3.3. The table contains the conditional probabilities for $X_1 = 1$ given its parents $X_2, X_3, X_4$. In a CPT all instances of the parents must be enumerated and each corresponding conditional probability should be stored. As we can see in the table, repeated values are unnecessarily stored. Decision trees can hold the same information in a compact way. Like the CPTs, each variable of the Bayesian network has one decision tree associated to it. The parents of the variable are represented as nodes in the tree and each node must have

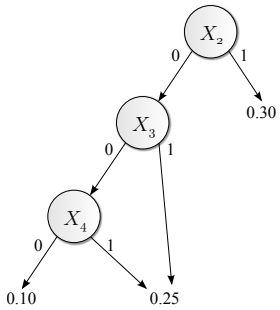
[3]adapted from [20].

| $X_2$ | $X_3$ | $X_4$ | $p(X_1=1 \mid X_2,X_3,X_4)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.10 |
| 0 | 0 | 1 | 0.25 |
| 0 | 1 | 0 | 0.25 |
| 0 | 1 | 1 | 0.25 |
| 1 | 0 | 0 | 0.30 |
| 1 | 0 | 1 | 0.30 |
| 1 | 1 | 0 | 0.30 |
| 1 | 1 | 1 | 0.30 |

(a) CPT




(b) Decision tree




(c) Decision graph

Figure 3.3: An example of a Conditional Probability Table (CPT) and the equivalent decision tree and decision graph. The use of a decision graph reduces the stored conditional probabilities from 8 to 3.

one edge for each possible value it can take. When a node is labeled with a variable $v$, we call it a *split* on $v$. A leaf of the tree contains valuable information for the instances that end the traversal of the tree in that leaf. In figure 3.3(b) we can see that the decision tree stores only 4 conditional probabilities instead of 8.

An extension to decision trees are decision graphs. They introduce an operator called *merge*. It takes two equal leaves and merges them into a single one. In figure 3.3(c) we can see how the leaf containing the value 0.25 was merged into a single one, thus reducing the stored conditional probabilities from 8 to 3 when compared to the CPT. Each variable $X_i$ will have one decision graph associated to it denoted by $G_i$.

## 3.5 Learning the Bayesian network

This section explains how the Bayesian network that models the dataset (in our case the population of promising solutions) is built.

This learning process is divided in two parts: (1) learning the structure and (2) learning the parameters.

### 3.5.1 Structure learning

Learning the structure has two components:

1. **A scoring metric:** a measure of how well a network models the data.
2. **A search procedure:** used to explore the search space of all possible networks in order to find the one with higher value of the metric.

#### 3.5.1.1 Scoring metric

One metric usually used to quantify the quality of a given network structure is the Bayesian Dirichlet metric (BD) [10] and is given by:

$$BD(B) = p(B) \prod_{i=1}^{\ell} \prod_{\pi_i} \frac{\Gamma(m'(\pi_i))}{\Gamma(m'(\pi_i) + m(\pi_i))} \prod_{x_i} \frac{\Gamma(m'(x_i, \pi_i) + m(x_i, \pi_i))}{\Gamma(m'(x_i, \pi_i))}, \tag{3.3}$$

where $p(B)$ is the prior probability of the network B, it can bias the construction towards particular structures by assigning higher prior probabilities to those structures [20]; $\Gamma$ function is defined as $\Gamma(a) = (a-1)!$; the product over $\pi_i$ runs over all instances of $\Pi_i$ (all possible combination of values of the parents of the variable $X_i$) and the product over $x_i$ runs over all instances of $X_i$ (in the binary case: 0 and 1).

$m(\pi_i)$ is the number of instances in the dataset with $\Pi_i$ instantiated to $\pi_i$. When the set $\Pi_i$ is empty, there is one instance of $\pi_i$, and $m(\pi_i)$ is set to $n$ (the size of the dataset); $m(x_i, \pi_i)$ is the number of instances in the dataset that have both $X_i$ set to $x_i$ as well as $\Pi_i$ set to $\pi_i$.

We have also that:

$$m(\pi_i) = \sum_{x_i} m(x_i, \pi_i), \tag{3.4}$$

where the sum runs over all instances of $X_i$ (0 and 1 in the binary case).

$m'(\pi_i)$ and $m'(x_i, \pi_i)$ denote prior information about the values $m(\pi_i)$ and $m(x_i, \pi_i)$ respectively. The K2 variant of the metric [10] uses an uninformative prior which assigns $m'(x_i, \pi_i) = 1$ and analogically to equation 3.4 $m'(\pi_i) = \sum_{x_i} m'(x_i, \pi_i)$.

An example adapted from [23] of a simple dataset and the usage of the K2 metric with $p(B) = 1$ (all networks are treated equally) follows next.

Consider we have two variables $X_1$ and $X_2$ and the following dataset:

| $X_1$ | $X_2$ |
|---|---|
| 0 | 0 |
| 0 | 0 |
| 1 | 1 |
| 0 | 0 |

First, we will compute the value of the K2 metric for a network with no edges $B_{empty}$. The $m'(x_i, \pi_i)$ are set to 1 and using the analogy to equation 3.4 the $m'(\pi_i)$ are set to 2. The values of the terms of equation 3.3 for each variable are show in the next tables.

| $x_1$ | $m(x_1, \pi_1)$ |
|---|---|
| 0 | 3 |
| 1 | 1 |

| $x_2$ | $m(x_2, \pi_2)$ |
|---|---|
| 0 | 3 |
| 1 | 1 |

Since $\Pi_1$ and $\Pi_2$ are empty, marginal frequencies are counted in the above tables and $m(\pi_1) = m(\pi_2) = 4$ (size of the dataset).

The result of the K2 metric for this network is:

$$\begin{aligned} BD(B_{empty}) &= \frac{(2-1)!}{(2+4-1)!} \cdot \frac{(1+3-1)!}{(1-1)!} \cdot \frac{(1+1-1)!}{(1-1)!} \cdot \\ &\cdot \frac{(2-1)!}{(2+4-1)!} \cdot \frac{(1+3-1)!}{(1-1)!} \cdot \frac{(1+1-1)!}{(1-1)!} \\ &= \frac{1}{400}. \end{aligned}$$

For a network $B_{1\to 2}$ with one edge from $X_1$ to $X_2$, the set of parents of $X_1$ remains empty, thus $m(\pi_1) = 4$, but for $X_2$, $\Pi_2 = \{X_1\}$. $m'(x_i, \pi_i)$ and $m'(\pi_i)$ are again set to 1 and 2 respectively.

The tables with the terms of equation 3.3 follows:

| $x_1$ | $m(x_1, \pi_1)$ |
|---|---|
| 0 | 3 |
| 1 | 1 |

| $x_2$ | $\pi_2$ | $m(x_2, \pi_2)$ |
|---|---|---|
| 0 | 0 | 3 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $\pi_2$ | $m(\pi_2)$ |
|---|---|
| 0 | 3 |
| 1 | 1 |

The result of the K2 metric for this network is:

$$\begin{aligned} BD(B_{1\to 2}) = &\frac{(2-1)!}{(2+4-1)!} \cdot \frac{(1+3-1)!}{(1-1)!} \cdot \frac{(1+1-1)!}{(1-1)!} \cdot \\ &\cdot \frac{(2-1)!}{(2+3-1)!} \cdot \frac{(1+3-1)!}{(1-1)!} \cdot \frac{(1+0-1)!}{(1-1)!} \cdot \\ &\cdot \frac{(2-1)!}{(2+1-1)!} \cdot \frac{(1+0-1)!}{(1-1)!} \cdot \frac{(1+1-1)!}{(1-1)!} \\ = &\frac{1}{160}. \end{aligned}$$

The result would be the same with an edge from $X_2$ to $X_1$. The network with connected nodes gets higher score than the network without edges. In fact, we can see in the dataset that the variables are correlated, the value of one variable determines the value of the other.

#### 3.5.1.2 Scoring metric for decision graphs

When using decision graphs the BD metric is very similar:

$$BD(B) = p(B) \prod_{i=1}^{\ell} \prod_{l \in L_i} \frac{\Gamma(m'_i(l))}{\Gamma(m_i(l) + m'_i(l))} \prod_{x_i} \frac{\Gamma(m_i(x_i, l) + m'_i(x_i, l))}{\Gamma(m'_i(x_i, l))}, \quad (3.5)$$

where $L_i$ is the set of leaves in decision graph $G_i$, $m_i(l)$ is the number of instances in the dataset which end the traversal through $G_i$ in the leaf $l$, $m_i(x_i, l)$ is the number of instances that end the traversal through $G_i$ in the leaf $l$ and have $X_i = x_i$; $m'_i(l)$ and $m'_i(x_i, l)$ represent prior knowledge about the values $m_i(l)$ and $m_i(x_i, l)$.

The K2 variant of the BD metric for Bayesian networks with decision graphs assigns $m'_i(x_i, l) = 1$ as an uninformative prior. The prior probability

$p(B)$ of each network can be adjusted in order to favor simpler networks over more complex ones by using the following penalty [20]:

$$p(B) = 2^{-0.5 \log_2(n) \sum_{i=1}^{\ell} |L_i|} \tag{3.6}$$

where $|L_i|$ is the number of leaves in decision graph $G_i$.

Another popular metric used is the Bayesian Information Criterion (BIC) [28]. It is a minimum description length metric and is based in the assumption that the model quality is related to the amount of data compression that the model can achieve. Regarding the scope of this thesis in substructural local search, the BD metric with a modification in the penalty showed elsewhere better results when compared to the BIC metric [13].

#### 3.5.1.3 Search procedure

A search procedure generates networks to be evaluated by the scoring metric. However, the problem of finding the best network has been shown to be NP-complete for most scoring metrics [2]. Thus, in BOA, a simple greedy algorithm is used to build the network to achieve a good compromise between search efficiency and model quality. The algorithm starts with an empty network and at each step it applies the operation that improves the scoring metric the most until no more improvement is possible. Common operations are (1) *edge additions*, (2) *edge removals* and (3) *edge reversals*. It is important to maintain the network acyclic, thus, operations that introduce cycles in the network should not be performed. To upper-bound the complexity of the final network it is useful to limit the number of edges that end in any node by a number $k$ [20, 22]. Figure 3.4 describes the pseudo-code for the learning algorithm.

For decision graphs, the greedy algorithm used to construct the Bayesian network differs slightly from the previous one. In this case, the algorithm does not manipulate directly the network, but it modifies the decision graphs corresponding to the variables of the network. Note that the nodes in a decision graph $G_i$ of the variable $X_i$ are in fact the parents of $X_i$. Thus,

*Bayesian Network Learning greedy algorithm*

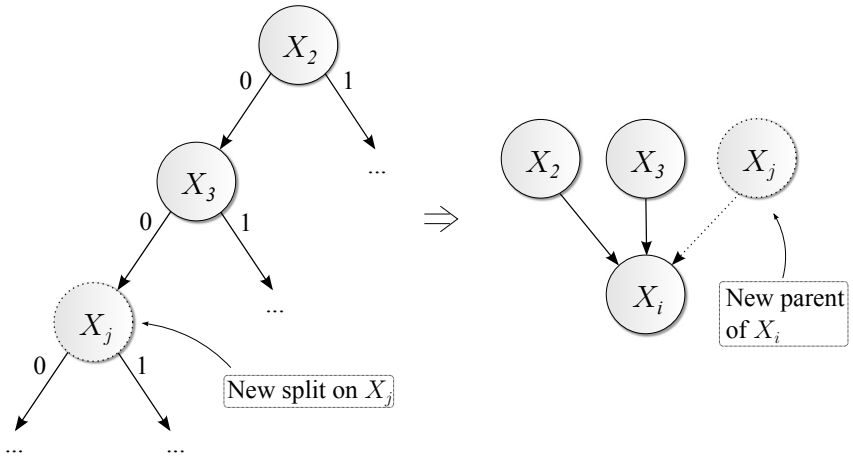
1: Initialize the network $B$ to an empty network.
2: Compute the scoring metric for $B$.
3: **for all** operations applicable to $B$ **do**
4: Compute the scoring metric for the resulting network.
5: **if** no operation improves the previous metric score **then**
6: Finish.
7: **else**
8: Apply the operation that improves the scoring metric the most to $B$ and return to step 3.

Figure 3.4: Pseudocode of the greedy algorithm used to learn the structure of a Bayesian network.

when splitting a leaf of the graph $G_i$ on $X_j$ this corresponds to introduce an edge $X_j \rightarrow X_i$ in the Bayesian network, if the edge does not exist already (see figure 3.5).

The greedy algorithm starts with the graph $G_i$ for each variable $X_i$ initialized to a single leaf containing only marginal probabilities $p(X_i)$. This corresponds to initialize the Bayesian network $B$ to an empty network. In each iteration, all operations (splits and merges) applicable to all decision graphs $G_i$ are examined and the scoring metric is computed. The operator that increases the metric the most is applied to the decision graph and the network $B$ is updated. When performing a split it should be verified that no cycles are introduced into $B$. The algorithm stops when no further improvement is possible.

Modifying the decision graphs instead of modifying directly the Bayesian network allows the construction process to be made with smaller and more specialized steps which can lead to improving the quality of the resulting model [20].




(a) Decision graph $G_i$ for variable $X_i$ (b) Bayesian network $B$

Figure 3.5: The effect of splitting a variable in the decision graph. The new split corresponds to introducing an edge in the Bayesian network.

### 3.5.2 Parameter learning

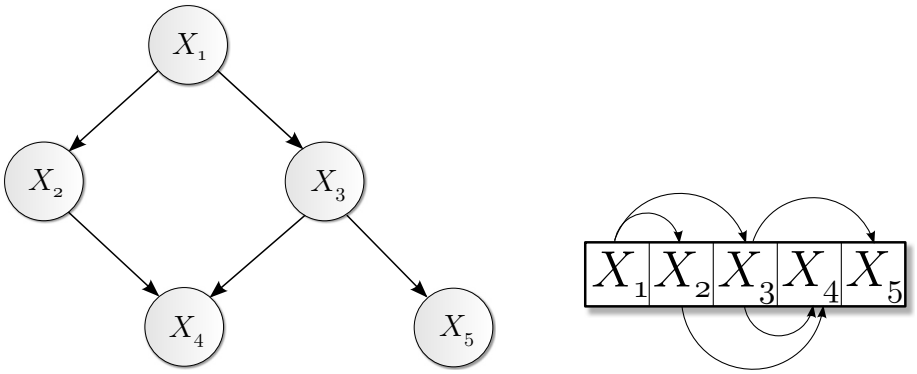
Learning the parameters is a simple task in BOA. After the structure is learned, the conditional probabilities stored in the CPTs or in the leafs of the decision graphs are calculated from the dataset by computing the relative frequencies observed. The probability vector of UMDA shown in section 2.3.2 is an example of learning the parameters for an empty network.

## 3.6 Sampling from the Bayesian network

After the Bayesian network is learned, new individuals are created according to the distribution encoded by the network. The generation of new individuals is performed using probabilistic logic sampling which consists in two steps:

1. computing an ancestral ordering (topological sort) of the nodes of the network.

2. generate values for each variable according to the computed ordering and the conditional probabilities.




(a) A Bayesian network $B$ (b) An ancestral ordering of $B$

Figure 3.6: A Bayesian network and an ancestral ordering of its nodes.

The ancestral ordering is made in order to have each variable preceded by its parents (see figure 3.6). Then, following the ancestral ordering, the value for each variable is generated according to the stored conditional probabilities. The ancestral ordering ensures that when attempting to generate a value for a variable $X_i$, the values of the variables that $X_i$ depends on (values of its parents) have already been generated. The second step is performed to generated each new individual.

## 3.7 Hierarchical BOA

This section presents the hierachical BOA (hBOA) [20, 21]. It is an extension made to BOA that is able to solve hierarchically decomposable functions (HDFs), a harder class of artificial problems than the previous ones presented. In short, HDFs are functions defined on multiple levels where each level receives as input the solutions found on lower levels. The solutions on each level are partitioned according to some structure defined by the function. The overall fitness is then computed as the sum of the fitness contribution of all partitions on all levels.

The pseudocode of hBOA is presented in figure 3.7. It differs from the one for BOA shown in section 3.2 in steps 3 and 6.

*Hierarchical Bayesian Optimization Algorithm*

1: Generate initial population.
2: Perform selection on the current population.
3: Build a Bayesian network $B$ with local structures for the selected solutions.
4: Sample a set of new solutions from $B$.
5: Evaluate the sampled set.
6: Insert new solutions in the current population using RTR.
7: If the termination criteria are not met, go to step 2.

Figure 3.7: Pseudocode of the Hierarchical Bayesian Optimization Algorithm.

The differences from BOA are the following:

1. BOA uses traditional Bayesian networks (with CPTs) and hBOA uses Bayesian networks with decision graphs.

2. hBOA introduces a different replacement strategy in order to maintain diversity in the population.

The first difference addresses one of the keys for hierarchical success which is chunking [20]. The goal of chunking is to allow groups of variables to be merged as a single variable or an intact block while representing partial solutions efficiently. A compact representation allows to encode larger solutions in the model. This compact representation is allowed by using local structures such as decision graphs.

Another key for hierarchical success is the preservation of alternative solutions. This is usually referred to as *niching* and is used to maintain diversity in the population. The purpose for preserving multiple alternative solutions is that in some problems it is not clear whether a solution will be good or not in future generations until some generations had passed already and more knowledge about the problem is gathered. Solutions that are similar will share the same niche, making different niches to coexist in the population.

Niching is achieved in hBOA by using a different replacement method explained in the following section.

### 3.7.1 Restricted Tournament Replacement

The replacement strategy used in hBOA is called *restricted tournament replacement* (RTR). For each new solution in the set sampled from the Bayesian network a subset of solutions from the current population is selected at random. The size of this subset is fixed to a constant $w$, called the *window size*. Then, the new solution is compared to all solutions in the selected subset in order to find the most similar using for this purpose the Hamming distance[4]. If the fitness of the new solution is better than the most similar, it replaces the latter in the population; otherwise, the new solution is discarded.

As suggested in [20], the window size is set to the number of bits in the problem, $w = \ell$.

## 3.8 Summary

This chapter explains in detail each of the components of BOA. An introduction about Bayesian networks is given as well as for decision graphs, local structures that replace the conditional probability tables to enhance storage efficiency. A greedy algorithm to learn Bayesian networks (BNs) from data is presented, both for traditional BNs and BNs with decision graphs. These algorithms use the Bayesian Dirichlet metric to measure how well a given network models the data. This metric has given better results than other tested metrics in previous works in terms of substructural local search (a topic to be addressed in the next chapter). The final step of learning Bayesian networks, parameter learning, is briefly introduced an then the procedure to generate new individuals from the learned Bayesian network is explained.

[4]The Hamming distance between two binary strings is the number of positions at which the bits are different.

The chapter ends with a description of the hierarchical BOA. This algorithm uses decision graphs to store the conditional probabilities and also uses a different replacement strategy called restricted tournament replacement that maintains diversity in the population.

The next chapter will review some methods that take advantage of the linkage structure learned by the Bayesian network and use this information to perform an informed local search.

# Chapter 4

# hBOA with Substructural Local Search

## 4.1 Introduction

The previous chapter introduced an algorithm, the hBOA [20], that was able to identify the underlying structure of the problem being solved. This structure is represented as a Bayesian network.

This chapter introduces two local search methods that exploit the substructures of the learned Bayesian network to produce high quality individuals. The first method is the Substructural Local Search (SLS) [15, 13]. Its inclusion in hBOA is presented in section 4.2. This algorithm uses substructural fitness information to perform the local search. That information is gathered using fitness estimation which is explained in section 4.3. In section 4.4 the procedure of the SLS is detailed.

The second method is called loopy SLS [14, 13] which is based in loopy Belief Propagation [19]. It uses *message passing* techniques among the nodes of the network to propagate the *beliefs* of each node to its neighbors. This algorithm returns the most probable configuration for a given network which is then used to instantiate a new individual. The loopy SLS as well as the Belief Propagation algorithm are detailed in section 4.5. This section also

*Hierarchical Bayesian Optimization Algorithm with SLS (hBOA+SLS)*

1: Generate initial population.
2: Perform selection on the current population.
3: Build a Bayesian network $B$ with local structures for the selected solutions.
4: Sample a set of new solutions from $B$.
5: Evaluate the sampled set.
6: Submit individuals from the sampled set to SLS with probability $p_{ls}$.
7: Insert new solutions in the current population using RTR.
8: If the termination criteria are not met, go to step 2.

Figure 4.1: Pseudocode of the Hierarchical Bayesian Optimization Algorithm with substructural local search.

includes a simple example to illustrate the behavior of the Belief Propagation algorithm.

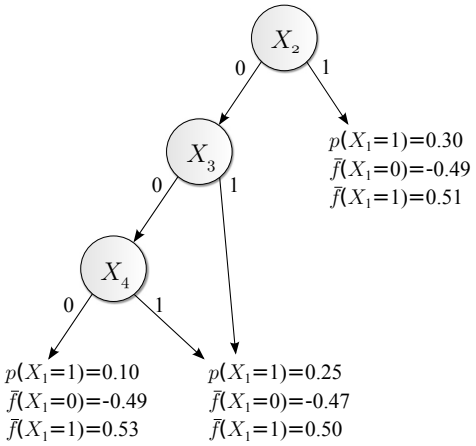
## 4.2 Substructural Local Search in hBOA

As for traditional local search methods, the substructural local search (SLS) is used to speedup the convergence to good solutions. The SLS takes place in hBOA after the offspring are sampled from the learned Bayesian network and evaluated (step 6 of Figure 4.1). The SLS is only performed for a proportion of the population, thus, each individual of the offspring is submitted to SLS with probability $p_{ls}$.

Before presenting the SLS procedure it is introduced first how to estimate the fitness of an individual in BOA, which is then needed by the SLS algorithm.

| $X_2$ | $X_3$ | $X_4$ | $p(X_1=1 \mid X_2,X_3,X_4)$ | $\bar{f}(X_1=0 \mid X_2,X_3,X_4)$ | $\bar{f}(X_1=1 \mid X_2,X_3,X_4)$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0.10 | -0.49 | 0.53 |
| 0 | 0 | 1 | 0.25 | -0.52 | 0.49 |
| 0 | 1 | 0 | 0.25 | -0.38 | 0.51 |
| 0 | 1 | 1 | 0.25 | -0.50 | 0.53 |
| 1 | 0 | 0 | 0.30 | -0.45 | 0.46 |
| 1 | 0 | 1 | 0.30 | -0.52 | 0.62 |
| 1 | 1 | 0 | 0.30 | -0.63 | 0.58 |
| 1 | 1 | 1 | 0.30 | -0.55 | 0.47 |

(a) Conditional probability table




(b) Decision graph

Figure 4.2: Example of a conditional probability table and a decision graph including fitness information.

## 4.3 Fitness estimation

Estimating the fitness of a proportion of the population instead of using the actual fitness function can lead to a reduction of the total number of function evaluations spent [25].

In order to use fitness estimation it is needed to store in each row of the CPT an average fitness of solutions with $X_i = x_i$ for each instance $\pi_i$ of $X_i$'s parents $\Pi_i$. In the binary case, two additional entries are added to each row. Figure 4.2(a) shows an example of a CPT with fitness information included.

The fitness of an individual can be estimated as

$$f_{est}(X_1, X_2, \ldots, X_\ell) = \bar{f} + \sum_{i=1}^{\ell} \bar{f}(X_i|\Pi_i), \tag{4.1}$$

where $\bar{f}$ is the average fitness of the solutions used to estimate the fitness from and $\bar{f}(X_i|\Pi_i)$ is the conditional average fitness of solutions with $X_i$. We have that

$$\bar{f}(X_i|\Pi_i) = \bar{f}(X_i, \Pi_i) - \bar{f}(\Pi_i), \tag{4.2}$$

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

When using decision graphs the average fitness is stored in each leaf. Figure 4.2(b) shows a decision graph extended with fitness information.

The solutions used to compute statistics for fitness estimation come from two sources:

1. Selected parents that were evaluated using the actual fitness function.

2. Sampled offspring that were evaluated using the actual fitness function.

This restriction is made because the Bayesian network is directly related to these two sources, it is learned from the selected parents and used to sample the offspring. The reason to restrict the computation of statistics to solutions that were evaluated using the actual fitness function is to avoid the propagation through generations of possible errors caused by the estimation of the fitness. As we will see in the next section, the fitness estimation is used for substructural local search purposes only and it is not used to estimate the fitness of individuals, thus maintaining the population free of noise.

## 4.4 SLS procedure

The procedure of the substructural local search is described in figure 4.3. The reason to use the reverse ancestral ordering is that in this way higher order

*Substructural Local Search (SLS)*

1: Select the first variable $X_i$ according to the reverse ancestral ordering of the variables in the Bayesian network.
2: Choose the values $(x_i, \pi_i)$ associated with the higher substructural fitness $\bar{f}(X_i, \Pi_i)$.
3: Set variables $(X_i, \Pi_i)$ of the individual being considered to values $(x_i, \pi_i)$ if the overall fitness of the individual is improved with the modification, otherwise leave the individual unmodified.
4: Repeat steps 2 and 3 for the remaining variables of the reverse ancestral ordering.
5: Evaluate the final individual.

Figure 4.3: Pseudocode of the substructural local search.

dependencies within the same linkage group are optimized first, thus reducing the possibility of doing incorrect decisions when considering problems whose lower-order statistics can be misleading [13].

When an individual is submitted to SLS it is evaluated several times as long as it improves (step 3). These evaluations are performed using the estimated fitness of the individual (equation 4.1) keeping the number of function evaluations unchanged. After all variables are visited, the individual is evaluated with the actual fitness function to avoid possible propagation of errors. Thus, the additional cost of performing substructural local search can be estimated as $n \cdot p_{ls}$ [13].

In [13] empirical results performed on onemax and $m$-$k$ trap functions show that SLS can reduce significantly the number of function evaluations spent. The scalability of SLS with $p_{ls} = 0.0005$ on the trap-5 with varying $m$ was analyzed and showed speedups[1] that grow approximately as $\Theta(\ell^{0.45})$. Other values for $p_{ls}$ were tested but failed to maintain the speedup for higher $m$.

[1]The speedup is the ratio of the number of evaluations required by BOA with and without local search.

Since the SLS depends on the model learned, the more accurate it is the better SLS will perform. The $s$-penalty is a modification on the penalty for the K2 metric (equation 3.6). In the standard penalty each leaf addition has associated a penalty of $0.5 \log_2(n)$. This is modified to include a factor $c_s$ that depends on the tournament size $s$, yielding

$$0.5 c_s \log_2(n). \tag{4.3}$$

$c_s = s$ was determined to be the most appropriate factor and showed to increase the model structural accuracy with increasing tournament sizes.

## 4.5 Loopy Substructural Local Search

A different local search method was also introduced in [13]. It is called *Loopy Substructural Local Search* (loopy SLS) and is based on loopy Belief Propagation (BP) [19]. The BP algorithm is usually applied to graphical models called *factor graphs* and it is mainly used for two purposes:

1. obtaining marginal probabilities for some variables (sum-product algorithm).

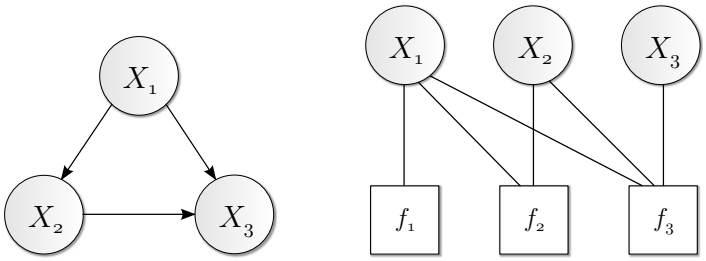
2. finding the most probable instance for the graphical model (max-product algorithm).

### 4.5.1 Factor graphs

Bayesian networks can be represented directly as factor graphs. These models explicitly express the factorization structure of the corresponding probability distribution.

Consider a function $g(X)$ whose joint probability distribution can be written as a product of factors:

$$g(x_1, x_2, \ldots, x_\ell) = \frac{1}{Z} \prod_{I \in \mathcal{F}} f_I(x_{N_I}), \tag{4.4}$$




(a) (b)

Figure 4.4: Example of a Bayesian network (a) and the corresponding representation as a factor graph (b).

where $I$ is the factor index, $N_I$ is a subset of variable indices associated with factor $I$, factor $f_I$ is a nonnegative function and $Z = \sum_x \prod_{I \in \mathcal{F}} f_I(x_{N_I})$ is a normalization constant. For a Bayesian network each factor corresponds to a conditional probability table.

A factor graph $(\mathcal{V}, \mathcal{F}, \mathcal{E})$ is a bipartite graph consisting of variable nodes $i \in \mathcal{V}$, factor nodes $I \in \mathcal{F}$, and an undirected edge $\{i, I\}$ between $i$ and $I$ if and only if $i \in N_I$, i.e., if $f_I$ depends on $x_i$. The neighbors of a factor node $f_I$ are the variables $N_I$ and the neighbors $N_i$ of a variable node $i$ are the factors that depend on that variable. Factors are usually represented as rectangles and variable nodes as circles. Figure 4.4 shows an example of a Bayesian network and an equivalent representation as a factor graph. The factor graph represents this factorization:

$$g(x_1, x_2, x_3) = \frac{1}{Z} f_1(x_1) f_2(x_1, x_2) f_3(x_1, x_2, x_3). \tag{4.5}$$

### 4.5.2 Belief propagation

Belief propagation (BP) [19] performs inference by passing messages through the nodes of the factor graph. If applied to a graphical model with cycles the method is known as *loopy belief propagation* and, in this case, the convergence to exact beliefs cannot be guaranteed as it is for acyclic graphs. Each node passes messages to its neighbors (see figure 4.5): factor nodes send messages

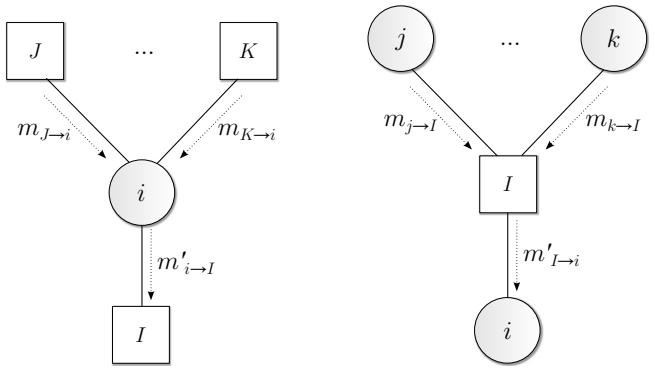



Figure 4.5: Message passing between factor and variable nodes.

$m_{I\to i}$ to variable nodes and variable nodes send messages $m_{i\to I}$ to factor nodes. The outgoing messages are functions of the received messages at each node and are given by:

$$m'_{i\to I}(x_i) = \prod_{J\in N_i\backslash I} m_{J\to i}(x_i) \qquad \forall i \in \mathcal{V}, \forall I \in N_i, \tag{4.6}$$

$$m'_{I\to i}(x_i) = \sum_{x_{N_I\backslash i}} f_I(x_{N_I}) \prod_{j\in N_I\backslash i} m_{j\to I}(x_j) \qquad \forall I \in \mathcal{F}, \forall i \in N_I, \tag{4.7}$$

where $N_i\backslash I$ is the set of neighboring factor nodes of variable $i$ excluding factor node $I$, $N_I\backslash i$ is the set of neighboring variable nodes of factor $I$ excluding variable $i$, and $x_{N_I\backslash i}$ stands for all possible combination of values that all variables but $X_i$ in $X_{N_I}$ can take while variable $X_i$ remains instantiated with $x_i$.

Equation 4.7 is used in the sum-product algorithm, whereas for the max-product algorithm the following equation is used instead:

$$m'_{I\to i}(x_i) = \max_{x_{N_I\backslash i}} \Big( f_I(x_{N_I}) \prod_{j\in N_I\backslash i} m_{j\to I}(x_j) \Big) \qquad \forall I \in \mathcal{F}, \forall i \in N_I, \tag{4.8}$$

When messages stop changing over a period of time the BP converges and the approximate marginals, also called *beliefs*, can be obtained as the normalized product of all messages received by $X_i$:

$$g_i(x_i) \propto \prod_{I\in N_i} m_{I\to i}(x_i). \tag{4.9}$$

For the max-product algorithm, it can be obtained the most probable configuration (MPC) for each variable $X_i$ by assigning the value associated with the highest probability:

$$MPC(X_i) = \arg\max_{x_i}(g_i(x_i)). \tag{4.10}$$

The initial messages are usually taken to be uniform. Messages are then sent following an *update schedule* that specifies a particular ordering of the message updates in time [16]. Some update schedules are:

- *Sequential updates.* Update messages in a fixed linear order, using the most recent message available for each update.
- *Parallel updates.* Calculate all new messages as a function of the current messages and then set all messages to their new values simultaneously.
- *Maximum residual updating.* Calculate the differences between the updated and current messages (residuals) and update only the message with the largest residual.

### 4.5.3 Loopy SLS description

The loopy SLS [13] uses the max-product algorithm to find the MPC for each substructure. It uses substructural fitness information $\bar{f}(X_i, \Pi_i)$ in the factor nodes to guide the algorithm.

Some parameters for the BP must be specified. The maximum number of iterations is $2\ell$ and the allowed difference when comparing two messages is of at least $10^{-6}$. The update schedule used is the maximum residual updating.

When translating a Bayesian network to a factor graph it is typical that the factor graph contains loops, thus the result can only be interpreted as an approximation. In this case it may happen that the algorithm does not converge to a stable state. However, the configuration found after reaching the maximum number of iterations ($2\ell$) is used as result of the loopy SLS. It may also happen that ties occur in certain positions ($g_i(x_i) = 0.5$) and

*Loopy Substructural Local Search (loopy SLS)*

1: Convert the current Bayesian network $B$ to a factor graph $F$ and store substructural fitness information $\bar{f}(X_i, \Pi_i)$ in the factor nodes.

2: Remove non-relevant factors in $F$.

3: Perform loopy belief propagation in $F$ and return the most probable configuration (MPC) and the possible number of tied positions $n_t$.

4: If $n_t = 0$, instantiate an individual with the values of the MPC.
Else, if $2^{n_t} \leq \ell$, instantiate $2^{n_t}$ individuals, one for each of the $2^{n_t}$ possible configurations.
Else, instantiate $\ell$ individuals at random from all $2^{n_t}$ possible configurations.

5: Evaluate the resulting individuals.

Figure 4.6: Pseudocode of the loopy substructural local search.

the MPC cannot distinguish between a 0 and a 1. When this happens all possible configurations are enumerated and inserted in the population. If the number of ties ($n_t$) is such that the number of possible configurations ($2^{n_t}$) is bigger than $\ell$, then $\ell$ configurations are chosen at random.

Another feature of the loopy SLS is the selection of relevant factors before performing the local search. Factor nodes (and corresponding edges) whose variable set is a subset of another factor are removed. In the example of figure 4.4 the relevant factor is factor $f_3$ because the variable sets of factors $f_1$ and $f_2$ are already included in the variable set of $f_3$. This procedure simplifies and improves the information exchange especially when the lower-order statistics can be misleading.

The loopy SLS pseudocode is presented in figure 4.6 and it takes place between steps 4 and 5 on the pseudocode of figure 3.7.

The differences between the loopy SLS and the standard loopy BP are two: (1) loopy SLS uses substructural fitness information instead of conditional probabilities for the factor nodes and (2) removes non-relevant factors.

The loopy SLS was tested in several problems in [13]. Its performance was

compared with the standard loopy BP in trap-5 functions with two overlapping variables. The loopy SLS required bigger populations but in exchange this lead to gathering more accurate fitness information and consequently to converge faster to optimal solutions. It showed increasing speedups with increasing problem size whereas the speedups of the standard loopy BP were very close to 1.

The loopy SLS was also tested with 1 and 3 overlapping variables and showed also increasing speedups although with $o3$ the speedups were much lower than with $o1$.

In non-overlapping trap-5 the loopy SLS and the SLS had a similar performance. Both algorithms perform also similarly in the hierarchical trap-3 function although in this case they showed no advantage over the traditional BOA.

The next subsection presents and example of the belief propagation algorithm using fitness information in the factor nodes.

### 4.5.4 Belief propagation example

In this example we take the factor graph from figure 4.4(b). After removing non-relevant factors we get the factor graph shown in figure 4.7. The fitness information used by factor $f_3$ is presented in the following table:

| $X_1$ | $X_2$ | $X_3$ | $f_3(X_1, X_2, X_3)$ |
|---|---|---|---|
| 0 | 0 | 0 | 2 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 3 |

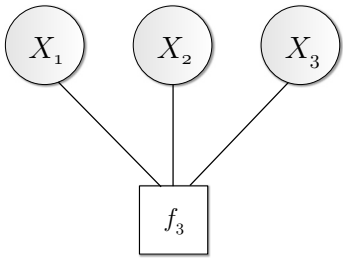



Figure 4.7: The resultant factor graph after removing non-relevant factors in figure 4.4.

The values in the table are the fitness of a trap-3 function.

The initial messages, sent from variable nodes to the factor node (equation 4.6) are uniformly initialized:

$$
\begin{aligned}
&m_{1\to f_3}(x_1=0)=0.5, \qquad && m_{1\to f_3}(x_1=1)=0.5,\\
&m_{2\to f_3}(x_2=0)=0.5, \qquad && m_{2\to f_3}(x_2=1)=0.5,\\
&m_{3\to f_3}(x_3=0)=0.5, \qquad && m_{3\to f_3}(x_3=1)=0.5.
\end{aligned}
$$

The factor $f_3$ receives all messages and propagates them including information from its neighbors, note that the max-product equation is used in the calculations (equation 4.8):

$$
\begin{aligned}
&m_{f_3\to 1}(x_1=0)=\\
&=\max\Big(f_3(x_1=0,x_2=0,x_3=0)\times m_{2\to f_3}(x_2=0)\times m_{3\to f_3}(x_3=0),\\
&\qquad f_3(x_1=0,x_2=0,x_3=1)\times m_{2\to f_3}(x_2=0)\times m_{3\to f_3}(x_3=1),\\
&\qquad f_3(x_1=0,x_2=1,x_3=0)\times m_{2\to f_3}(x_2=1)\times m_{3\to f_3}(x_3=0),\\
&\qquad f_3(x_1=0,x_2=1,x_3=1)\times m_{2\to f_3}(x_2=1)\times m_{3\to f_3}(x_3=1)\Big)=\\
&=\max\Big(2\times 0.5\times 0.5, 1\times 0.5\times 0.5, 1\times 0.5\times 0.5, 0\times 0.5\times 0.5\Big)=\\
&=\max\Big(0.5, 0.25, 0.25, 0\Big)=0.5.
\end{aligned}
$$

$$m_{f_3\to 1}(x_1 = 1) = \max\Big(1 \times 0.5 \times 0.5, 0, 0, 3 \times 0.5 \times 0.5\Big) =$$
$$= \max\Big(0.25, 0, 0, 0.75\Big) = 0.75.$$

Normalizing the messages $m_{f_3\to 1}(x_1)$ to sum to one we get:

$$m_{f_3\to 1}(x_1 = 0) = 0.4, \qquad m_{f_3\to 1}(x_1 = 1) = 0.6.$$

Making analogous calculations for $m_{f_3\to 2}(x_2)$ and $m_{f_3\to 3}(x_3)$ and after normalizing, we get the following:

$$m_{f_3\to 2}(x_2 = 0) = 0.4, \qquad m_{f_3\to 2}(x_2 = 1) = 0.6.$$
$$m_{f_3\to 3}(x_3 = 0) = 0.4, \qquad m_{f_3\to 3}(x_3 = 1) = 0.6.$$

The most probable configuration (MPC) for each variable is therefore:

$$MPC(X_1) = 1, \qquad MPC(X_2) = 1, \qquad MPC(X_3) = 1.$$

The individual 111 would be instantiated and inserted in the population as result of the loopy SLS.

## 4.6 Summary

This chapter presents two methods that take advantage of the structure learned by hBOA. They perform local search in each of the substructures identified and in this way they are able to speedup the convergence to good solutions. The first method is the Substructural Local Search (SLS). Using fitness estimation it is able to gather substructural fitness information that is used by the local searcher to evaluate competing substructures. The *s*-penalty is also introduced. It is a modification in the penalty of the K2 metric that takes into account the tournament size used and helps to improve the model structural accuracy.

The second method, the loopy SLS, is based on loopy Belief Propagation. This algorithm returns the most probable configuration (MPC) of a given

graphical model and is usually applied to factor graphs. Bayesian networks are converted directly to factor graphs and, after identifying and removing non-relevant factors, the belief propagation algorithm is ran, with substructural fitness information included in the factors. Messages are passed between the factors and the variables of the factor graph until convergence or until a specified number of iterations is attained. These messages change the *beliefs* of each variable about its own states, taking into account the *beliefs* of their neighbors.

The chapter ends with an example to illustrate the behavior of the Belief Propagation algorithm.

# Chapter 5

# Substructural Local Search in the MAXSAT problem

## 5.1 Introduction

The two local searchers introduced in the previous chapter (*SLS* and *loopy SLS*) have shown to reduce the number of function evaluations needed to achieve the optimum in BOA algorithm in a set of artificial problems. Testing on those artificial problems is important because they present different bounds of difficulty and if the algorithm is able to overcome those difficulties then it is expected to solve real problems that are bounded by the same difficulty.

In this chapter the utility of the substructural local searchers is investigated when applied to the MAXSAT problem. In addition to SLS and loopy SLS, a simple hill climber is also included in the comparisons.

## 5.2 MAXSAT

MAXSAT (Maximum Satisfiability) is an important problem of complexity theory and artificial intelligence [20]. The task is to find an interpretation of propositions for a propositional logic formula expressed in conjunctive

normal form, that maximizes the number of satisfied clauses. Formulas in conjunctive normal form with clauses of length at most $k$ are called $k$-CNF formulas. A CNF is a *logical and* of clauses, where each clause is a *logical or* of literals. Each literal can be a proposition or a negation of a proposition. An example of a 3-CNF formula with 2 clauses and propositions $X_1$, $X_2$, $X_3$, $X_4$ follows:

$$(X_1 \vee X_2 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee \neg X_4). \tag{5.1}$$

The interpretation of propositions ($X_1 = true$, $X_2 = false$, $X_3 = true$, $X_4 = false$) satisfies both clauses, thus is one of the optima for this example. In fact, any interpretation with $X_1 = true$ would be an optimum. MAXSAT is NP-complete for $k$-CNF with $k \geq 2$ [20].

In GA and EDA context, the representation of an individual is given by a binary string where each position of the string corresponds to one proposition of the formula, a 1 represents a *true* assignment and a 0 a *false* assignment. The size of the individual is equal to the number of propositions (or variables). The individual for the previous interpretation is therefore 1010. The fitness value corresponds to the number of clauses that are satisfied.

### 5.2.1 Tested instances

The MAXSAT instances used in the experiments were obtained from the Satisfiability Library SATLIB[1] and are listed in table 5.1. From each package 10 instances were randomly selected and are listed in the last column of the table. The order of the instances corresponds to the order shown latter in the results. All the instances considered are 3-CNF formulas and are all satisfiable, i.e., all clauses can be satisfied.

[1] http://www.satlib.org

Table 5.1: List of instances used in the experiments.

| Package | Variables | Clauses | Total instances | Randomly selected instances | |
|---|---|---|---|---|---|
| uf20-91.tar.gz | 20 | 91 | 1000 | 1: uf20-0537.cnf | 6: uf20-0251.cnf |
| | | | | 2: uf20-0846.cnf | 7: uf20-0558.cnf |
| | | | | 3: uf20-0326.cnf | 8: uf20-0596.cnf |
| | | | | 4: uf20-0235.cnf | 9: uf20-0183.cnf |
| | | | | 5: uf20-0360.cnf | 10: uf20-0718.cnf |
| uf50-218.tar.gz | 50 | 218 | 1000 | 1: uf50-0322.cnf | 6: uf50-0405.cnf |
| | | | | 2: uf50-013.cnf | 7: uf50-0552.cnf |
| | | | | 3: uf50-0906.cnf | 8: uf50-0611.cnf |
| | | | | 4: uf50-0838.cnf | 9: uf50-0731.cnf |
| | | | | 5: uf50-0955.cnf | 10: uf50-0534.cnf |
| uf75-325.tar.gz | 75 | 325 | 100 | 1: uf75-0100.cnf | 6: uf75-063.cnf |
| | | | | 2: uf75-044.cnf | 7: uf75-078.cnf |
| | | | | 3: uf75-065.cnf | 8: uf75-092.cnf |
| | | | | 4: uf75-051.cnf | 9: uf75-038.cnf |
| | | | | 5: uf75-029.cnf | 10: uf75-035.cnf |
| uf100-430.tar.gz | 100 | 430 | 1000 | 1: uf100-0473.cnf | 6: uf100-0157.cnf |
| | | | | 2: uf100-0422.cnf | 7: uf100-0957.cnf |
| | | | | 3: uf100-050.cnf | 8: uf100-0541.cnf |
| | | | | 4: uf100-0926.cnf | 9: uf100-0401.cnf |
| | | | | 5: uf100-0770.cnf | 10: uf100-0114.cnf |

## 5.3 Bisection method

In order to determine a population size that is neither too small nor too large, the bisection method [20] is used. The bisection method is somewhat equivalent to a kind of "binary search" method for finding a population size that provides reliable convergence to a certain solution quality. The method starts with an arbitrary population size, e.g., 10000, in order to determine the initial lower and upper bounds. A population size is said to be successful if the optimum is found in 10 independent runs using that size.

If the initial population size succeeds, then smaller ones should be tested to find one more adequate. The population size is divided by 2 until it fails, being the lower bound the last population size and the upper bound the last successful one. On the other hand, if the initial population size fails, then it was not sufficient and a bigger one is needed. In this case, the population is doubled until it succeeds, and when it does, the upper bound is the successful

*Bisection method*

```
/* Determine initial lower and upper bounds */

n = 10000
success = optimum found in 10 independent runs?
if success then
   while success do
      n = n/2
      success = optimum found in 10 independent runs?
   lower = n
   upper = n × 2
else
   while not success do
      n = n × 2
      success = optimum found in 10 independent runs?
   lower = n/2
   upper = n

/* Optimize population size between lower and upper */
while ((upper − lower)/lower ≥ 0.10) do
   n = (upper + lower)/2
   success = optimum found in 10 independent runs?
   if not success then
      lower = n
   else
      upper = n

return upper
```

Figure 5.1: Pseudocode of the bisection method.

population size and the lower bound is the previous one tested.

Once the initial bounds are determined, the population size is optimized between those values until it is at most 10% more than the minimum population size required to ensure that the algorithm converges in all the 10 runs.

The pseudocode of the bisection method is presented in figure 5.1.

## 5.4 Experimental setup

To test the utility of the substructural local search in the hBOA algorithm, five different combinations are tested: (1) hBOA alone with no local search, (2) hBOA with SLS, (3) hBOA with loopy SLS, (4) hBOA with a hill climber (HC) and (5) hBOA with the HC and with loopy SLS.

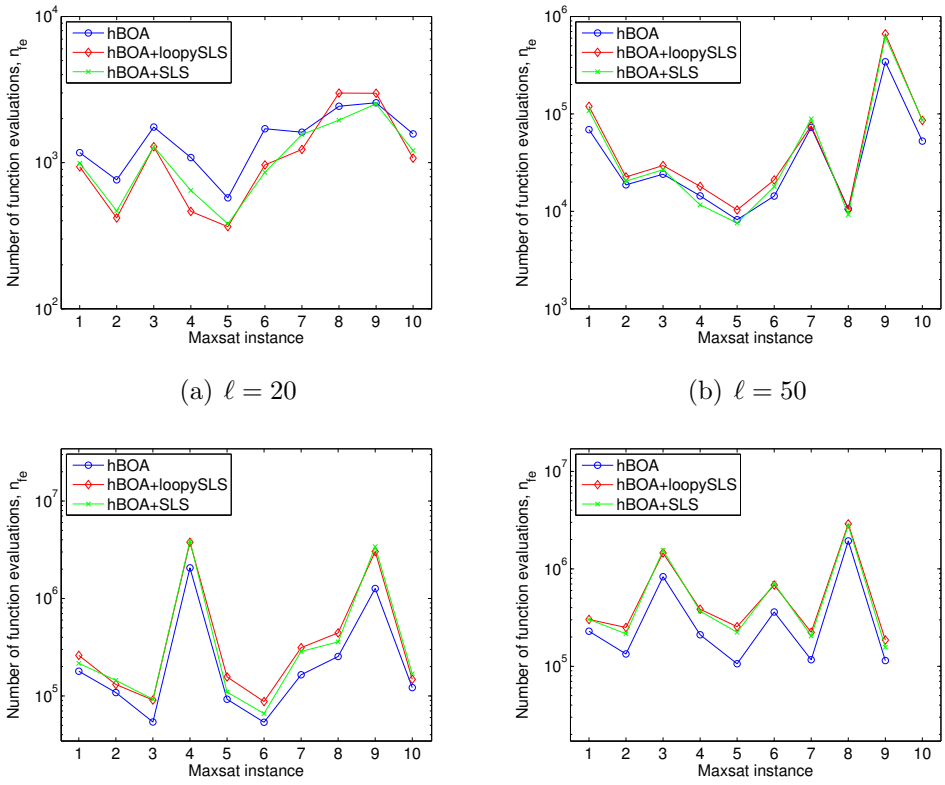
The hill climber is performed by flipping the bit of the individual that improves its fitness the most until no more improvement is possible. Note that the actual fitness function is used in each bit flip. The HC is applied to every individual before it is submitted to evaluation.

For each algorithm, and for each MAXSAT instance, 5 bisection runs are performed and in each one the optimum is required to be found in 10 independent runs. The results are averaged over 50 ($5 \times 10$) runs. The selection operator used is the tournament without replacement with size $s = 2$. The metric used to evaluate the Bayesian network is the K2 metric with penalty. In algorithms that use substructural local search (2, 3 and 5) the s-penalty is used instead.

## 5.5 Results

Figures 5.2 and 5.3 show each one 4 plots, one for each problem size tested. Each plot shows the number of function evaluations[2] $n_{fe}$ spent to achieve the

[2]Number of function evaluations is the number of times that the objective function is calculated.




(a) $\ell = 20$ (b) $\ell = 50$

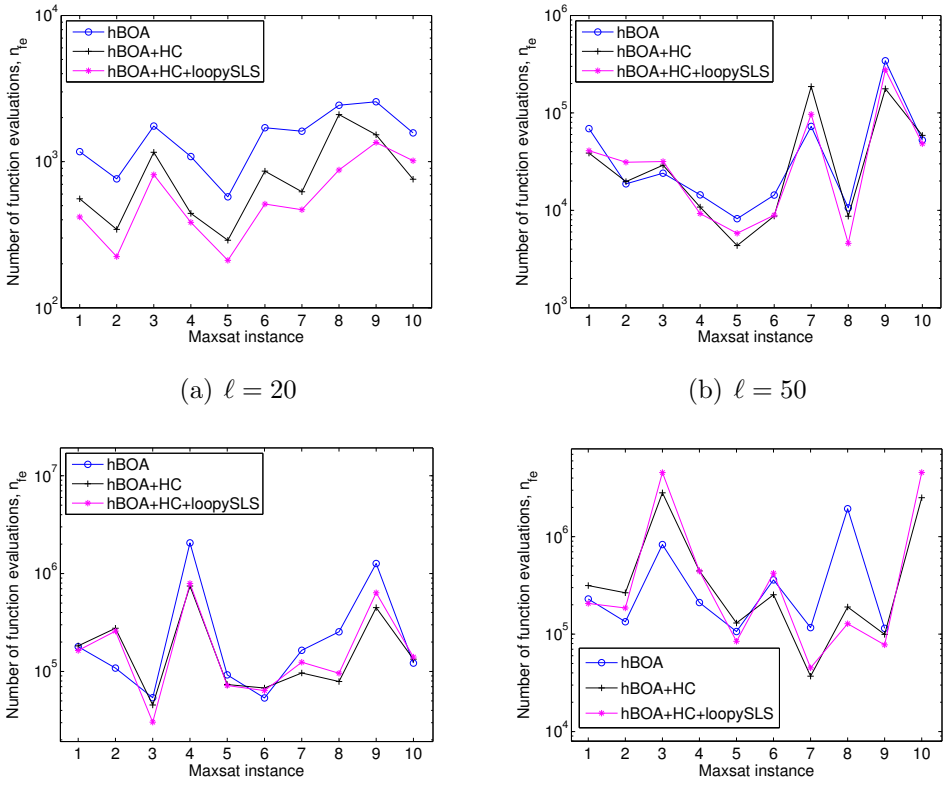
(c) $\ell = 75$ (d) $\ell = 100$

Figure 5.2: Number of function evaluations required to find the optimum for hBOA, hBOA+SLS and hBOA+loopySLS on the MAXSAT problem with different problem sizes $\ell$.

optimum for each tested instance using the different algorithms mentioned before. The lower the $n_{fe}$ the better the algorithm. The plots are separated in two figures for a better visualization of the results.

The performance of hBOA+SLS and hBOA+loopySLS is compared to that of hBOA without any local search in figure 5.2. For size 20, which is rather small, both local search algorithms succeed in reducing the $n_{fe}$ in most cases. For bigger sizes this is not accomplished. In this case the performance of hBOA decreases when including SLS or loopySLS[3].

[3]When testing on instance 100.10 none of the three algorithms was able to finish the bisection method after many days of computation.




(a) $\ell = 20$ (b) $\ell = 50$

(c) $\ell = 75$ (d) $\ell = 100$

Figure 5.3: Number of function evaluations required to find the optimum for hBOA, hBOA+HC and hBOA+HC+loopySLS on the MAXSAT problem with different problem sizes $\ell$.

Figure 5.3 compares hBOA+HC, hBOA+HC+loopySLS and hBOA alone. For size 20 both local search algorithm succeed also in reducing the $n_{fe}$. hBOA+HC+loopySLS is in this case the algorithm with best performance in this size. For medium sizes, 50 and 75, the additions of HC and of HC+SLS are able to reduce the $n_{fe}$ in more than half of the instances but in the rest the performance of hBOA decreases. For size 100 the number of instances where the performance of hBOA decreases is about the same as the number of instances where it increases.

Figure 5.4 shows the average $n_{fe}$ for each problem size, for a global vi-

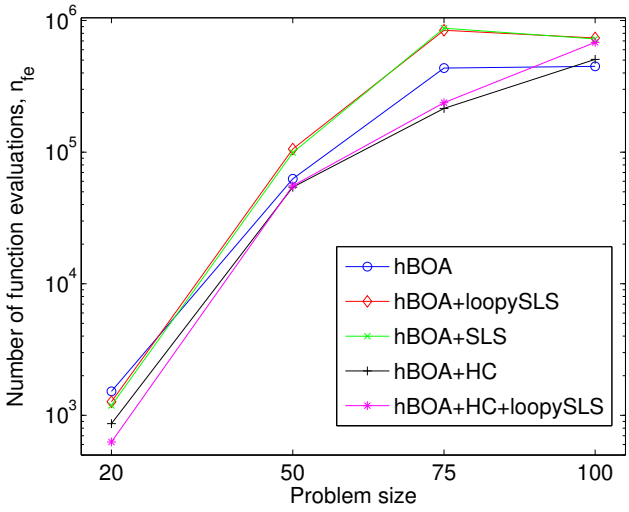


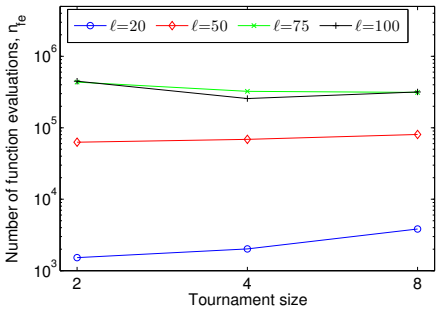

Figure 5.4: Average number of function evaluations for each problem size tested.

sualization of the results[4]. The algorithms that include HC have on average the lower $n_{fe}$.

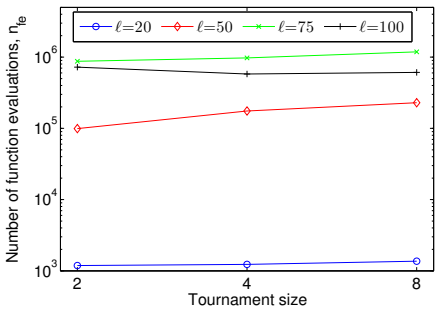
In addition to the previous experiments, another ones were made using different parameters for the algorithms. Different tournament sizes, $s = 2$, $s = 4$ and $s = 8$, were tested in the same instances. The results are summarized in figure 5.5. Each plot shows the average $n_{fe}$ with increasing tournament sizes for each algorithm. Although in a few cases the $n_{fe}$ is reduced, in most cases, the $n_{fe}$ either maintains or increases with increasing tournament size.

It was also made a comparison between K2 and BIC metrics, both with $s = 2$. Figure 5.6 shows the results as before in terms of average $n_{fe}$. The performance is not affected too much when using one metric or the other.

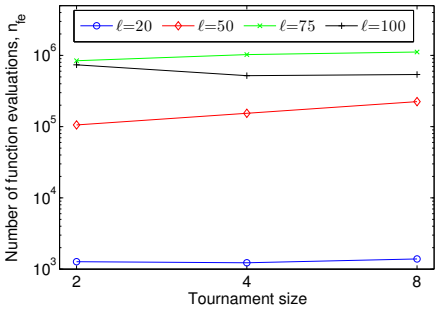
[4] $n_{fe}$ of instance 100.10 is not included in the average to allow a fair comparison.



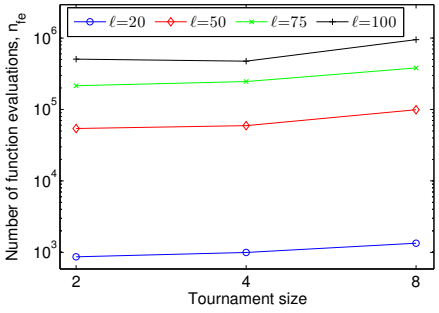

(a) hBOA



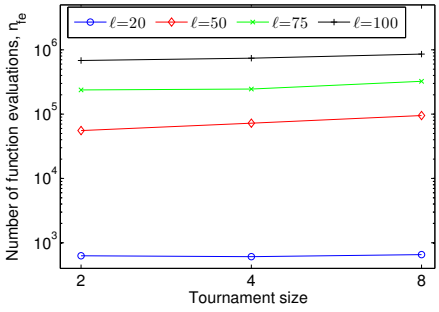

(b) hBOA+SLS




(c) hBOA+loopySLS

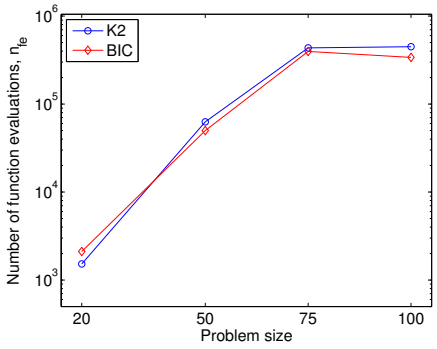



(d) hBOA+HC

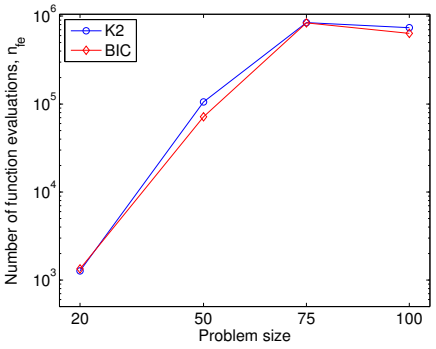



(e) hBOA+HC+loopySLS

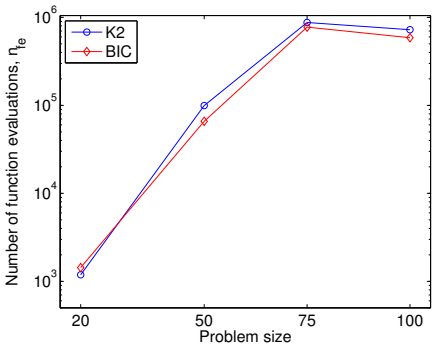
Figure 5.5: Number of function evaluations required to find the optimum using different tournament sizes in each algorithm.



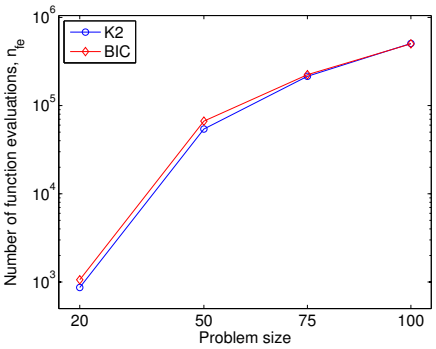

(a) hBOA



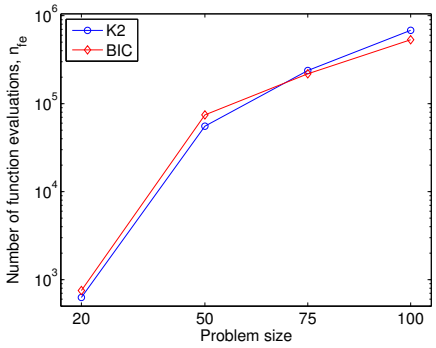

(b) hBOA+loopySLS




(c) hBOA+SLS

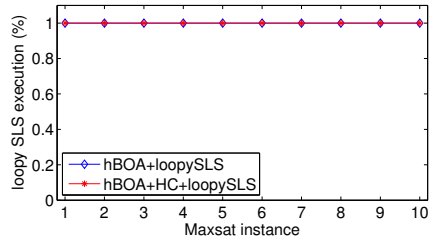



(d) hBOA+HC

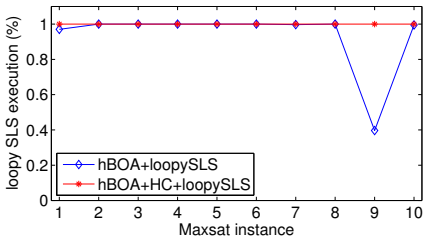



(e) hBOA+HC+loopySLS

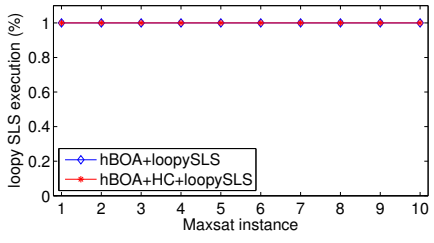
Figure 5.6: Number of function evaluations required to find the optimum using different metrics: K2 and BIC.



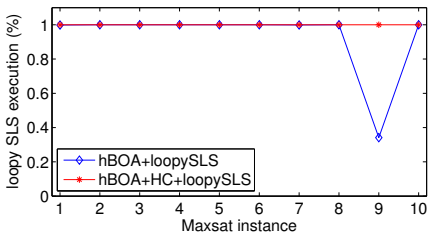

(a) $s = 2$, $\ell = 20$



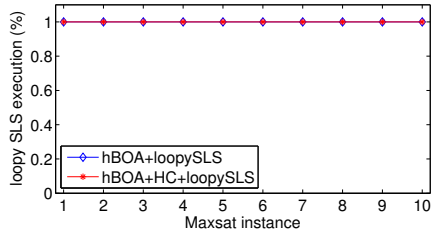

(b) $s = 2$, $\ell = 50$



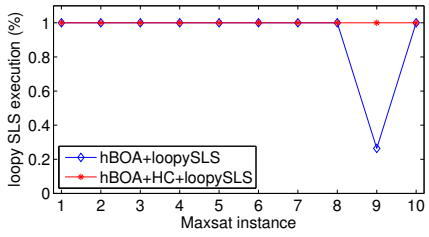

(c) $s = 4$, $\ell = 20$




(d) $s = 4$, $\ell = 50$




(e) $s = 8$, $\ell = 20$

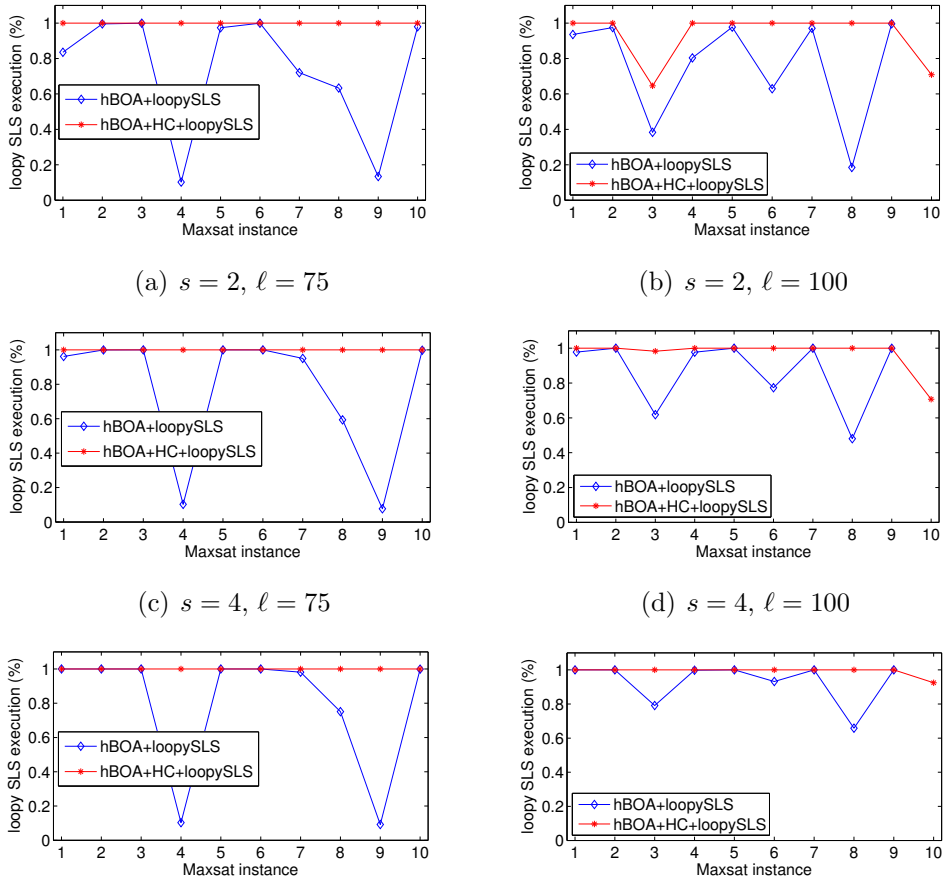



(f) $s = 8$, $\ell = 50$

Figure 5.7: Number of times (in percentage) that loopy SLS was executed in hBOA+loopySLS and hBOA+HC+loopySLS for problem sizes 20 and 50 using different tournament sizes.

### 5.5.1 Loopy SLS analysis

For efficiency reasons, the execution of loopy SLS is conditioned to the number of neighbors of the factors. If this number is too large then loopy SLS is not performed. The maximum number of neighbors allowed was set to 14. Figures 5.7 and 5.8 show the number of times (in percentage) that loopy SLS was executed in hBOA+loopySLS and hBOA+HC+loopySLS algorithms, using different tournament sizes. It can be observed that for sizes 20 and




(a) $s = 2$, $\ell = 75$ (b) $s = 2$, $\ell = 100$

(c) $s = 4$, $\ell = 75$ (d) $s = 4$, $\ell = 100$

(e) $s = 8$, $\ell = 75$ (f) $s = 8$, $\ell = 100$

Figure 5.8: Number of times (in percentage) that loopy SLS was executed in hBOA+loopySLS and hBOA+HC+loopySLS for problem sizes 75 and 100 using different tournament sizes.

50, loopy SLS was executed in almost all cases. In sizes 75 and 100 the percentage decreases for hBOA+loopySLS although for hBOA+HC+loopySLS it stays in 100% in most cases. Looking from top to bottom in figure 5.8, we can also see that as the tournament size increases, the execution of loopy SLS gets closer to 100%.

When increasing the tournament size, and therefore the selection pressure, the search of the GA gets more focused on a specific area of the search space, thus, the learned model gets more specialized (less general) and it encodes

only the dependencies that are more relevant to that area of the search space. The problem of getting more specialized is that diversity decreases. The reduction of encoded dependencies is the reason for the increase of execution of loopy SLS with increasing tournament size.

This agrees with the observations made in [13] where, when using the s-penalty, the model structural accuracy increased with increasing tournament size. This was, in fact, the motivation to test with different tournament sizes.

The inclusion of HC makes each individual to be a local optimum, thus, the specialization of the model happens from the first generation. Since the model has less dependencies between nodes, loopy SLS is executed 100% of the time.

One more observation can be made from these plots. If we compare sizes 75 and 100 of figure 5.8 with $s = 2$ with those of figure 5.2 we can see that harder instances (the ones with higher $n_{fe}$) produced more complex models, because they have lower percentages of loopy SLS execution. In fact, looking at the shape of the plots of figures 5.8(a) and 5.8(b), if we flip them vertically, we can see a close match between them and those of figures 5.2(a) and 5.2(b) respectively.

## 5.6 Discussion

Results showed that substructural local search only reduced the number of function evaluations ($n_{fe}$) in MAXSAT problems of small size. Some observations are presented to try to explain such behavior.

Substructural local search depends on two factors: (1) the quality of the structure used to guide the local search and (2) the quality of the fitness estimation.

In the previous subsection we saw that the quality of the structure increased with the tournament size when using the s-penalty. But the increase in model quality did not translated into a decrease of $n_{fe}$.

On the other hand, the estimated fitness used, must be accurate enough to

allow the local search to evaluate correctly the substructures. Another fitness estimator [27] that uses linear regression methods could be used instead. This method, as opposed to the one used in the experiments, obtains better results on noisy problems and on hierarchically decomposable problems [27].

We have to note also that SLS and loopy SLS had difficulties in artificial problems where high order overlapping was present and also in hierarchical problems where they brought little or no advantage over the traditional BOA. If these difficulties are present in the MAXSAT instances tested it would explain why when increasing the percentage of execution of loopy SLS (by increasing the tournament size and/or by using HC+loopySLS) this did not reflected a decrease of $n_{fe}$.

From an overall perspective, none of the local search methods applied to hBOA reduce significantly the number of function evaluations on the MAXSAT instances tested. Therefore, more competence is needed when performing local search, to be able to achieve this goal.

## 5.7 Summary

This chapter studies the utility of SLS and loopy SLS when applied to hBOA in the MAXSAT problem. The utility of a simple hill climber (HC) is studied as well.

Several instances of the MAXSAT problem with 20, 50, 75 and 100 variables are tested. Algorithms hBOA, hBOA+SLS, hBOA+loopySLS, hBOA+HC and hBOA+HC+loopySLS are compared in terms of number of function evaluations ($n_{fe}$) required to achieve the optimum in each of the tested instances.

The goal is to reduce the $n_{fe}$ spent by hBOA by applying local search. Only with instances of size 20 the algorithms are able to reduce the $n_{fe}$, being in this size hBOA+HC+loopySLS the algorithm with better performance. For bigger sizes only algorithms that included HC are able able to show some improvement, although not too significant.

Different tournament sizes and metrics are tested but with similar results in terms of $n_{fe}$ compared to the first setup.

Additionally, the behavior of loopy SLS is analyzed by recording the number of times that it is executed. For efficiency reasons, loopy SLS is not performed when the number of neighbors of the factors grow too large. It is observed that when using hBOA+HC+loopySLS, loopy SLS is executed almost 100% of the time but when using just hBOA+loopySLS its execution is reduced for harder instances. It is also observed that the number of encoded dependecies is reduced when increasing the tournament size, providing a more accurate model.

Finally, some observations are presented to provide some explanation about the results obtained. If high-order overlapping is present in the MAXSAT instances tested, then it is expected that SLS and loopy SLS do not improve the search because they have some difficulty when solving problems where it is present. On the other hand, another fitness estimator that uses linear regression methods could be used to investigate its influence in this problem. That fitness estimator obtains better results in noisy problems and on hierarchically decomposable problems than the one used in the experiments. Investigation must continue to overcome these difficulties and provide more robust substructural local search.

# Chapter 6

# Future work and conclusions

## 6.1 Future work

Some topics suggested for future work follow:

- **Test on different real problems.** Testing on one real problem is not sufficient to draw final conclusions about the utility of substructural local search. The type of interactions between variables may vary from problem to problem, therefore, more experiments on different real problems are needed.

- **Improve the efficiency of substructural local search.** In previous works [13] loopy SLS showed positive speedups in trap functions with overlapping but the speedup decreased with the degree of overlapping. It should be investigated how to maintain speedups with high-order overlapping interactions and also to improve the utility of SLS and loopy SLS on hierarchical problems. In this way, the optimization of problems bounded by this difficulty should converge faster when incorporating substructural local search.

- **Experiment with other fitness estimator.** Another fitness estimation method [27] has shown better results in noisy problems and

hierarchically decomposable problems. It should be applied in substructural local search and compared its utility with the method used in this thesis.

## 6.2 Conclusions

This thesis studies the utility of substructural local search (SLS) and loopy SLS in the hierarchical Bayesian optimization algorithm (hBOA) when applied to MAXSAT problem instances. Besides SLS and loopy SLS, a simple hill climber (HC) and a combination of it whit loopy SLS (HC+loopySLS) are also included in the comparisons.

The local searchers succeed in reducing the number of function evaluations ($n_{fe}$) only for problem instances with a small number of variables. For bigger instances, SLS and loopy SLS decrease the performance of hBOA and in average the algorithms that include HC have the lower $n_{fe}$.

Experiments with different tournament sizes and with different metrics show results similar to those obtained with the first setup. Nevertheless, the increase of the tournament size has shown to influence the construction of a simpler model by hBOA in this problem as observed before in [13] for artificial problems. Results show that the inclusion of HC is also responsible for generating simpler models. If more robust substructural local search is developed and is able to improve the performance of hBOA in the MAXSAT problem, then changing these parameters (tournament size and including HC) should be considered and it is expected to benefit the performance of the substructural local search.

# Bibliography


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

[2] D. M. Chickering. Learning bayesian networks is NP-complete. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag, 1996.

[3] J. S. De Bonet, C. L. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 424–430. The MIT Press, Cambridge, 1997.

[4] K. Deb and D. E. Goldberg. Analyzing deception in trap functions. *Foundations of Genetic Algorithms 2*, pages 93–108, 1993.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[6] D. E. Goldberg. Genetic and evolutionary algorithms in the real world. IlliGAL. Report No. 99013, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, 1999.

[7] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253, 1999.

[8] G. Harik, F. G. Lobo, and K. Sastry. Linkage learning via probabilistic modeling in the ECGA. In M. Pelikan, K. Sastry, and E. Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, pages 39–61. Springer, 2006.

[9] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287–297, 1999.

[10] D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[11] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Bradford Books. MIT Press, 1992.

[12] K. Korb and A. E. Nicholson. *Bayesian Artificial Intelligence*. Chapman & Hall /CRC, 2003.

[13] C. F. Lima. *Substructural Local Search in Discrete Estimation of Distribution Algorithms*. PhD thesis, Universidade do Algarve, 2009.

[14] C. F. Lima, M. Pelikan, F. G. Lobo, and D. E. Goldberg. Loopy substructural local search for the bayesian optimization algorithm. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, Second International Workshop, SLS 2009*, volume 5752 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2009.

[15] C. F. Lima, M. Pelikan, K. Sastry, M. V. Butz, D. E. Goldberg, and F. G. Lobo. Substructural neighborhoods for local search in the bayesian optimization algorithm. In *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, volume 4193 of *Lecture Notes in Computer Science*, pages 232–241. Springer, 2006.

[16] J. M. Mooij. *Understanding and Improving Belief Propagation.* PhD thesis, Radboud University Nijmegen, May 2008.

[17] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In H.-M. Voigt et al., editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 178–187, Berlin, 1996. Kluwer Academic Publishers.

[18] R. E. Neapolitan. *Learning Bayesian Networks.* Prentice Hall, 2003.

[19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[20] M. Pelikan. *Hierarchical Bayesian Optimization Algorithm - Toward a New Generation of Evolutionary Algorithms*, volume 170 of *Studies in Fuzziness and Soft Computing.* Springer, 2005.

[21] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 511–518. Morgan Kaufmann, 2001.

[22] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian Optimization Algorithm. In W. Banzhaf et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, pages 525–532, San Francisco, CA, 1999. Morgan Kaufmann.

[23] M. Pelikan, D. E. Goldberg, and E. Cantú-paz. Linkage problem, distribution estimation, and bayesian networks. *Evol. Comput.*, 8(3):311–340, September 2000.

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

[25] M. Pelikan and K. Sastry. Fitness Inheritance in the Bayesian Optimization Algorithm. In *Genetic and Evolutionary Computation Conference, GECCO-2004*, volume 3103 of *Lecture Notes in Computer Science*, chapter 5, pages 48–59. Springer, Berlin, Heidelberg, 2004.

[26] K. Sastry and D. E. Goldberg. Modeling Tournament Selection With Replacement Using Apparent Added Noise. In *Intelligent Engineering Systems Through Artificial Neural Networks*, volume 11, pages 129–134, 2001.

[27] K. Sastry, C. F. Lima, and D. E. Goldberg. Evaluation relaxation using substructural information and linear estimation. In *In Keijzer, M., et al. (Eds.), Proceedings of the ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2006)*, pages 419–426. ACM Press, 2006.

[28] G. Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, 1978.

[29] D. Thierens and D. E. Goldberg. Mixing in genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 38–45, San Mateo, CA, 1993. Morgan Kaufmann.