UNIVERSIDADE DO ALGARVE

Substructural Local Search in Discrete Estimation of Distribution Algorithms

Cláudio Miguel Faleiro de Lima

Doutoramento em Engenharia Electrónica e Computação,
Especialidade em Ciências de Computação

2009

UNIVERSIDADE DO ALGARVE

Substructural Local Search in Discrete Estimation of Distribution Algorithms

Cláudio Miguel Faleiro de Lima

Tese orientada por
Fernando Miguel Pais da Graça Lobo

Doutoramento em Engenharia Electrónica e Computação,
Especialidade em Ciências de Computação

2009

Abstract

The last decade has seen the rise and consolidation of a new trend of stochastic optimizers known as estimation of distribution algorithms (EDAs). In essence, EDAs build probabilistic models of promising solutions and sample from the corresponding probability distributions to obtain new solutions. This approach has brought a new view to evolutionary computation because, while solving a given problem with an EDA, the user has access to a set of models that reveal probabilistic dependencies between variables, an important source of information about the problem.

This dissertation proposes the integration of substructural local search (SLS) in EDAs to speedup the convergence to optimal solutions. Substructural neighborhoods are defined by the structure of the probabilistic models used in EDAs, generating adaptive neighborhoods capable of automatic discovery and exploitation of problem regularities. Specifically, the thesis focuses on the extended compact genetic algorithm and the Bayesian optimization algorithm. The utility of SLS in EDAs is investigated for a number of boundedly difficult problems with modularity, overlapping, and hierarchy, while considering important aspects such as scaling and noise. The results show that SLS can substantially reduce the number of function evaluations required to solve some of these problems. More importantly, the speedups obtained can scale up to the square root of the problem size $O(\sqrt{\ell})$.

Resumo

A última década tem assistido ao aparecimento e consolidação de uma nova tendência de optimizadores estocásticos conhecidos por algoritmos da estimação da distribuição (AEDs). Essencialmente, os AEDs constroem modelos probabilísticos das soluções promissoras e amostram das correspondentes distribuições de probabilidade para obter novas soluções. Esta abordagem trouxe uma nova visão à computação evolutiva porque, enquanto um determinado problema é resolvido com um AED, o utilizador tem acesso a um conjunto de modelos que revelam dependências probabilísticas entre variáveis, uma importante fonte de informação sobre o problema.

Esta dissertação propõe a integração de pesquisa local sub-estrutural (PLS) em AEDs para acelerar a convergência para soluções óptimas. As vizinhanças sub-estruturais são definidas pela estrutura dos modelos probabilísticos utilizados nos AEDs, dando origem a vizinhanças adaptáveis, capazes de descobrirem automáticamente e explorem as regularidades do problema. Especificamente, esta tese foca-se no algoritmo genético compacto estendido e no algoritmo de optimização Bayesiana. A utilidade da PLS nos AEDs é investigada para vários problemas de dificuldade limitada com modularidade, sobreposição, e hierarquia, considerando também aspectos como o escalonamento e o ruído. Os resultados mostram que a PLS pode reduzir substancialmente o número de avaliações da função objectivo necessárias para resolver alguns destes problemas. Mais importante, é o facto do

speedup obtido crescer no melhor caso de acordo com a raíz quadrada do tamanho do problema $O(\sqrt{\ell})$.

Acknowledgments

First of all, I would like to thank my advisor Fernando G. Lobo for his guidance, support, and friendship. From day one, I found his enthusiasm about genetic algorithms, and research in general, contagious. With him, I have learned many things concerning research, writing, organization, and life in general. I feel privileged to have him as my advisor.

I am also very grateful to Prof. David E. Goldberg for his guidance and for accepting my several visits to the Illinois genetic algorithms laboratory (IlliGAL). Our weekly meetings were a constant source of inspiration and advice for me. His expertise and ability to see the big picture of things is remarkable.

I am very thankful to Martin Pelikan, who I first met at IlliGAL. Martin did his dissertation on the Bayesian optimization algorithm and his advice in this area was priceless. Some of the ideas presented in this thesis resulted from discussions with him. I also would like to thank him and Mark Hauschild for receiving me so well in my short stay at MEDAL in St. Louis.

I am similarly grateful to Kumara Sastry, former student lab director of IlliGAL, who has helped me a lot on my first stays at Urbana-Champaign. During my first stay at IlliGAL, I sat right behind him and interact with him on a daily basis. This was perhaps one of the most profitable learning periods towards this dissertation. For his discussions, help, sense of humor, and Dirty Harry quotes my thanks!

Along these years, I had the opportunity to collaborate with a number of researchers. I would like to thank Fernando G. Lobo, David E. Goldberg, Martin Pelikan, Kumara Sastry, Tian-Li Yu, Martin Butz, Xavier Llorà, Paul Winward, Carlos Fernandes, and Mark Hauschild. I also thank Carlos for inviting me to visit him in Granada and for showing me the great city it is.

I would like to thank all the current and former IlliGAL members with whom I had the pleasure to work and interact. Moments such as round tables at Mandarin-Wok, brunch at the Original House of Pancakes, foozball games at Illini Union, movie sessions at the lab library, wii sessions at Noriko's place, or 2-on-1 tennis games, will always remain in my memory. I also would like to thank all the friends I have made in Illinois.

I am very grateful to all my colleagues and friends from UAlg, Faro, Lisbon, and Tavira, for their company and friendship.

Last, but certainly not least, I would like to thank to my family, especially to my mother Maria Judite and my grandfather João.

The work in this thesis was sponsored by *Fundação para a Ciência e Tecnologia* through doctoral grant SFRH/BD/16980/2004 and research project PTDC/EIA/67776/2006.

Contents

List of Figures

List of Tables

List of Abbreviations

ACO	Ant colony optimization
BAA	Binary ant algorithm
BB	Building block
BBMA	Building-block-wise mutation algorithm
BD	Bayesian-Dirichlet
BIC	Bayesian information criterion
BMDA	Bivariate marginal distribution algorithm
BN	Bayesian network
BOA	Bayesian optimization algorithm
BP	Belief propagation
cGA	Compact genetic algorithm
CPT	Conditional probability table
DT	Decision tree
EA	Evolutionary algorithm
EBNA	Estimation of Bayesian networks algorithm
eCGA	Extended compact genetic algorithm
eCMA	Extended compact mutation algorithm
EDA	Estimation of distribution algorithm
EET	Efficiency enhancement technique

FDA	Factorized distribution algorithm
GA	Genetic algorithm
hBOA	Hierarchical Bayesian optimization algorithm
heCGA	Hybrid extended compact genetic algorithm
K2	BD metric with uninformative prior
LBP	Loopy belief propagation
LFDA	Learning factorized distribution algorithm
MBS	Model-based search
MDL	Minimum description length
MIMIC	Mutual-information-maximization input clustering
MPC	Most probable configuration
MPM	Marginal product model
PBIL	Population-based incremental learning
PLS	Probabilistic logic sampling
SLS	Substructural local search
UMDA	Univariate marginal distribution algorithm

Chapter 1

Introduction

1.1 Motivation

Estimation of distribution algorithms (EDAs) are search and optimization procedures that use probabilistic modeling of promising solutions to bias the search in the solution space. By incorporating advanced machine learning techniques into evolutionary algorithms (EAs), EDAs can solve many challenging optimization problems in an efficient and scalable manner, significantly outperforming standard EAs and other optimization techniques [133].

Although EDAs are effective at exploring the search space to find promising regions, they inherit a common drawback from traditional global search methods—slower convergence to optimal solutions when compared to appropriate local searchers that start the search within the basin of attraction of the optima. This observation often leads to the combination of global search with more local search methods. In this context, EDAs are no exception and many applications in real-world optimization have been accomplished with the help of some sort of local search. However, systematic methods for hybridizing and designing competent

global and local-search methods that automatically identify the problem decomposition and important problem substructures are still scarce.

One of the key requirements for designing efficient local search operators is to ensure that they search in the correct neighborhood [149, 10, 168]. This is often accomplished by exploiting and incorporating domain- or problem-specific knowledge in the design of neighborhood operators. When these neighborhood operators are designed for a particular search problem, oftentimes on an ad-hoc basis, they do not generalize their efficiency beyond a small number of instances. On the other hand, simple bitwise hillclimbers are frequently used as local search methods with more general applicability, providing inferior but still competitive results, especially when combined with population-based search procedures. Clearly, there is a tradeoff between generalization and efficiency for neighborhood operators with fixed structure. Therefore, it is important to study systematic methods for designing neighborhood operators that can solve a broad class of search problems.

This thesis demonstrates that the probabilistic models of EDAs contain useful information about the underlying problem structure that can be exploited to speedup the convergence of EDAs to optimal solutions. The exploration of neighborhoods defined by the probabilistic models in EDAs—named as *substructural neighborhoods*—is an approach that exploits the underlying problem structure while not loosing the generality of application. The resulting mutation operators explore a more *global*, problem-dependent neighborhood than traditional local search procedures purely representation-dependent. To investigate the utility of incorporating substructural local search (SLS) in EDAs, the thesis concentrates on the extended compact genetic algorithm (eCGA) and the Bayesian optimization algorithm (BOA).

1.2 Thesis Objectives

The thesis has three main objectives, which are:

- Extend the concept of substructural neighborhoods to Bayesian EDAs.
- Design more robust and efficient EDAs by combining model sampling with substructural local search.
- Investigate the importance of exploiting model-based information to improve the search in structure-learning EDAs.

1.3 Main Contributions

The main contributions of this thesis are:

- Integration of substructural local search in the extended compact genetic algorithm [104, 100]. The resulting algorithm combines the best of both approaches and implicitly allows to choose between a global and local search operator based on problem requirements.
- Development of substructural neighborhoods for Bayesian EDAs [103, 100]. These adaptive neighborhoods consider more complex variable interactions than previous works.
- Application of fitness modeling with Bayesian networks to the context of substructural local search [103, 102, 100]. This feature allows to substantially reduce the cost of local search in terms of function evaluations.
- A detailed study on model structural accuracy in the Bayesian optimization algorithm [99, 98, 101, 96, 70, 71]. This study identifies both the selection

operator and the scoring metric (that guides model search) as the main factors[1] to influence model quality in Bayesian EDAs. These results demystifies previous empirical comparisons between different Bayesian EDAs.

- Development of a substructural local searcher based on loopy belief propagation [102, 100]. The proposed loopy SLS is shown to speedup the search for optimal solutions when compared to a previous approach.

- Integration of substructural local search in the Bayesian optimization algorithm [103, 102, 100]. The resulting algorithm is able to significantly improve the performance of standard BOA for problems with both overlapping and non-overlapping interactions.

1.4 Organization

The thesis is composed by seven chapters. Chapter 1 introduces the motivation for this work, clarifies the main objectives of the thesis, enumerates the contributions to the current state-of-the-art, and details the organization of the thesis.

Chapter 2 reviews the topic of scalable optimization through probabilistic modeling. It starts by introducing genetic algorithms (GAs) as search methods inspired by natural selection and genetics, and then proceeds to the design of scalable GAs. Estimation of distribution algorithms are presented as a combination of GAs and machine learning methods for competent and scalable problem-solving. A connection between EDAs and other model-based search (MBS) methods is also made, recognizing that EDAs are among the most general frameworks in MBS. The chapter ends by referring to some of the efficiency-enhancement techniques that can be used in GAs and EDAs.

[1] Given a sufficient population size (learning data set).

Chapter 3 presents the integration of substructural local search in the extended compact genetic algorithm. Consequently, the probabilistic model of eCGA is used for two distinct purposes: (1) effective recombination of partial solutions that provides rapid global-search capabilities, and (2) effective search in the neighborhood of partial solutions that locally provides high-quality solutions. Experiments performed for different dimensions of problem difficulty show that, independently from the faced difficulty dimension(s), the eCGA which integrates SLS obtains a more robust performance, following the behavior of the best single-operator-based approach for each problem.

Chapter 4 extends the concept of SLS for the Bayesian optimization algorithm. The substructural neighborhoods explored in local search are consequently defined by the conditional dependencies learned in the Bayesian networks. Additionally, a surrogate fitness model is used to evaluate the alternatives in the subsolution search space. The results show that incorporating SLS in BOA can lead to a significant reduction in the number of generations necessary to solve the problem, while providing significant savings in function evaluations. However, for larger problem instances the efficiency decreases. The reason behind such behavior is associated with the structural accuracy of the probabilistic models in BOA.

Chapter 5 investigates the relationship between the Bayesian networks learned in BOA and the underlying problem structure. The purpose of the chapter is threefold. First, model building in BOA is analyzed to understand how the problem structure is learned. Second, it is shown how the selection operator can lead to model overfitting in Bayesian EDAs. Third, the scoring metric that guides the search for an adequate model structure is modified to take into account the non-uniform distribution of the mating pool generated by tournament selection. Overall, this chapter makes a contribution towards understanding and improving model accuracy in BOA, providing

more interpretable models to assist efficiency-enhancement techniques and human researchers.

Chapter 6 proposes new developments to improve the efficiency of SLS in BOA. The $s-$penalty (proposed in Chapter 5) is shown to improve significantly the performance of local search for larger problem instances. The presence of overlapping interactions as a new source of problem difficulty is investigated while performing SLS. To efficiently tackle overlapping problems, a new SLS method is proposed based on the principles of loopy belief propagation. The resulting loopy SLS can efficiently solve problems with both overlapping and non-overlaping interactions. The utility of such local searcher is also discussed for problems with hierarchical interactions, where it fails to improve the results obtained by standard BOA. The chapter ends by recognizing that integrating loopy SLS in BOA enables the resulting algorithm to have the best of both worlds—efficient mutation and recombination of partial solutions—for problems with non-overlapping, overlapping, and hierarchical interactions.

Chapter 7 presents some topics for further research and concludes the thesis.

Chapter 2

Scalable Optimization via Probabilistic Modeling

2.1 Introduction

When facing an optimization problem, one must decide how to solve it. Exhaustive search methods can be used to find the exact best solution(s) to the problem at hand. However, that comes to a price of evaluating a great part of the entire solution space. While this may be a reasonable approach for some small-sized problems, it becomes a serious bottleneck when facing larger problem instances. For example, the problem of finding the optimal order of visit for a given number of cities such that the total distance traveled is minimized, is known as the traveling salesman problem (TSP). For a TSP instance with 10 cities there is a total of $\frac{9!}{2} = 181,440$ possible solutions, which is still a manageable search space. But simply doubling the problem size to 20 cities makes the search space grow to more than 10^{16} solutions! Therefore, for many problems a rather good solution obtained in a few seconds is preferred over the exact best solution that can take years to achieve.

Classical methods from operations research (OR) were among the first solvers to address this kind of problems. However, most of these methods require problem-specific knowledge to solve large-scale problems in reasonable time. But domain-specific knowledge to reduce the combinatorial complexity is often hard to obtain. Moreover, decision variables can interact in such ways that are not clear from the optimization problem formulation. In black-box optimization, there is no information about the relation between the performance measure and the semantics of the solutions [125]. Therefore, the problem is treated as a black box, where for each possible solution as an input parameter the box returns its quality measure as an output. In such cases, the only way of learning something about the problem at hand is to generate candidate solutions and evaluate them.

This chapter introduces genetic algorithms which are particularly suited for black-box optimization, although there are several ways to incorporate prior information about the problem into these algorithms. The design of scalable genetic algorithms is also discussed along with the definition of the test problems used in the thesis. Estimation of distribution algorithms, which combine machine learning methods with genetic algorithms, are also introduced with some detail and their connection with other model-based search methods is referred. The chapter ends with a description of some efficiency enhancement techniques that can be used to speedup the search process of genetic algorithms and estimation of distribution algorithms.

2.2 Genetic Algorithms

Genetic algorithms (GAs) [77, 47, 51] approach black-box optimization by evolving a population of candidate solutions with operators inspired by natural selection and genetics [125]. In GAs, the decision variables of the optimization problem

***Simple Genetic Algorithm* (sGA)**

(1) Initialize a population `P` of n individuals.

(2) Evaluate population `P`.

(3) Select `P'` individuals from `P` .

(4) Recombine individuals in `P'` to generate the offspring population `O`.

(5) Mutate individuals in `O`.

(6) Evaluate population `O`.

(7) Replace all (or some) individuals in population `P` by those from `O`.

(8) If stopping criteria are not satisfied, return to step 3.

Figure 2.1: Pseudocode of the simple genetic algorithm (sGA).

are encoded in strings of a certain length and cardinality. Each string represents a candidate solution and is referred to as *chromosome*, while each string position is known as *gene* and its value *allele*. Each candidate solution can be represented either by its *phenotype*, which is the solution representation in the objective function domain, or by its *genotype*, that is the solution representation under which the genetic operators operate. Depending on the problem and encoding scheme, the chromosome can be binary, integer, real-valued, etc. The size of the chromosome can be either fixed or variable during the search process. This thesis focuses on binary encoding and fixed chromosome length.

The major input of a GA is the fitness function which determines the quality of candidate solutions. This can be an *objective* function, given by a mathematical model or a computer simulation, or take the form of a *subjective* function where humans choose better solutions over worse ones. Figure 2.1 presents the pseudocode of a simple GA. Each essential component of the GA is detailed below:

Initialization Typically, the initial population of individuals is generated at ran-

dom across the entire search space. Alternatively, it can be initialized based on some prior knowledge about the problem, by inserting previously known good solutions or applying local search methods before the GA run.

Selection The selection operator is responsible for applying the survival-of-the-fittest principle in GAs. The main idea is to select promising individuals over worse individuals according to fitness. Several selection schemes have been proposed in the literature, but this work focuses on tournament selection and truncation selection.

In tournament selection [56, 18], s individuals are randomly picked from the population and the best one is selected for the mating pool. This process is repeated n times, where n is the population size. There are two popular variations of tournament selection, with and without replacement. With replacement, the individuals are drawn from the population following a discrete uniform distribution. Without replacement, individuals are also drawn randomly from the population but it is guaranteed that every individual participates in exactly s tournaments. While the expected outcome for both alternatives is the same, the latter is a less noisy process.

In truncation selection [121] the best $\tau\%$ individuals in the population are selected for the mating pool. This method is equivalent to the standard (μ, λ)-selection procedure used in evolution strategies [138], where $\tau = \frac{\mu}{\lambda} \times 100$.

Note that when increasing the size of the tournament s, or decreasing the threshold τ, the selection intensity increases. The population of selected individuals is often referred to as *mating pool*.

Recombination Crossover or recombination operators combine parts of two or more parent solutions to create new, possibly better solutions, which are re-

ferred as *offspring*. The recombination operator is performed according to a user-defined probability p_c, which is usually high. There are many ways to achieve recombination of different solutions, from which one-point crossover, α-point crossover, and uniform crossover are classical examples. One-point crossover between two parents consists in randomly picking a position along the chromosome and exchanging one of the two portions defined by the crossover point. For the α-point crossover, there are $\alpha + 1$ portions from which half is exchanged between the two parents (exchange the first, leave the second, exchange the third, leave the fourth, and so on). For uniform crossover, every allele in the chromosome has probability p (typically 0.5) of being exchanged. These crossover operators have different biases and mixing abilities [147].

In GAs, properly designed recombination mechanisms are often identified as the main ingredient for achieving competent performance. Consequently, a significant effort has been made in designing competent recombination operators, that automatically identify the important components of parental solutions and effectively exchange them [47, 51, 130, 95, 133, 108, 23].

Mutation Like in nature, mutation is responsible for introducing small variations to chromosomes, useful for maintaining diversity in the population. This is achieved by performing random modifications locally around a solution. For a binary chromosome, the bit-wise mutation operator simply flips the values of genes according to a mutation probability p_m, which is typically low. Note that without mutation, the offspring chromosomes are limited to the alleles present in their parents.

Replacement The replacement method specifies in which way the offspring population is combined with the original parental population. Replacement tech-

niques such as elitism, generation-wise, steady-state, or niching can be used. This thesis mainly considers the generation-wise replacement where the offspring fully replaces the original population.

Stopping Criteria The stopping criteria determine one or more termination conditions to end the search process. The conditions can be based on the number of generations, number of fitness evaluations, solution quality, or simply clock time.

2.3 Design of Scalable Genetic Algorithms

While nature, with all its beauty and diversity, is very useful to understand the working mechanisms of GAs, it poses difficulty as a design metaphor because its underlying processes are themselves extremely complex and not well understood. Facing this dilemma, some researchers have taken a design decomposition approach to understand and design *competent* genetic algorithms, which can solve hard problems, quickly, reliably, and accurately [51].

Based on the notion of *building block* (BB) [77]—highly fit partial solutions[1]—the problem of designing competent selectorecombinative GAs can be decomposed in the following seven subproblems [51]:

1. Know what GAs process—building blocks.
2. Know thy BB challengers—building-block-wise difficult problems.
3. Ensure an adequate supply of raw BBs.
4. Ensure increased market share for superior BBs.

[1]Along the thesis, the definition of BB is often relaxed to a partial solution, module, or substructure, which is not necessarily associated with high fitness.

5. Know BB takeover and convergence times.

6. Make decisions well among competing BBs.

7. Mix BBs well.

Research done along these lines include problem difficulty [1, 46, 31, 169, 126, 172], adequate supply [76, 48, 139, 59], market share growth [58], takeover time [52, 4, 20], convergence time [121, 164, 5, 112, 166, 105], decision making [29, 57, 53, 62, 166, 105], and understanding mixing [55, 165, 163, 147, 148]. A more detailed review of this methodology can be found elsewhere [51].

Efficient mixing has been a central topic of research and eventually led to the development of competent GAs [51, 130, 95, 133, 108, 23]. In contrast to traditional GAs, competent GAs use recombination operators that are able to capture and adapt themselves to the underlying problem structure. The existence of competent GAs significantly reduces the importance of choosing an adequate codification or genetic operator that characterizes many GA applications. Depending on the technique used to discover the problem decomposition, competent GAs can be classified into the following categories:

- Perturbation techniques [56, 54, 86, 122, 175, 73].

- Linkage adaptation techniques [66, 22].

- Probabilistic model building techniques [130, 95, 125, 16, 133, 108].

This thesis focuses on the third approach to automatically recognize and mix important problem subsolutions. The next section details the test problems used along the thesis.

2.3.1 A Design Approach to Problem Difficulty

To verify the performance of competent GAs, the approach taken is to *design* bounding *adversarial problems* that represent different dimensions of problem difficulty [51]. In this way, different algorithms can be tested against a limited number of boundedly difficult problems, where the algorithm success for these particular problems ensures success against a large class of problems no harder than the test cases[2]. This adversarial design method contrasts sharply with the common practices of using historical, randomly generated, or ad hoc functions [51]. While adversarial problems are not real-world problems as well, they provide an important *proof-of-concept* which has been successfully used in designing state-of-the-art solvers.

The testbed used in the thesis combines the core of three important problem difficulty dimensions [51]:

1. **Intra-BB difficulty** where *deception* is considered to be the core of the difficulty generated from within a building-block;
2. **Inter-BB difficulty** where *scaling* is considered to be the core of the difficulty generated between or among building-blocks;
3. **Extra-BB difficulty** where *noise* is considered to be the core of the difficulty generated outside the problem or any building-block.

As the core of intra-BB difficulty, deceptive functions are among the most challenging problems for competent GA candidates. This kind of functions normally have one or more deceptive optima that are far away (in the genotype space) from the global optimum and which misleads the search, in the sense that the attraction area of the deceptive optima is much greater than the one of the optimal solution.

[2]What is often referred as performance envelope.

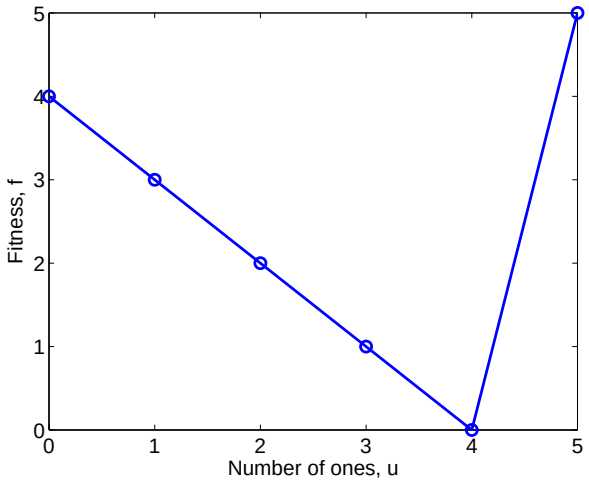


Figure 2.2: The trap function of order $k = 5$ (trap-5). The fitness value is a function of the number of ones in the string. The local optimum is located at 00000 and the global optimum at 11111. Note that the attraction area of the local optimum is much greater than for the optimal solution.

A well known deceptive function is the trap function [1, 31] defined as follows:

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

where u is the number of ones in the string and k is the size of the trap function. Figure 2.2 illustrates a trap function of order $k = 5$ (trap-5). For $k \geq 3$, the trap function is said to be fully deceptive [31] which means that any lower than k-order statistics will mislead the search away from the optimum.

Note that if the entire problem is composed by a single large trap function, the problem becomes as hard as the needle-in-a-haystack (NIAH) problem, for which has been shown that no algorithm can do any better (in finding the optimal solution) than random search or deterministic enumeration. Therefore, a bound on difficulty has to be set such that $k < \ell$. One such problem is the m-k trap problem, which is

composed by m copies of the trap function with order k:

$$f_{mk}(X) = \sum_{i=1}^{m} f_{trap}(x_{I_i}), \tag{2.2}$$

where each index set I_i is a disjoint tuple of variable indexes belonging to the i^{th} subfunction.

The inter-BB difficulty can be explored together with the intra-BB difficulty. Specifically, the same bounded deceptive function is used, but now each subfunction fitness contribution to the overall fitness is exponentially scaled such that

$$f_{scaling}(X) = \sum_{i=1}^{m} 2^{i-1} f_{trap}(x_{I_i}). \tag{2.3}$$

While scaling difficulty does not necessarily need to be exponential, this function act as bounding case for power-law scaling.

The core of extra-BB difficulty can be divided in two components: non-determinism or noise, and non-stationarity [51]. This thesis concentrates on noise as the main source of external difficulty. Specifically, it is assumed that the exogenous noise follows a Gaussian distribution with mean 0 and variance σ_N^2. To make the problem even more challenging, a noisy version of the m-k trap problem is considered. The function is defined as follows

$$f_{noise}(X) = f_{mk}(X) + \mathbf{G}(0, \sigma_N^2). \tag{2.4}$$

While the three problem difficulty dimensions have been defined based on a single type of modular interaction, other important types of interactions are considered along the thesis as well:

- **No interaction** when variables can be optimized independently from each

other.

- **Overlapping interaction** when different subproblems share common variables.
- **Hierarchical interaction** when important dependencies are expressed at more than a single level.

These additional types of interactions will be explored through the following test problems.

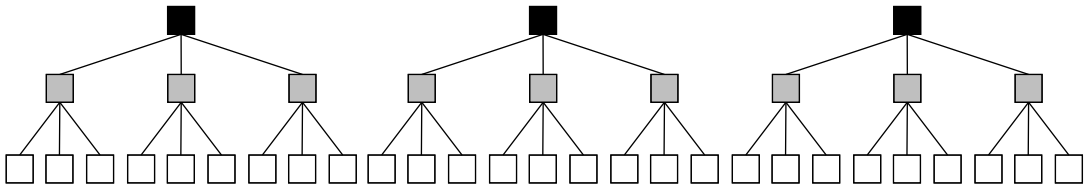
In onemax the fitness is simply given by the sum of ones in a binary string:

$$f_{onemax}(X) = \sum_{i=1}^{\ell} x_i. \tag{2.5}$$

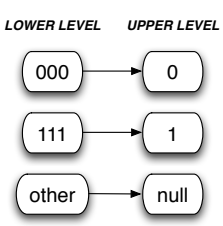
This is a simple linear function with the optimum in the solution with all ones. While there is no need of linkage learning to be able to solve this problem efficiently, it allows to set a lower bound on problem difficulty.

The overlapping difficulty is addressed by considering an overlapping version of m-k trap problem, where each index set I_i (corresponding to subfunction i) is no longer a disjoint tuple of variable indexes. Instead, each trap function shares o variables with two other neighboring subproblems. More precisely, a trap-k subfunction $f_j(x_i, x_{i+1}, \ldots, x_{i+k-1})$ will overlap with $f_{j-1}(x_{i-k+o}, x_{i-k+o+1}, \ldots, x_{i+o-1})$ and $f_{j+1}(x_{i+k-o}, x_{i+k-o+1}, \ldots, x_{i+2k-o-1})$.

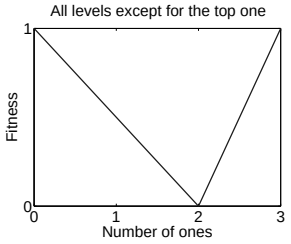
Finally, the hierarchical trap problem [126] poses a more difficult challenge to search procedures, as important dependencies are expressed at more than a single level. For this problem, the interactions at an upper level are too weak to be detected unless all lower levels are already solved. The hierarchical trap is defined by three components:



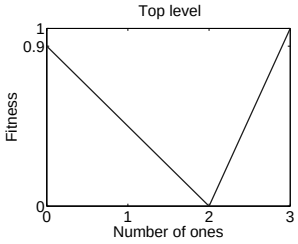
(a) Structure



(b) Mapping function



(c) Contribution function 1



(d) Contribution function 2

Figure 2.3: The 27-bit hierarchical trap problem with three levels and $k = 3$. This problem is defined by three components: (a) structure, (b) mapping function, and (c)(d) contribution functions. Note that 000 and 111 are equally good except for the top level. However, the local optimum 000 is easier to climb, which requires the maintenance of all alternatives until a decision can be reached at the top level.

1. *Structure* which is given by a balanced $k-$ary tree.

2. *Mapping function* that maps variables from a lower level to the next level. A block of all 0's and 1's is mapped to 0 and 1 respectively, and everything else is mapped to *null*.

3. *Contribution functions* are based on trap functions of order k. If any position of the string is *null*, the corresponding fitness contribution is zero.

Figure 2.3 shows a 27-bit hierarchical trap with three levels and $k = 3$. The contributions on each level are multiplied by 3^{level} so that the total contribution of each level is the same. The optimal solution is given by the string with all 1's. Notice that subsolutions 000 and 111 are equally good except for the top level. However, the local optimum 000 is easier to climb, which requires the maintenance of all alternatives until a decision can be reached at the top level.

2.4 Estimation of Distribution Algorithms

Estimation of distribution algorithms (EDAs) [130, 95, 16, 133, 108], also known as probabilistic model-building genetic algorithms (PMBGAs) or iterated density estimation algorithms (IDEAs), use probabilistic modeling of promising solutions to guide the search instead of the traditional recombination and mutation operators of GAs. Genetic algorithms use the genetic operators to bias the exploration of new regions in the search space, where new solutions inherit components from their parents. However, fixed, problem-independent variation operators have been proven to be inefficient for hard problems with bounded difficulties [115, 163], because BBs are often disrupted and not mixed efficiently.

On the other hand, EDAs are able to capture the underlying problem decomposition *on-the-fly*, thus scaling much better than simple GAs in terms of fitness evaluations required to solve the problem. The main feature in EDAs is to prevent disruption of important partial solutions contained in the population by giving them high probability of being present in the offspring population. The working principles of EDAs are similar to those of simple GAs except that the genetic operators are replaced by the following two steps:

1. Estimate the joint probability distribution of promising solutions (selected individuals).

2. Generate new solutions by sampling from the estimated distribution.

This procedure tries to mimic the behavior of an ideal recombination operator that combines subsolutions with minimal disruption and maximum efficiency.

The estimation of the true distribution however is not a trivial task. There is a tradeoff between the accuracy and the efficiency of the estimate. For each choice of the probabilistic model used in an EDA there is an associated model

expressiveness and learning complexity. Therefore, these algorithms are typically classified according to the expressiveness/complexity of the probabilistic model they rely on. This chapter classifies EDAs into univariate, bivariate, and multivariate.

Several EDAs have been proposed for discrete [130, 95], real-valued [130, 95, 87], computer programs [158], and other representations. Since this thesis focuses on discrete EDAs, the overview presented below concentrates on this domain as well.

2.4.1 Univariate Models

The first proposed EDAs rely on univariate models where the variables of the problem are assumed to be independent. These models consist on the product of local probabilities for each variable. Therefore, univariate EDAs use a fixed model structure and only learn the corresponding parameters. The model contains a set of frequencies of all possible values for each string position. For a representation with cardinality χ and size ℓ, the model stores $(\chi - 1)\ell$ frequencies[3]. These frequencies are first estimated by inspecting the population of promising solutions, and then used to construct new solutions through sampling.

The population-based incremental learning (PBIL) algorithm [8], also known as equilibrium genetic algorithm [85], replaces the entire population by a probability vector. This probability vector is initially set to $p_i = 0.5$, where p_i represents the probability of 1 for the ith variable position. Consequently, there is an equal probability of sampling 0 and 1, which simulates a randomly generated population. For each iteration of PBIL, a number of solutions is generated according to the current probability vector and the best one is used to update the probabilities in

[3]Given that all relative frequencies should sum up to 1 for each chromosome position, only $\chi - 1$ need to be stored.

the following way

$$p_i = (1 - \lambda)p_i + \lambda x_i, \tag{2.6}$$

where λ is the learning rate (between 0 and 1) and x_i is the ith bit of the best solution. According to the above update rule, the probability vector is shifted towards the values of the best solutions.

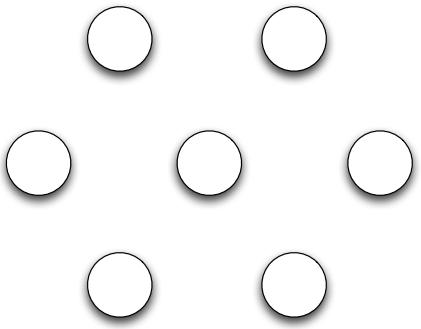
The compact genetic algorithm (cGA) [67] is similar to PBIL but uses a different update rule inspired on the behavior of a steady-state GA with binary tournament selection. In each iteration, two solutions are sampled from the probability vector and undergo tournament selection, which determines the winner W and the looser L. The probability vector is then updated as follows

$$p_i = \begin{cases} p_i + \frac{1}{n} & \text{if } W_i = 1 \text{ and } L_i = 0 \\ p_i - \frac{1}{n} & \text{if } W_i = 0 \text{ and } L_i = 1 \\ p_i & \text{otherwise,} \end{cases} \tag{2.7}$$

where n is the simulated population size, W_i and L_i represent the ith bit of the winner and looser solution. Note that while cGA does not maintain a population, the update step of the probability vector corresponds to replacing the looser solution by another copy of the winner solution in a population of size n.

The univariate marginal distribution algorithm (UMDA) [120] differs from PBIL and cGA by explicitly maintaining a population. The probability vector computed using the selected population is then used to sample the entire offspring population. Essentially, UMDA is equivalent to a GA with n-parent uniform crossover, where each offspring can inherit components from any solution in the parental population (instead of the typical pairwise crossover).

All these algorithms present a similar behavior. They perform very well for linear



(a) PBIL, cGA, UMDA

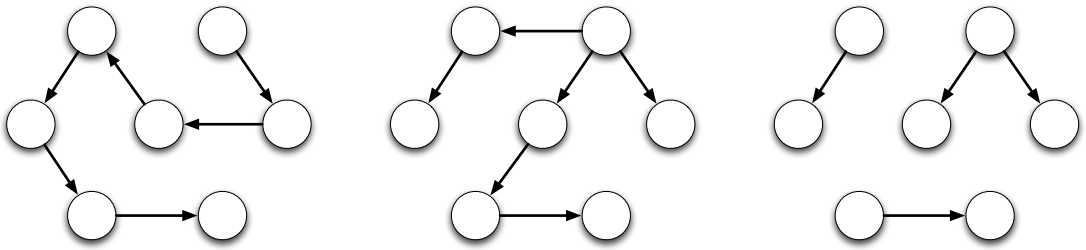
Figure 2.4: Graphical representation of the probabilistic models used by PBIL, cGA, and UMDA. Nodes represent variables and edges represent dependencies. The model can not express interactions between variables.

problems, achieving between linear and sub-quadractic performance, but they fail on problems with strong interactions between variables [130]. On the other hand, the model learning complexity is very low as it only requires parameter learning. Figure 2.4 shows the graphical representation for the probabilistic models used by PBIL, cGA, and UMDA.

2.4.2 Bivariate Models

Recognizing that univariate models are not sufficient to solve complex problems, structure-learning models were later introduced. EDAs based on bivariate models that represent pairwise interactions between variables were proposed under a number of frameworks, typically using chain or tree structures.

The mutual-information-maximization input clustering (MIMIC) [28] is one of such algorithms. The MIMIC algorithm uses a chain distribution which determines that every variable depends on another variable, with the exception of the initial variable in the chain that is independent (see Figure 2.5 (a)). In each generation, the permutation of variables in the chain is searched to maximize the mutual in-



(a) MIMIC (b) Tree-based EDA (c) BMDA

Figure 2.5: Graphical representation of the probabilistic models used by MIMIC, tree-based EDA, and BMDA. Nodes represent variables and edges represent dependencies. The models can cover pairwise interactions between variables using a (a) chain, (b) tree, or (c) forest distribution.

formation of neighboring variables. Because there are $\ell!$ possible chain structures, MIMIC uses a greedy algorithm to find an adequate permutation. In this way, the Kullback-Liebler divergence [94] between the chain and the complete joint distribution is (heuristically) minimized. Once the chain distribution is learned, each solution is sampled according to the chain order and the corresponding conditional probabilities.

Baluja and Davies [9] extended PBIL by replacing the population with a probability vector that contains all pairwise probabilities (initially set to 0.25). The model structure takes the form of a dependency tree, where the variable in the root is independent and every other variable is conditioned on its parent in the tree (see Figure 2.5 (b)). The model search starts by picking the variable to place at the root of the tree, and then keeps adding variables to the existing tree so that the mutual information between the new variable and its parent is maximized. New solutions are then sampled by following the tree structure and using the conditional probabilities computed from the probability vector.

The bivariate marginal distribution algorithm (BMDA) [131] extends the UMDA by using a distribution over a set of dependency trees, also known as forest distribu-

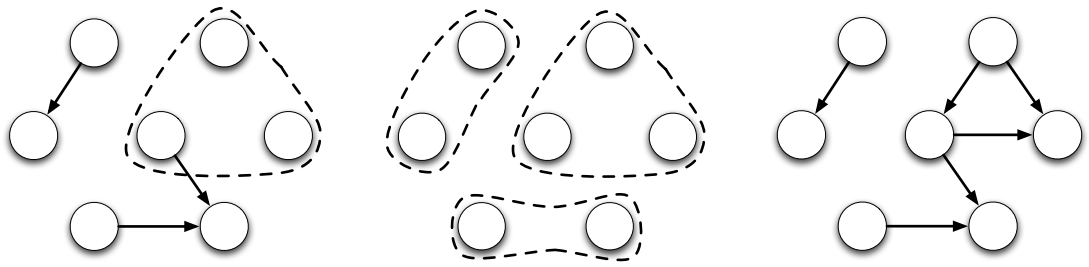
tion. This type of model is even more general than the previous one because it allows to represent several mutually independent dependency trees (see Figure 2.5 (c)). To guide the search for an adequate model structure, the Pearson's chi-square test [109] is used to determine which variables to connect and how many independent trees should have the final model.

Bivariate EDAs can reproduce and mix BBs of order two very efficiently, performing quite well on linear and quadratic problems. However, pairwise interactions are still insufficient to solve problems with multivariate and highly overlapping BBs. The complexity of model learning varies between quadratic and cubic with respect to the number of variables.

2.4.3 Multivariate Models

Multivariate EDAs are based on more powerful probabilistic models such as marginal product models or Bayesian networks. These models are capable of expressing interactions between multiple variables. However, such models don't come for free as they require more computational effort than simpler models to estimate the distribution of promising solutions. On the other hand, these models allow EDAs to solve many difficult problems that are simply intractable to simpler EDAs and GAs.

The factorized distribution algorithm (FDA) [118] uses a factorized distribution that contains multivariate marginal and conditional probabilities (see Figure 2.6 (a)). However, FDA only learns the corresponding parameters because the structure of the model has to be given to the algorithm. Therefore, when solving a given problem with FDA one must care about the problem decomposition and explicitly inform the algorithm about it. While in some cases it might be useful to incorporate prior knowledge about the problem, the FDA approach is not adequate for black-box optimization where the solver should be able to learn the regularities in the search



(a) FDA (b) eCGA (c) BOA, EBNA, LFDA

Figure 2.6: Graphical representation of the probabilistic models used by FDA, eCGA, and BOA (EBNA and LFDA as well). Nodes represent variables and edges represent dependencies. All these models can cover interactions between multiple variables. However, in FDA the model structure has to be specified, contrary to the remaining algorithms that learn an adequate structure on-the-fly. Also, eCGA can not capture overlapping interactions while others can.

space.

The extended compact genetic algorithm (eCGA) [65, 63] uses marginal product models that partition the problem variables into several non-overlapping clusters (see Figure 2.6 (b)). These clusters are then manipulated as whole in the same way that UMDA deals with each independent variable. Therefore, each cluster represents the structure of a building block that should be propagated and combined with other BBs. To determine which variables should be grouped together in each cluster, eCGA uses a greedy learning algorithm guided by a metric based on the minimum description length (MDL) principle [140]. Essentially, the MDL metric gives preference to models that better compress the data (set of promising solutions) and at the same time are not overly complex[4].

The Bayesian optimization algorithm (BOA) [127, 129] makes use of Bayesian networks to model the set of promising solutions. Bayesian networks represent a more general class of distributions than marginal product models. The model structure is represented by an acyclic directed graph and the parameters by con-

[4]Following Ockham's razor principle: "All other things being equal, the simplest solution is the best."

ditional probabilities according to the structure of the graph (see Figure 2.6 (c)). The search for an adequate Bayesian network structure can be guided by different metrics such as the Bayesian-Dirichlet (BD) [26, 74] and the Bayesian information criteria (BIC) [156]. Several algorithms have been proposed to learn the adequate structure (including GAs themselves), but for the EDA context a greedy algorithm is more appropriate to obtain a good compromise between search efficiency and model accuracy.

The estimation of Bayesian networks algorithm (EBNA) [37] and the learning factorized distribution algorithm (LFDA) [119] are Bayesian EDAs as well. These algorithms are very similar to BOA as they mainly differ in the score metrics used to guide model search and the selection method (truncation or tournament selection). Later on, Pelikan and Goldberg proposed the hierarchical BOA (hBOA) [126, 125], which uses Bayesian networks with decision trees and a niching mechanism to replace old solutions.

With the exception of FDA, all the algorithms described above can learn multivariate distributions to capture possible problem decompositions. This feature is highly desirable in black-box optimization. Of course, there is an additional complexity for learning these models which is typically $O(\ell^3)$. However, the computational time saved by the reduced number of evaluations necessary to solve the problem can be much larger than the time spent in learning the models, thereby reducing the overall time complexity. The rule of thumb, when facing a new problem with EDAs, is to start with simpler models and progress to more complex models as long as the solution quality improvements pays off (or the time spent to obtain the same solution quality decreases).

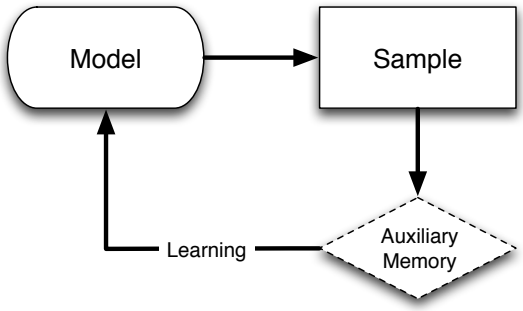


Figure 2.7: Scheme of model-based search (adapted from Zlochin *et al.* [178]). Auxiliary memory may be, or not, part of the loop. For instance, UMDA makes use of memory (population of solutions) while PBIL and cGA do not.

2.4.4 Model-Based Search: The Big Picture

Heuristic search algorithms can be classified as being either *instance-based* or *model-based* [137, 178]. Instance-based heuristics generate new candidate solutions using only the current solution or set of solutions, while model-based heuristics rely on a probabilistic model that is updated according to previously seen solutions. Most traditional search methods can be considered instance-based, including genetic algorithms [77, 47], simulated annealing [89, 167], tabu search [43, 44, 45], iterated local search [106, 107], and others. On the other hand, model-based search (MBS) is a more recent approach, which have gain increasing popularity in the last decade.

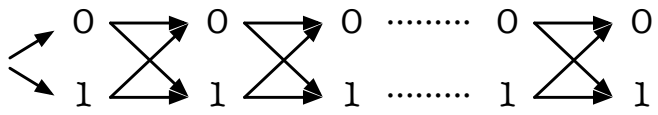
Model-based search algorithms generate candidate solutions using a parameterized probabilistic model that is updated using the previously seen solutions in such way that the search will concentrate in the regions containing high quality solutions [178]. Figure 2.7 presents the main scheme of model-based search. Clearly, EDAs belong to such class of algorithms. Other algorithms that can be classified as model-based include ant colony optimization (ACO) [32, 34, 33], cross-entropy method [142], and stochastic gradient ascent [141, 12].

Zlochin *et al.* [178] proposed a unifying MBS framework that accommodates all

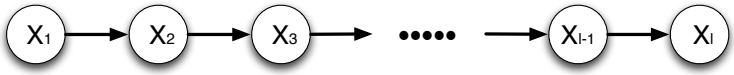
these algorithms and analyzed their similarities as well as their distinctive features. The authors identified two important components in the EDA approach that are not present in other MBS techniques: (1) a population of solutions which evolves during the search process and is used for building the probabilistic model, and (2) the use of a flexible model structure, which is determined using an appropriate learning algorithm. On the basis of this observation, it is fair to say that the EDA approach is the most general among MBS methods. A detailed overview about MBS is beyond the scope of this thesis and for that the reader is referred elsewhere [178]. Instead, an illustrative example is provided to demonstrate the similarities between EDAs and ACOs, and the generality provided by EDAs.

Ant colony optimization is inspired by the capacity of ants in finding near-shortest paths between their nests and food. Along their path, ants place a pheromone trail on the ground to inform other ants about the quality and amount of food that can be found in that trail. Heuristic search with ACO models such behavior for identifying promising solutions in the search space. The vast majority of ACO algorithms use models based on univariate statistics, fact which has already been identified as the main reason for poor performance in some problems [15].

One exception is the binary ant algorithm (BAA) [40] which builds pheromone maps over a graph of possible trails represented in Figure 2.8 (a). The generation of candidate solutions is done by starting from the left, instantiating the first variable with 0 or 1. After that, each 0 or 1 has two connections, each leading again to 0 or 1. These trails are unidirectional, therefore the solution is built starting from the first variable towards the last one. For each possible alternative in the trail there is an associated probability (or pheromone level to be more accurate), which essentially depends on the value of the current variable (if the ant is located at 0 or 1) and the next variable. Clearly, this is a probabilistic model that takes into account



(a) BAA (adapted from Fernandes *et al.* [40])



(b) MIMIC

Figure 2.8: The pheromone graph used in the binary ant algorithm (BAA) and its equivalent chain distribution used in MIMIC. In BAA the solution is built from left to right based on probabilities associated with each arrow. At a given point in the graph, the choice of the value for variable X_i depends on the value chosen for X_{i-1}, therefore pairwise interactions are captured by this model.

pairwise interactions. Moreover, the graph used in BAA is a particular case of the model used in MIMIC. The equivalent model in MIMIC is shown in Figure 2.8 (b), where the chain distribution follows the sequence $X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_\ell$. The fundamental difference is that MIMIC searches for a chain order that maximizes mutual information between neighboring variables, while BAA uses a fixed model structure.

Indeed, while learning the structure of the model based on search experience, EDAs demonstrate to be a more general framework than ACOs. Nevertheless, the important point is to recognize their similarities as well as their differences, so that lessons learned from each field can be generalized and transferred. One such example is the integration of a new update strategy for the probabilistic model in UMDA based on ACO's transition probability equations, with very promising results for dynamic optimization [39].

2.5 Efficiency Enhancement Techniques

Competent GAs successfully solve hard problems with bounded difficulties, requiring only a low polynomial (often subquadratic) number of function evaluations with respect to problem size. Essentially, competent GAs take problems that were *intractable* to first-generation GAs and renders them *tractable*. While from a theoretical perspective, low-polynomial scalability is a very good performance, from a practical point of view, even a subquadratic number of fitness evaluations can be very demanding when considering large-scale problems. This is especially true if the fitness evaluation requires a complex simulation or computation. Thus, a number of *efficiency enhancement techniques* (EETs) have been developed with the aim of taking problems from tractable to *practical*.

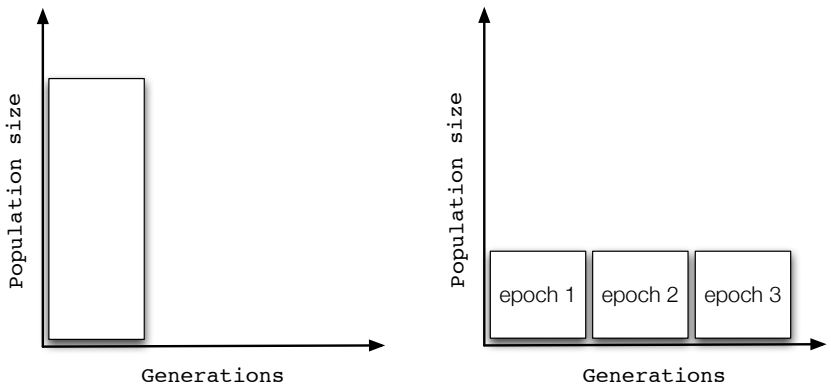
A clear advantage of EDAs is that the probabilistic models contain useful information about the underlying problem structure that can be exploited in the principled design of various efficiency-enhancement methods. Systematically incorporating problem knowledge *mined* through the model-learning process of EDAs into the design of an EET makes it adaptive and can potentially improve the speedup of the method [155]. EETs can be broadly classified into four categories [51, 155]:

Parallelization. When GAs are run over multiple processors to distribute the computational load. GAs themselves are implicitly parallel search procedures and many configurations for explicit parallelization can be found in the literature [19, 21]. Independently from the configuration used, the main idea is to distribute the computational effort on several independent processors to speedup the entire GA run. A relevant design theory for optimizing the key facets of parallel architecture, connectivity, and deme size is already available [21].

Hybridization. When GAs are combined with local search methods that use some sort of domain- or problem-specific knowledge [114, 68, 91, 159]. Many applications of GAs in industry follow this approach in order to gain from the benefits of hybridization. Typical motivations for incorporating local search methods in GAs include faster convergence, refinement of solutions obtained by the GA, initialization of the population, and repairing infeasible solutions. In this way, the GA can focus on finding good search regions while the local search ensures that local optima in these regions will be found.

Evaluation relaxation. When accurate, but expensive fitness functions are replaced by less accurate, but inexpensive fitness estimators, thereby reducing the total number of costly fitness evaluations [83, 160, 145]. The first proposal for relaxing evaluation is attributed to Fitzpatrick *et al.* [41], that obtained substantial speedups through reduced random sampling of pixels in a medical image. Although the first studies in evaluation-relaxation were mostly empirical, recent years have brought more theoretical work to understand and optimize the speedups obtained for fitness functions with known variance or known bias, integrated fitness functions, and fitness inheritance. Fitness inheritance [160, 152] is of particular importance because, when coupled with the probabilistic models of EDAs [132, 154, 153, 155], it has produced speedups substantially superior to those obtained by conventional schemes.

Time continuation/utilization. Considers the tradeoff between the search using a large population and a single convergence epoch or using a small population with multiple convergence epochs (see Figure 2.9). This tradeoff also includes the optimal use of both mutation and recombination capabilities to obtain the best possible solution within limited computational re-



(a) Scenario 1 (b) Scenario 2

Figure 2.9: Two scenarios of time utilization: (a) large population with a single convergence epoch, and (b) small population with multiple convergence epochs.

sources [50, 162, 150, 149, 151]. Previous theoretical work indicates that when both recombination and mutation operators have linkage information, a small population with multiple convergence epochs is more efficient for problems with BBs of equal (or nearly equal) salience. However, if the problem is noisy or has overlapping BBs, then a large population with a single convergence epoch is more effective [150]. On the other hand, if the BBs of the problem have non-uniform salience (requiring serial processing), then a small population with various epochs is more appropriate [50, 151].

The speedup obtained by any EET can be measured as the ratio between the computation effort required by a standard GA and a GA with efficiency-enhancement. The computational effort can be quantified simply as clock time or number of function evaluations (which for many problems dominates the total computational time). If one assumes that the performance of one EET does not affect the performance of others, then if more than one of the above methods is employed, the overall speedup is given by the product of individual speedups—what is known as *multiplicative* speedup [51].

While the possibility of obtaining multiplicative speedups with EETs is promising by itself, recent studies indicate that when integrating problem knowledge mined by EDAs in the design of the EET, substantially higher speedups can be obtained [132, 154, 104, 103, 153, 155]. These are known as *super-multiplicative* speedups.

The work presented in this thesis deals with three of the EETs mentioned above in the following way:

- Most hybridization methods are ad hoc and automatic methods for identifying and exploiting problem decomposition in both global search and local search methods are lacking. The thesis addresses this issue by integrating adaptive-neighborhood local search in multivariate EDAs. Note that conceptually this is different from typical hybrid EDAs in the sense that local search is based on the learned probabilistic model instead of using prior knowledge about the problem, which turns it into a more general applicable type of hybridization (with all the corresponding advantages and disadvantages).

- The local search methods proposed in this work use a form of BB-wise mutation which can be viewed as the counterpart of BB-wise recombination simulated by sampling candidate solutions from the probabilistic model. As mentioned above, the decision making involved in time continuation/utilization can also be posed as choosing between these two key operators. The robust behavior that characterizes the integration of substructural local search in EDAs will implicitly allow to switch between a global and local search operator based on problem requirements. This behavior introduces what is named as *adaptive time continuation*.

- The estimation of fitness is also considered to evaluate the local search steps in BOA. Previous results using the probabilistic models of EDAs to induce

the functional form of surrogates have shown this approach to be surprisingly accurate. Given that substructural local search looks for the most promising BBs, the choice of a model-based surrogate to evaluate the quality of each alternative becomes natural.

2.6 Summary

This chapter reviews essential topics to understand and situate the work presented in the thesis. It starts by introducing genetic algorithms and its basic components inspired on genetics and natural selection. While the importance of having scalable GAs is highlighted, a design methodology for competent GAs—GAs that solve boundedly difficult problems quickly, reliably, and accurately—is introduced, which is followed by the definition of boundedly-difficult problems used in the thesis to validate the proposed methods. Estimation of distribution algorithms are introduced by a brief survey on different types of discrete EDAs. A connection between EDAs and other model-based search methods is also made, recognizing that EDAs are a more general framework than other MBS approaches. The chapter ends by enumerating some of the efficiency-enhancement techniques that can be used in GAs and particularly in EDAs. The remainder of the thesis addresses most of these techniques.

Chapter 3

Substructural Local Search in eCGA

3.1 Introduction

The extended compact genetic algorithm (eCGA) divides decision variables into non-overlapping clusters, where each cluster is taken as a whole when recombining different solutions. Therefore, each cluster of variables represents a building-block or a substructural partition.

This chapter presents the integration of a *competent* recombination operator, that effectively exchanges key partial solutions of the problem, with a *competent* local search method that efficiently searches for the best subsolutions. Specifically, the probabilistic model of eCGA is used to identify an effective problem decomposition and important substructures. The probabilistic model, which automatically induces good neighborhoods, is subsequently used for two distinct purposes:

1. Effective recombination of BBs that provides rapid global-search capabilities.

2. Effective search in the BB neighborhood that locally provides high-quality

solutions.

The key idea is to obtain the benefits from both approaches, recombination without disrupting important BBs, and mutation (local search) that rapidly searches for the best BBs in each partition.

The next section introduces the extended compact genetic algorithm, while its selectomutative counterpart is described in the subsequent section. Section 3.4 proposes the incorporation of substructural local search in eCGA and outlines other possible applications of substructural mutation. In Section 3.5, computational experiments are performed in different problem difficulty dimensions to evaluate the performance of the hybrid approach. The chapter ends with a discussion about the advantages and limitations of such approach, followed by a brief summary.

3.2 Extended Compact Genetic Algorithm

The extended compact genetic algorithm [65, 63] was designed with the idea that the choice of a good probability distribution for promising solutions is equivalent to linkage learning. The eCGA uses a product of marginal distributions on a partition of variables. This type of probability distribution belongs to a class of probability models known as marginal product models (MPMs). For example, the following MPM, `[1,3][2][4]`, for a 4-bit problem represents that the 1^{st} and 3^{rd} variables are linked, and the 2^{nd} and 4^{th} variables are independent.

In eCGA, both the structure and the parameters of the model are searched and optimized to best fit the data (promising solutions). The measure of a good MPM is quantified based on the minimum description length (MDL) principle [140], that penalizes both inaccurate and complex models, thereby leading to an optimal distribution. According to this principle, good distributions are those under which

the representation of the distribution using the current encoding, along with the representation of the population compressed under that distribution, is minimal.

More precisely, the MPM complexity is given by the sum of model complexity C_m and compressed population complexity C_p. The model complexity C_m quantifies the model representation in terms of the number of bits required to store all the marginal probabilities. Let a given problem of size ℓ with binary encoding, have m partitions with k_i variables in the i^{th} partition, such that $\sum_{i=1}^{m} k_i = \ell$. Then each partition i requires $2^{k_i} - 1$ independent frequencies to completely define its marginal distribution. Taking into account that each frequency is of size $\log_2(n+1)$, where n is the population size, the model complexity C_m is given by

$$C_m = \log_2(n+1) \sum_{i=1}^{m} (2^{k_i} - 1). \tag{3.1}$$

The compressed population complexity quantifies the data compression in terms of entropy of the marginal distribution over all partitions. Therefore, C_p is given by

$$C_p = n \sum_{i=1}^{m} \sum_{j=1}^{2^{k_i}} -p_{ij} \log_2(p_{ij}), \tag{3.2}$$

where p_{ij} is the frequency of the j^{th} gene sequence of the genes belonging to the i^{th} partition. In other words, $p_{ij} = N_{ij}/n$, where N_{ij} is the number of chromosomes in the population (after selection) possessing bit sequence $j \in [1, 2^{k_i}]$ for the i^{th} partition. Note that a substructural partition of size k has 2^k possible bit sequences where the first is denoted by 00...0 and the last by 11...1.

The eCGA performs a greedy MPM search at every generation. The greedy search starts with the simplest possible model, assuming that all variables are independent (like in the compact GA [67]), and then keeps merging partitions of genes

MPM Greedy Search

(1) Assign each variable to an independent partition.

(2) Compute the MDL metric for the current model `M`.

(3) For all possible merges of two partitions in `M`:

  (3.1) Compute the MDL metric for each resulting model structure.

(4) Let `M'` be the resulting model with lowest metric value.

(5) If the metric value of `M'` is lower than that of `M`, `M'` becomes `M` and return to step 3, otherwise finish search with model `M` as the most adequate.

Figure 3.1: Pseudocode of the marginal product model (MPM) greedy search.

whenever the MDL score metric is improved. This process goes on until no further improvement is possible. An algorithmic description of this greedy search can be found in Figure 3.1.

Looking at Figure 3.2, it can be seen that eCGA is similar to a traditional GA, where the variation operators (crossover and mutation) are replaced by the probabilistic model building and sampling procedures. The offspring population is generated by randomly choosing subsets from the current individuals, according to the probabilities of the subsets stored in the MPM.

Analytical models have been developed for predicting the scalability of EDAs [128, 134]. In terms of number of fitness evaluations necessary to converge to the optimal solution, these models predict that for additively separable problems eCGA scales subquadratically with the problem size. Sastry and Goldberg [149] empirically verified this scale-up behavior for eCGA.

***Extended Compact Genetic Algorithm* (eCGA)**

(1) Create a random population `P` of n individuals.

(2) Evaluate population `P`.

(3) Select `P'` individuals from `P` using a selection procedure.

(4) Model the selected individuals `P'` by learning the most adequate marginal product model `M`.

(5) Generate a new population `O` by sampling from the joint probability distribution of `M`.

(6) Evaluate population `O`.

(7) Replace all (or some) individuals in population `P` by those from `O`.

(8) If stopping criteria are not satisfied, return to step 3.

Figure 3.2: Pseudocode of the extended compact genetic algorithm (eCGA).

3.3 Extended Compact Mutation Algorithm

In genetic algorithms, significant attention has been paid to the design and understanding of recombination operators. Systematic methods for successfully designing competent selectorecombinative GAs have been developed based on decomposition principles [49, 51].

Mutation, on the other hand, is usually a secondary search operator which performs a random walk locally around a solution and therefore has received far less attention. However, in evolutionary strategies [138], where mutation is the primary search operator, significant attention has been paid to the development of mutation operators. Several mutation operators, including adaptive techniques, have been proposed [138, 157, 6, 13, 61]. The mutation operators used in evolution strategies are powerful search operators; however, the neighborhood information is still local around a single solution. In fact, when solving boundedly difficult problems, local neighborhood information is not sufficient, and a mutation operator which uses local

neighborhood requires $\Theta(\ell^k \log \ell)$ function evaluations [115], which even for moderate values of k, grows extremely fast and the search becomes inefficient compared to competent GAs.

The building-block-wise mutation algorithm (BBMA) [149] is a selectomutative algorithm that performs local search in the BB neighborhood. Instead of using a bitwise mutation operator that scales polynomially with order k as the problem size increases, BBMA uses a substructural mutation operator that scales subquadratically, as shown by Sastry and Goldberg [149].

The main idea is to use the substructural information—obtained by estimating the distribution of promising solutions—to perform a local search that exploits decomposition principles. For instance, consider again the same problem of size ℓ with binary encoding, that has m partitions with k_i variables in the i^{th} partition, such that $\sum_{i=1}^{m} k_i = \ell$. According with this decomposition, each individual will have m substructural neighborhoods, each of size $2^{k_i} - 1$. Therefore, the i^{th} neighborhood is composed by the set of solutions that contain the remaining possible configurations for the i^{th} substructural partition, while variables outside the partition are kept unchanged. The exploration of all m neighborhoods requires a total of $\sum_{i=1}^{m}(2^{k_i} - 1)$ fitness evaluations.

For substructural identification, the probabilistic model building procedure of eCGA is used in this chapter. However, other probabilistic models or linkage learning methods can also be used. Since this chapter focus on eCGA, from now on this instance of BBMA is referred as the extended compact mutation algorithm (eCMA). Once the linkage groups are identified, an enumerative substructural mutation operator is used to find the best subsolution for each detected partition. A description of eCMA can be seen in Figure 3.3.

The performance of eCMA can be slightly improved by using a greedy heuristic to

***Extended Compact Mutation Algorithm* (eCMA)**

(1) Create a random population `P` of n individuals.

(2) Evaluate population `P`.

(3) Select `P'` individuals from `P` using a selection procedure.

(4) Model the selected individuals `P'`by learning the most adequate marginal product model `M`.

(5) Choose the best individual of the population `Best` for substructural mutation.

(6) For each partition of variables in `M`:

  (6.1) Create $2^k - 1$ unique individuals with all possible remaining values for the current partition, while other variables are kept as in `Best`.

  (6.2) Evaluate all $2^k - 1$ individuals and retain the best for mutation in the other partitions. This individual becomes the `Best`.

(7) Return `Best` as the final result.

Figure 3.3: Pseudocode of the extended compact mutation algorithm (eCMA).

search for the best among competing BBs in each partition. Even so, the scalability of eCMA is determined by the population size required to accurately identify the BB partitions. Therefore, the number of function evaluations scales at least as $\Omega(2^k m^{1.05})$ and at most as $O(2^k m^{2.1})$ [149, 134]. Recently, the model which estimates the population size required to obtain accurate substructural information has been refined to a more precise $\Theta(2^{2k} m \log m)$ [177].

It should be also noted that on eCMA the linkage identification is only done at the initial stage. This kind of offline linkage identification works well on problems of nearly equal salience, however, for problems with non-uniformly scaled BBs, the linkage information needs to be updated at regular intervals. This limitation will be empirically shown with experimental results (Section 3.5.2).

3.4 eCGA with Substructural Local Search

After presenting two *competent* operators based on the probabilistic model of eCGA, this chapter investigates the combination of both operators in the same algorithm [104]. Similar to eCGA, the hybrid extended compact genetic algorithm (heCGA) models promising solutions in order to effectively recombine important substructures and perform effective local search in the substructural space.

The incorporation of substructural local search (SLS) in eCGA is done after learning the adequate MPM for the present generation, and before sampling new individuals from the model (which mimics recombination). This contrasts with the typical approach in hybrid GAs where local search improves the solutions produced by crossover. The motivation for doing so is twofold. First, it allows to replicate the procedure of eCMA before sampling and evaluating a new population. This will be particularly useful for problems where eCMA is known to be superior to eCGA (and other selectorecombinative GAs) [150]. Second, the traits learned by local search update the model before new individuals are generated, thereby biasing the generation of the offspring population in the present generation, instead of only in the next generation.

Figure 3.4 shows that heCGA starts like the regular eCGA (steps 1-4), but after building the model the linkage information is used to perform substructural local search in the best individual of the population. Once a result is obtained, heCGA updates the substructural frequencies of the model according to the values of the recently improved best solution. This is done by increasing the sampling probability for each substructure of the new best individual by s/n and decreasing each substructure of the previous best solution (before performing local search) by s/n, where s is the tournament size. Note that the copies of the best individual can

***Hybrid Extended Compact Genetic Algorithm* (heCGA)**

(1) Create a random population `P` of n individuals.

(2) Evaluate population `P`.

(3) Select `P'`individuals from `P` using a selection procedure.

(4) Model the selected individuals `P'` by learning the most adequate marginal product model `M`.

(5) Apply substructural local search to the best individual `Best` of population `P`. The resulting individual is `NewBest`.

(6) Update model `M` according to the new substructures found in `NewBest`. For each partition of variables in `M`:

  (6.1) Increase the frequency of the substructure present in `NewBest` by s/n.

  (6.2) Decrease the frequency of the substructure present in `Best` by s/n.

(7) Generate a new population `O` by sampling from the joint probability distribution of the updated model `M`.

(8) Replace all (or some) individuals in population `P` by those from `O`.

(9) If stopping criteria are not satisfied, return to step 3.

Figure 3.4: Pseudocode of the hybrid extended compact genetic algorithm (heCGA). The parameter s is the tournament size.

also be replaced by the mutated one, with a similar overall effect[1]. Alternatively, a user-defined parameter can be introduced. Finally, a new population is generated according to the updated model. If the stopping criteria are not satisfied, the algorithm returns to step 3.

Other ways to integrate substructural mutation with substructural recombination can also be considered. An alternative approach to combine these operators would be to apply stochastic substructural mutation to all individuals in the population. In this way, instead of having the traditional bitwise mutation with a certain

[1]Notice that if tournament selection without replacement is used the best individual gets exactly s copies in the mating pool.

probability to be applied to each bit, it would be possible to have a substructural mutation with a certain probability to be applied to each substructure in each individual. Still, the impact of this kind of mutation will be slower than that of the proposed method. Although superior BBs can still be discovered by applying stochastic substructural mutation in the population, the combination of such BBs (that form good solutions) will take a certain number of generations known as *mixing time* [165, 51]. By applying local search to all substructural neighborhoods in the same individual, the resulting subsolutions will be instantly combined.

The additional cost in terms of function evaluations for doing substructural local search should also be taken into account when considering other hybridization configurations. For the proposed scheme, approximately $2^k m$ additional evaluations are spent in each generation[2]. Here, k and m refer to the decomposition learned by the probabilistic model, which is not necessarily the real decomposition of the problem. For the stochastic scheme mentioned above, the cost depends on the probability of applying substructural mutation. If one allows to spend the same $2^k m$ evaluations on average, the probability of applying stochastic mutation should have to be set around $p_m = 1/n$ (recall that n is the population size). This is a fairly low mutation rate (even in the substructural sense) and the high-quality substructures acquired by mutation will take long (if ever) to be mixed. If a stronger mutation rate is considered, for instance $p_m = 1/m$, on average n/m more evaluations will be spent than with the proposed method.

As mentioned earlier, the performance of the BB-wise mutation operator can be slightly improved using a greedy procedure to search for the best among competing substructures. This can be particularly useful for the application of stochastic mutation, especially for higher mutation rates. For the case of performing SLS it is

[2]Note that $\sum_{i=1}^{m}(2^{k_i} - 1) \approx 2^k m$, if $k_1 = k_2 = \ldots = k_m$.

not desirable to apply greedy search, since failing possible subsolutions (due to the nature of the greedy search) will probably be much costlier than what is about to be saved.

Finally, it should be noted that eCGA, and consequently eCMA and heCGA, can only build linkage groups with non-overlapping variables. However, the ideas here presented for integrating SLS can be extended to other linkage identification techniques that can handle overlapping BBs such as the Bayesian optimization algorithm or the dependency structure matrix driven genetic algorithm (DSMGA) [175]. The next chapter addresses the application of substructural local search in BOA.

3.5 Experiments

This section performs computational experiments with various problems of bounded difficulty. Following a design approach to problem difficulty [51], the described algorithms are tested on a set of problems that combine the core of three important problem difficulty dimensions: (1) *deception*, (2) *scaling*, and (3) *noise*.

For each algorithm, it is empirically determined the minimal number of function evaluations to obtain a solution with at least $m - 1$ building blocks solved (optimal solution with an error of $\alpha = 1/m$). For eCGA and eCMA, a bisection method [145, 125] over the population size is used to search for the minimal population size necessary to find a target solution. However, for heCGA an interval halving method [30] is more appropriate given the algorithm behavior as the population increases, as will be shown later (Figure 3.6). The results for the minimal sufficient population size are averaged over 30 bisection runs. In each bisection run, the solution quality obtained with a given population size is averaged over another 30 runs. Thus, the results for the number of function evaluations and the num-

ber of generations spent are averaged over 900 (30x30) independent runs. For all experiments, tournament selection without replacement is used with size $s = 8$.

3.5.1 Problem 1: Deception

As the core of intra-BB difficulty, deceptive functions are among the most challenging problems for competent GA candidates. Figure 3.5 presents the results obtained for the m-k trap problem. The number of BBs (or subfunctions) m is varied between 2 and 20, for size $k = \{4, 5\}$.

While eCGA needs smaller populations than eCMA and heCGA to solve the problem, it takes more function evaluations than both algorithms. This happens because in eCGA (1) the BBs are discovered in a progressive way and (2) more generations are required to exchange the right BBs. Although increasing the population size for eCGA accelerates the BB identification process, additional generations are still needed to mix the correct BBs into a single individual. Since eCGA (like every selectorecombinative GA) requires this mixing time, relaxing the BB identification process (using smaller populations, thus saving function evaluations) to a certain point seems to be the best way to tune eCGA performance.

The scalability difference between eCGA and eCMA is not surprising and was verified before [149, 150]. The similarity between eCMA and heCGA performances supports that the best way to use heCGA on deterministic and uniformly-scaled boundedly deceptive functions (and problems bounded by this one), is to set a large enough population size to get the problem structure in the first generation(s), and then perform substructural local search to rapidly achieve the global optimum.

These results suggest that there is no direct gain of heCGA over eCMA for this problem; however, another observation can be made. From a practitioner point of view, heCGA is a more flexible search algorithm because it is able to solve the prob-

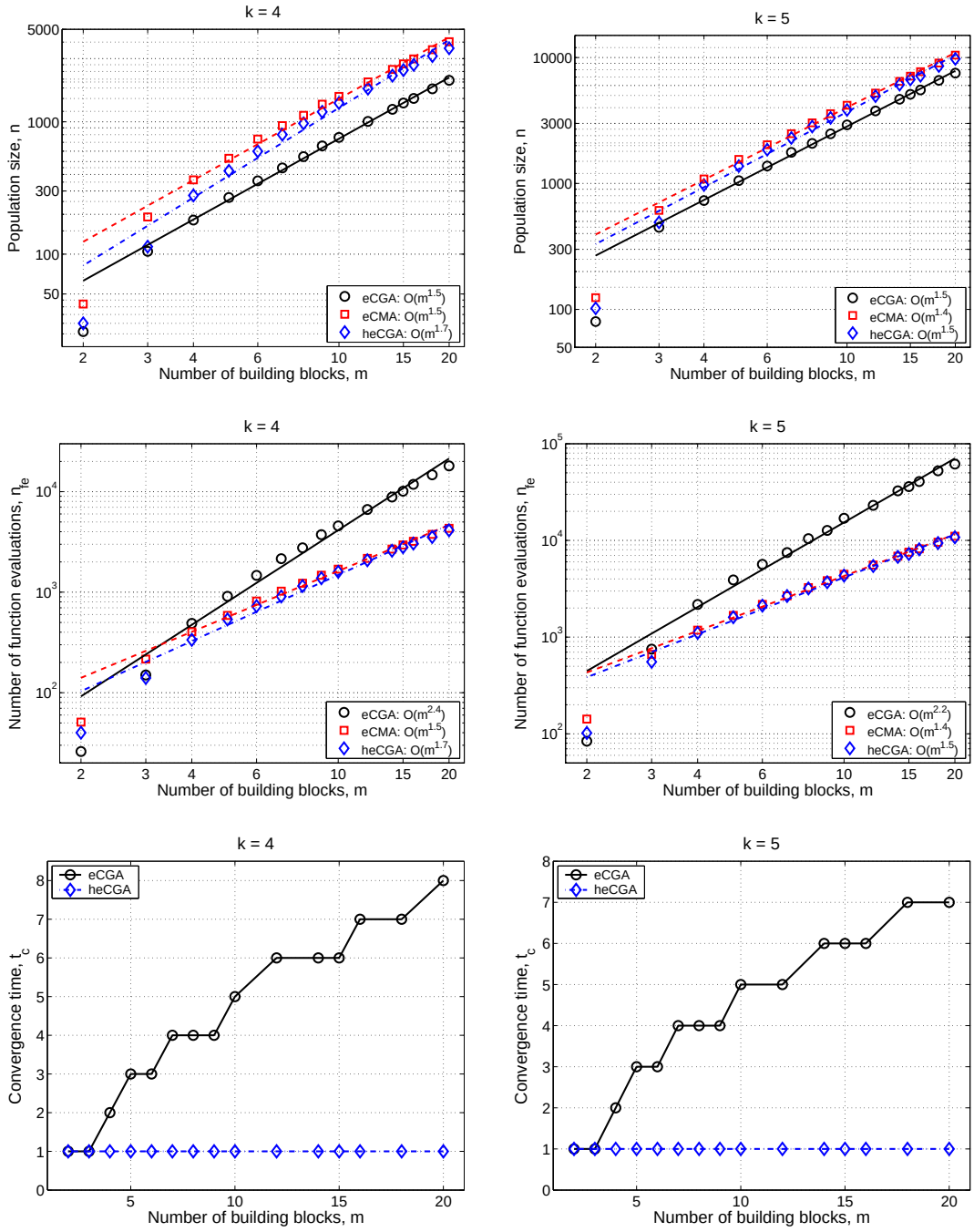


Figure 3.5: Population size (top), number of function evaluations (middle), and number of generations (bottom) required by eCGA, eCMA, and heCGA for the m-k trap problem ($m = [2, 20]$ and $k = \{4, 5\}$).

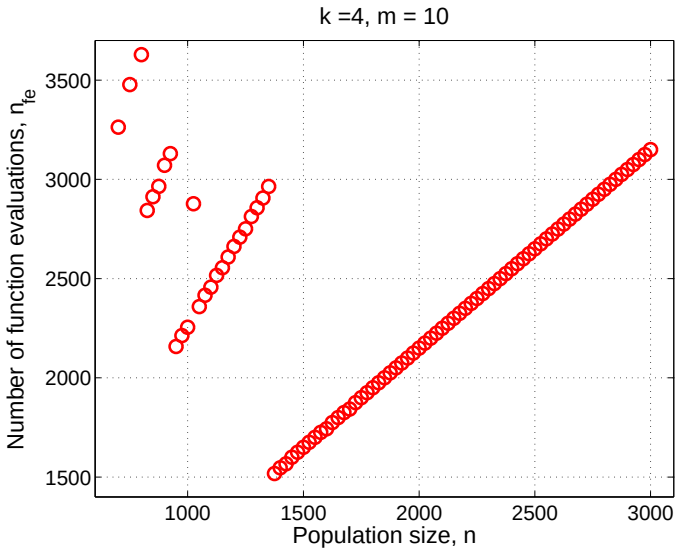


Figure 3.6: Number of function evaluations required by heCGA for the m-k trap problem ($k = 4$ and $m = 10$) when using increasing population sizes.

lem within a larger range of population size values. Figure 3.6 shows the number of function evaluations for heCGA to find the target solution ($k = 4$ and $m = 10$), as the population size increases. Note that only population sizes that find the target solution (over 30 independent runs) are shown. The points plotted form four increasing lines. In each line, as the population increases the number of function evaluations also increases until it falls down into a lower line and then keeps increasing again. This behavior is cyclic until the population size is enough to discover all[3] correct substructural partitions in the first generation, being the problem solved by the SLS procedure in the initial generation. Each discontinuity between lines represents a decrease in the number of generations necessary for heCGA to successfully solve the problem. This happens because, as the population size increases, the model building procedure can capture more correct substructural partitions, improving the ability of local search to quickly solve the problem.

[3]In this specific case, $m - 1$ partitions due to the stopping criterion used.

3.5.2 Problem 2: Deception + Scaling

In this problem, the inter-BB difficulty is explored together with the intra-BB difficulty. Specifically, the same m-k trap problem is used, but now each subfunction fitness contribution to the overall fitness is exponentially scaled. The weight of each subfunction fitness contribution is given by powers of 2, being the resulting exponentially scaled problem already defined in Equation 2.3.

This function has the interesting property that a high-scaled subfunction gives more fitness contribution than the sum of all inferior subfunctions. When solving this problem with a GA, in the initial generations the signal that comes from the low-salient BBs is negligible when faced with the decision making between the high-salient BBs. Whenever the higher BBs are solved, the next highly-scaled BBs will have their time of attention by the GA, and so on. Given this property, the correct substructural partitions can only be discovered in a sequential way, which contrasts with the uniformly-scaled case where the problem structure can be captured in the first generation(s) with a sufficiently large population size. Thus, eCMA is not able to solve exponentially-scaled problems with reasonable population sizes, as was pointed out before [149]. The model built based on the selected initial individuals will only be able to get the high-salient substructures, failing the rest. Therefore, the model of eCMA has to be updated at a regular schedule to capture the problem structure in a sequential manner.

Figure 3.7 empirically shows that eCMA needs prohibitively large population sizes to achieve the target solution. In heCGA the model is updated every generation and the substructural local search can benefit from that. Nevertheless, heCGA spends approximately the same number of function evaluations to solve the problem as the regular eCGA. In this case, heCGA behaves similarly to eCGA, preferring a reasonable population size, enough to get the most relevant substructures and then

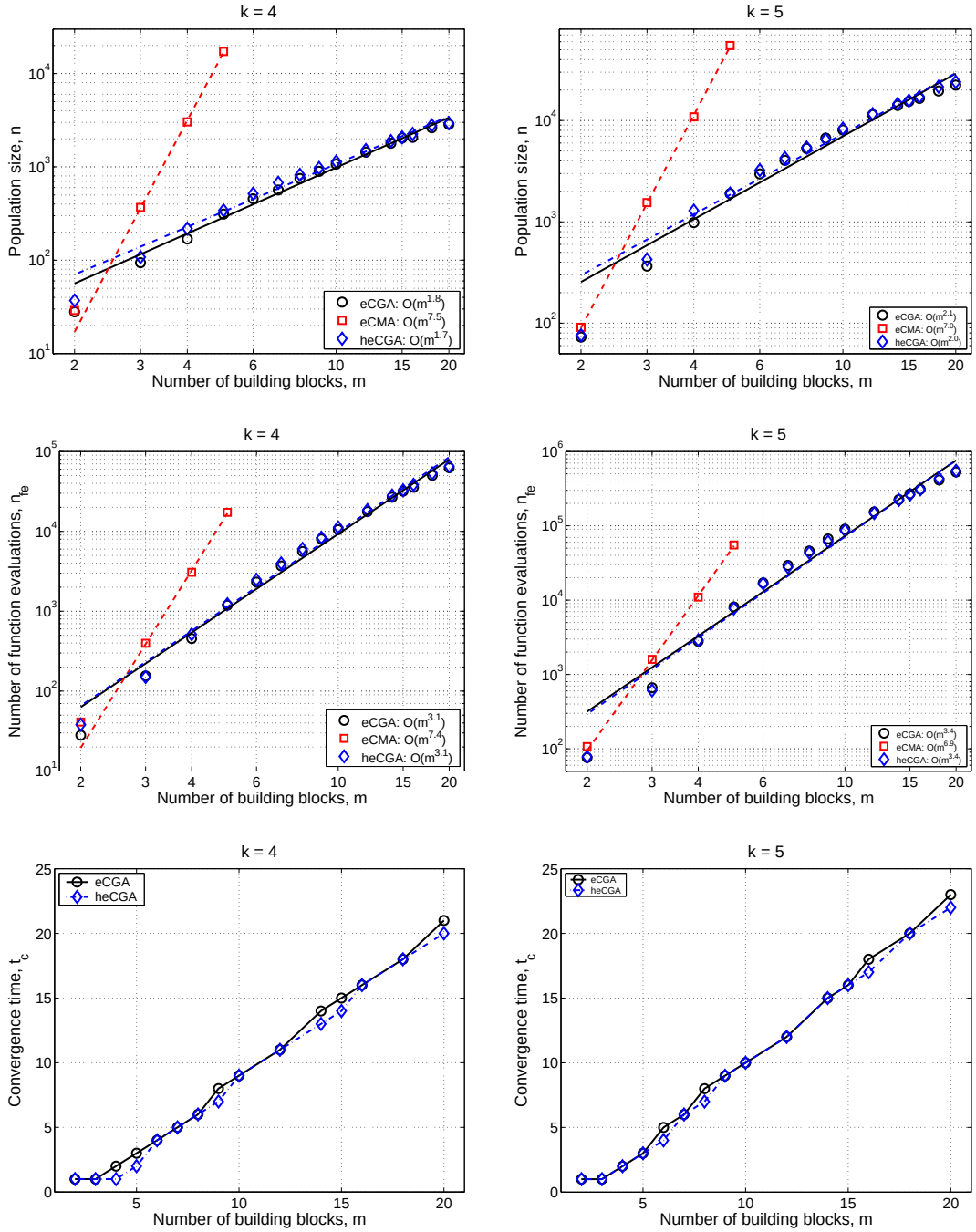


Figure 3.7: Population size (top), number of function evaluations (middle), and number of generations (bottom) required by eCGA, eCMA, and heCGA for the exponentially-scaled m-k trap problem ($m = [2, 20]$ and $k = \{4, 5\}$).

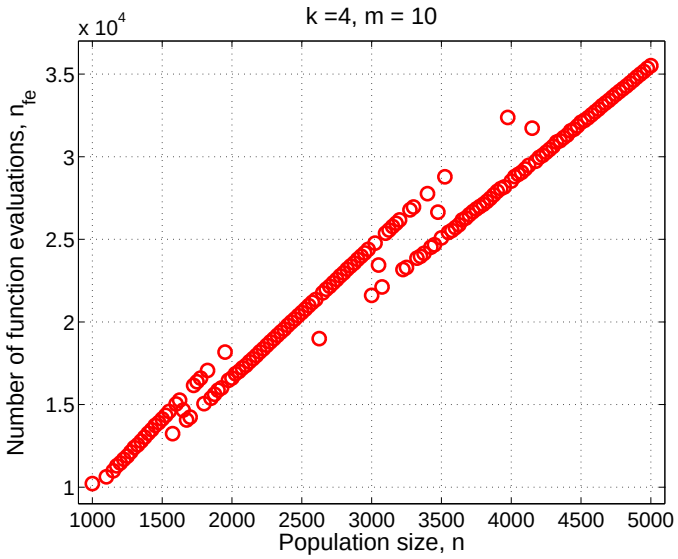


Figure 3.8: Number of function evaluations required by heCGA for the exponentially-scaled m-k trap problem ($k = 4$ and $m = 10$) when using increasing population sizes.

keep going sequentially to the remaining ones. In terms of number of generations, heCGA scales as the eCGA, increasing linearly with the problem size.

Figure 3.8 shows the number of function evaluations that heCGA needs to solve this problem as the population size increases. In this case, the number of function evaluations scales approximately linearly with the population size. Since increasing the population size will not reveal much more correct substructural information, the effect on the overall search process is minor.

3.5.3 Problem 3: Deception + Noise

Noise is a common feature in many real-world optimization problems. Sources of noise can include physical measurement limitations, incomplete sampling of large spaces, stochastic simulation models, human-computer interaction, among others [3]. Furthermore, evaluation-relaxation techniques [145, 83] are commonly used in evolutionary algorithms for performance enhancement, bringing an additional source of

noise to the original optimization problem. Thus, analyzing heCGA performance in noisy environments is important to investigate its behavior when facing the core of the extra-BB difficulty dimension. For the experiments, it is assumed that the exogenous noise follows a Gaussian distribution with mean 0 and variance σ_N^2. To make the problem even more challenging, a noisy version of the uniformly-scaled deceptive function is considered (see Equation 2.4).

To overcome the noise with eCMA, the evaluation of a solution in the local search phase needs to be performed over an average of function evaluations. The number of times that each individual needs to be evaluated, to allow correct decision making between competing substructures, depends on the noise variance. Therefore, to obtain the optimal results for eCMA in noisy conditions two different bisections runs need to be performed, one over the initial population size and the other over the number of fitness samples that is necessary to correctly evaluate an individual. First, the bisection method is used to find the minimal population size that generates a model with at least $m - 1$ correct substructural partitions. Then, for each population that captures the target dependencies, a second bisection method is performed over the number of fitness samples to obtain the minimal number of times that an individual needs to be evaluated, in order to achieve a final solution with the subproblems detected by the model optimally solved.

Figure 3.9 depicts the results obtained for a uniformly-scaled m-k trap problem with additive noise for $k = 4$ and $m = \{5, 10\}$. As the noise-to-signal ratio σ_N^2/σ_f^2 increases, two different scenarios can be identified. For small values of noise, as $\sigma_N^2/\sigma_f^2 \to 0$, the scenario is somewhat similar to the deterministic case, where eCMA and heCGA perform better than eCGA. However, in this case eCMA performs slightly worse than heCGA (this is particularly true for small m, see $m = 5$). This occurs because the MPM is required to detect at least $m - 1$

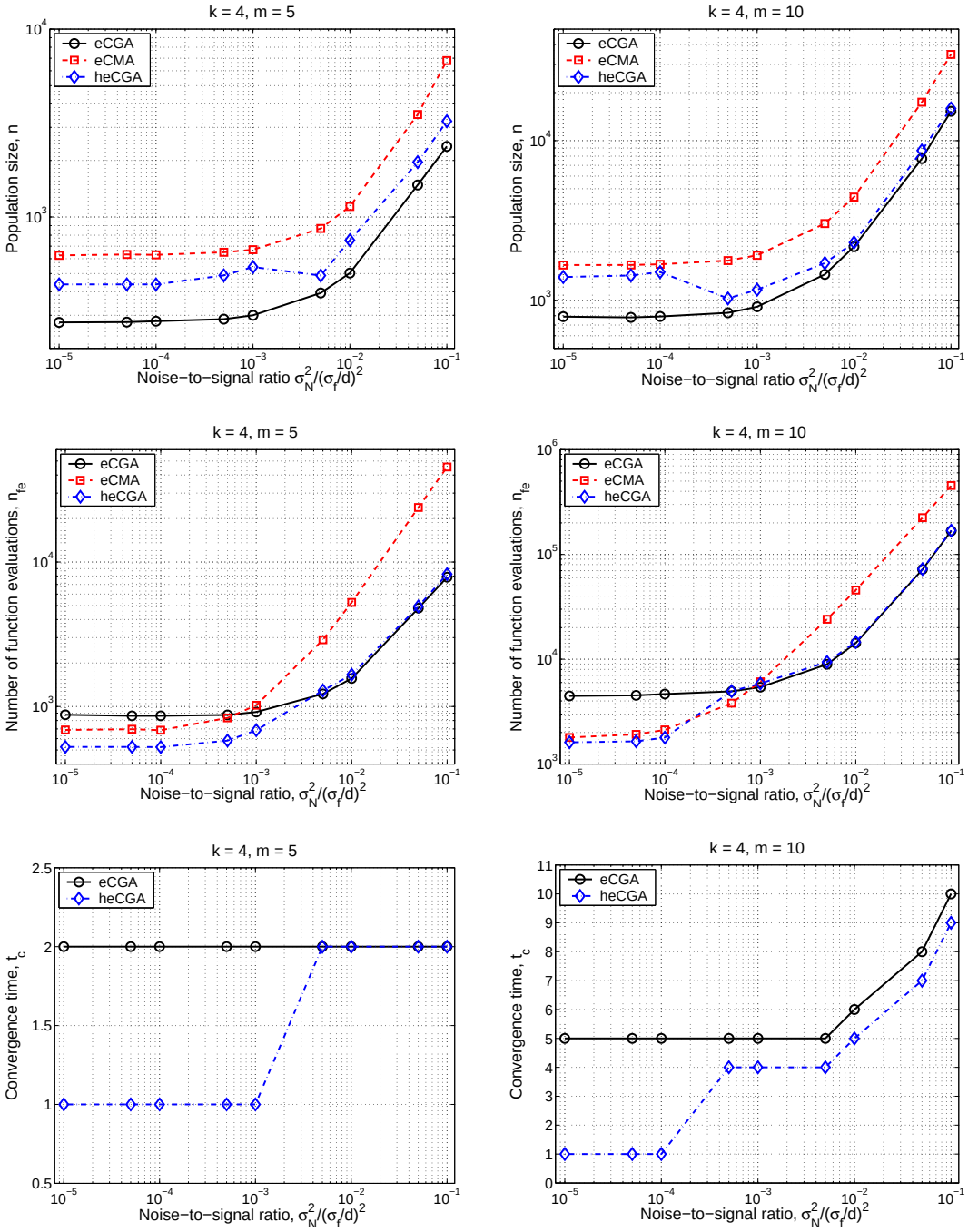


Figure 3.9: Population size (top), number of function evaluations (middle), and number of generations (bottom) required by eCGA, eCMA, and heCGA for the noisy m-k trap problem ($m = \{5, 10\}$ and $k = 4$), with varying noise-to-signal ratio σ_N^2/σ_f^2.

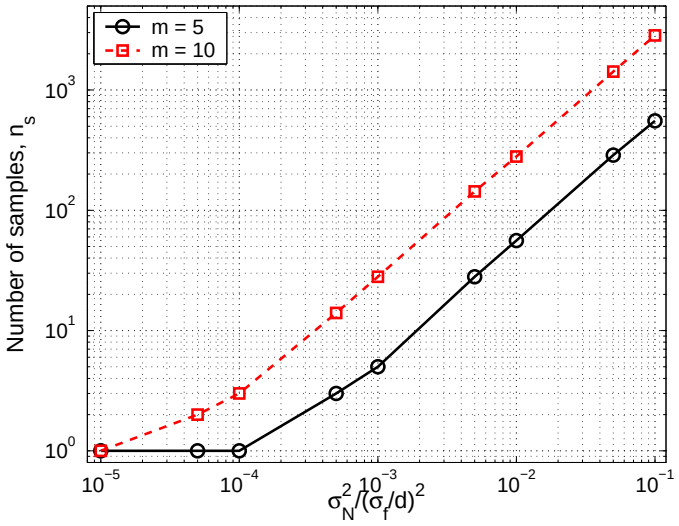


Figure 3.10: Number of fitness samples (n_s) used in each averaged fitness evaluation in eCMA for the noisy m-k trap problem ($m = \{5, 10\}$ and $k = 4$).

partitions, while in the deterministic case this criteria is only applied to the target solution.

When noise increases the behavior of the algorithms changes. Considering the tested cases, $k = 4$ and $m = \{5, 10\}$, the scenario changes around $\sigma_N^2/\sigma_f^2 = 0.001$. At this point, eCGA starts to perform better than eCMA, which is expected given that crossover is likely to be more useful than mutation in noisy environments [150]. However, heCGA, which was behaving like eCMA (using larger population sizes to quickly solve the problem) to small noise values, starts performing similarly to eCGA, that is known to be a best approach than eCMA for moderate-to-high noise values. This change in heCGA behavior can be better observed in the population size and number of generations plots.

In Figure 3.10, it is shown the minimal number of fitness samples necessary to eCMA correctly decide between competing substructures in noisy conditions. Note that in heCGA the substructural local search does not take advantage of the averaging technique used in eCMA, because the objective is to test heCGA in various

difficulty dimensions as a black-box method. Based on these results, the robust behavior of heCGA stands for noisy conditions as well, confirming the observations made for the first two problems.

3.6 Discussion

Looking at the behavior of heCGA on both uniformly and exponentially scaled problems, distinct dynamics can be observed for each situation. In the uniformly-scaled case, heCGA has a similar behavior to eCMA, which is the algorithm that performs better. For the exponentially-scaled problem, heCGA changes completely its dynamics behaving like eCGA, which is known to perform much better than eCMA. While in both cases there is no direct gain in using heCGA over the best algorithm, it becomes clear what seems to the greatest advantage of the proposed approach: robustness. For both problems, heCGA obtains a very similar performance to the best algorithm for each domain.

A better insight on this robust behavior can be obtained by looking at the results for the problem with additive exogenous noise. Depending on the noise amount, eCGA which integrates SLS is able to follow the performance of the best single-operator-based approach. This is indeed quite important in the absence of domain- or problem-specific knowledge.

As previously mentioned, time continuation in evolutionary algorithms deals with the tradeoff between using a large population for a single convergence epoch or a small population for multiple convergence epochs. For the latter, a continuation operator is then required to inject diversity between epochs. Typically, this operator is assumed to be some form of mutation or local search method. Therefore, the decision making involved in time continuation can also be posed as choosing

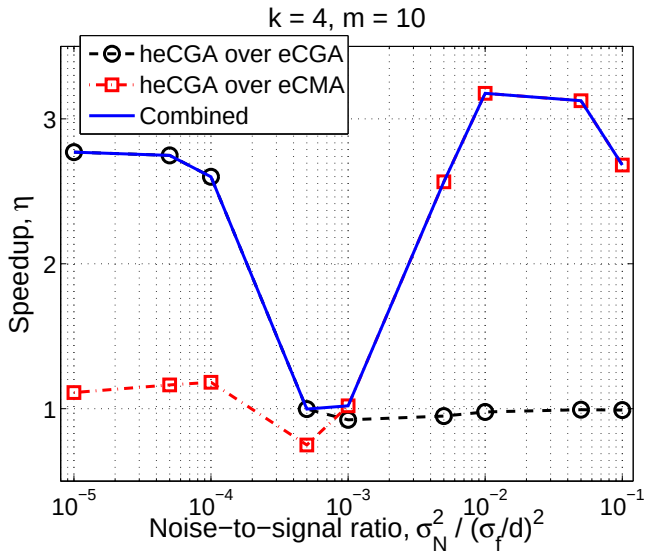


Figure 3.11: Speedup obtained when using substructural local search in eCGA for the noisy m-k trap problem ($k = 4$ and $m = 10$). heCGA obtains significant savings when compared with the least adequate single-operator-based approach for both scenarios (low or high values of noise). This *adaptive* behavior can be achieved by setting the population size accordingly.

between two key variation operators—recombination and mutation [155]. From this standpoint, heCGA introduces what can be named as *adaptive time continuation*.

The incorporation of SLS in eCGA implicitly allows to switch between a global and local search operator based on problem requirements. This can result in significant savings that depend upon the type of problem being solved. For example, consider the previous noisy problem. Figure 3.11 shows the speedup (ratio in the number of evaluations) of heCGA over the single-operator-based approaches. Clearly, heCGA obtains substantial speedups over the less adequate approach, even for a moderate problem size ($k = 4$ and $m = 10$). This *adaptive* behavior can be achieved by setting the population size accordingly.

While the experiments focused on problems with non-overlapping subfunctions, many problems can have different substructures that share common components. The effect of overlapping interactions between variables has been studied before [51],

where it was argued to be similar to that of exogenous noise. However, the probabilistic model used by the algorithms in this chapter (marginal product models) can not capture overlapping dependencies. For that purpose, the more powerful Bayesian optimization algorithm [127, 125] will be considered in the next chapter, where the utility of incorporating SLS will be investigated as well. Additionally, the use of a surrogate fitness model for evaluating local search steps will also be investigated.

3.7 Summary

This chapter integrates substructural local search in eCGA. The resulting algorithm—the hybrid extended compact genetic algorithm (heCGA)—combines the substructural recombination operator from eCGA with a substructural mutation operator that is also based on the probabilistic model of eCGA. Essentially, heCGA makes use of *mined* substructural information to perform (1) effective recombination of subsolutions and (2) effective local search in the substructural search space. Experiments are performed on three different test functions that combine important problem difficulty dimensions: deception, scaling, and noise. The results show that, independently from the faced difficulty dimension(s), the eCGA which integrates SLS obtained the most robust performance, mimicking the behavior of the best approach (recombination-based or mutation-based) for each problem.

Additionally, the incorporation of substructural local search in eCGA implicitly allows to switch between a global and local search operator based on problem requirements. This form of adaptive time utilization can result in significant savings even for moderate problem sizes, as it is empirically shown for the noisy problem.

Overall, the results presented in this chapter indicate the robustness of using

both search operators—recombination and mutation—in the context of EDAs, as it is known to be advantageous for traditional evolutionary algorithms.

Chapter 4

Substructural Local Search in BOA

4.1 Introduction

The Bayesian optimization algorithm (BOA) is an EDA that uses Bayesian networks (BNs) as the probabilistic model. In BNs, the interactions between variables are expressed under conditional dependencies represented by a directed acyclic graph. Similar EDAs include the estimation of Bayesian networks algorithm and the learning factorized distribution algorithm. Bayesian networks are more powerful probabilistic models than MPMs because they allow to represent more complex non-linearities such as overlapping and hierarchical dependencies.

This chapter introduces the concept of substructural local search for Bayesian EDAs. The substructural neighborhoods explored in local search are defined by the conditional dependencies learned by the Bayesian networks. Additionally, a surrogate fitness model that also makes use of substructural information is used to evaluate the alternatives while performing hillclimbing in the subsolution search space.

In this way, the cost of performing local search can be reduced to a small number of fitness evaluations, while providing effective learning of important subsolutions.

The chapter starts by introducing the Bayesian optimization algorithm, followed by a description of fitness modeling with Bayesian networks. In Section 4.4, several substructural neighborhoods are considered, and the most adequate is chosen. The incorporation of substructural local search in BOA is presented in Section 4.5. Section 4.6 and 4.7 present and discuss empirical results. The chapter finalizes with a brief summary.

4.2 Bayesian Optimization Algorithm

The Bayesian optimization algorithm [127, 125] uses Bayesian networks to capture the (in)dependencies between the decision variables of the optimization problem. In BOA, the traditional crossover and mutation operators of evolutionary algorithms are replaced by (1) building a BN which models promising solutions and (2) sampling from the corresponding probability distribution to generate new solutions. The pseudocode of BOA is detailed in Figure 4.1.

Bayesian networks [124] are powerful graphical models that combine probability theory with graph theory to encode probabilistic relationships between variables of interest. A BN is defined by its structure and corresponding parameters. The structure is represented by a directed acyclic graph where the nodes correspond to the variables of the problem and the edges correspond to conditional dependencies. The parameters are represented by the conditional probabilities for each variable given any instance of the variables that this variable depends on. More formally, a

Bayesian Optimization Algorithm (BOA)

(1) Create a random population `P` of n individuals.

(2) Evaluate population `P`.

(3) Select `P'` individuals from `P` using a selection procedure.

(4) Model the selected individuals `P'` by learning the most adequate Bayesian network `B`.

(5) Create a new population `O` by sampling from the joint probability distribution of `B`.

(6) Evaluate population `O`.

(7) Replace all (or some) individuals in population `P` by those from `O`.

(8) If stopping criteria are not satisfied, return to step 3.

Figure 4.1: Pseudocode of the Bayesian optimization algorithm.

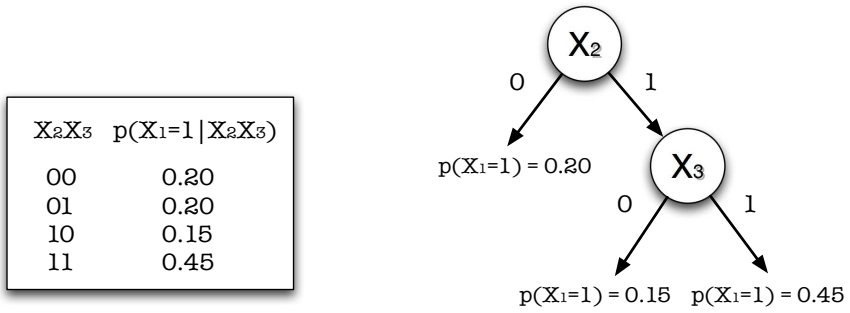
Bayesian network encodes the following joint probability distribution,

$$p(X) = \prod_{i=1}^{\ell} p(X_i|\Pi_i), \tag{4.1}$$

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

The parameters of a Bayesian network can be represented by a set of conditional probability tables (CPTs) or local structures. Using local structures such as decision trees (DTs) allows a more efficient and flexible representation of local conditional distributions, improving the expressiveness of BNs [25, 42, 125]. Figure 4.2 illustrates the differences between a CPT and a decision tree. This thesis focuses on BNs with decision trees.

The quality of a given network structure is quantified by a scoring metric. There



(a) CPT (b) Decision tree

Figure 4.2: Example of a conditional probability table for $p(X_1|X_2X_3)$ using the (a) traditional representation and a (b) decision tree. The decision tree allows a more efficient and flexible representation of the conditional probabilities.

are two popular metrics for BNs: the Bayesian-Dirichlet metric (BD) [26, 74] and the Bayesian information criterion (BIC) [156]. The BD metric for BNs with decision trees [25] is given by

$$BD(B) = p(B) \prod_{i=1}^{\ell} \prod_{l \in L_i} \frac{\Gamma(m'_i(l))}{\Gamma(m_i(l) + m'_i(l))} \prod_{x_i} \frac{\Gamma(m_i(x_i, l) + m'_i(x_i, l))}{\Gamma(m'_i(x_i, l))}, \tag{4.2}$$

where $p(B)$ is the prior probability of the network structure B, L_i is the set of leaves in the decision tree T_i (corresponding to X_i), $m_i(l)$ is the number of instances in the population that contain the traversal path in T_i ending in leaf l, $m_i(x_i, l)$ is the number of instances in the population that have $X_i = x_i$ and contain the traversal path in T_i ending in leaf l, $m'_i(l)$ and $m'_i(x_i, l)$ represent prior knowledge about the values of $m_i(l)$ and $m_i(x_i, l)$. This work considers the K2 variant of the BD metric, which uses an uninformative prior that assigns $m'_i(x_i, l) = 1$.

To favor simpler networks over more complex ones, the prior probability of each network $p(B)$ can be adjusted according to its complexity, that is given by the description length of the parameters required by the network [25, 42]. Based on this

principle, the following penalty function was proposed for BOA [125],

$$p(B) = 2^{-0.5 \log_2(n) \sum_{i=1}^{\ell} |L_i|}, \tag{4.3}$$

where $|L_i|$ is the number of leaves in decision tree T_i.

The BIC metric is based on the minimum description length (MDL) principle [140] and is given by

$$BIC(B) = \sum_{i=1}^{\ell} \left(\sum_{l \in L_i} \sum_{x_i} \left(m_i(x_i, l) \log_2 \frac{m_i(x_i, l)}{m_i(l)} \right) - |L_i| \frac{\log_2(n)}{2} \right). \tag{4.4}$$

It has been shown that the behavior of these metrics is asymptotically equivalent; however, the results obtained with each metric can differ for particular domains, particularly in terms of sensitivity to noise. In the context of EDAs, when using CPTs to store the parameters, the BIC metric outperforms the K2 metric, but when using decision trees or graphs, the K2 metric has shown to be more robust [125]. This observation will be confirmed in the next chapter.

To learn the most adequate structure for the BN, a greedy algorithm is usually used for a good compromise between search efficiency and model quality. A simple learning algorithm starts with an empty network and at each step performs the operation that improves the metric the most, until no further improvement is possible. Figure 4.3 describes the learning algorithm in more detail.

When using DTs to store conditional probabilities, the greedy search does not directly manipulate the Bayesian network structure. Instead, the search operates over the set of decision trees $\{T_1, T_2, \ldots, T_\ell\}$. The search operator(s) modify the DTs and by consequence the BN structure is updated. The operator used is the *split*, which splits a leaf on some variable and creates two new children on that leaf. Each time a split on X_j takes place at tree T_i, an edge from X_j to X_i is added to

Bayesian Network Greedy Search

(1) Start with an empty Bayesian network with no edges `B` and ℓ decision trees `T(i)` with a single leaf (storing $p(X_i)$).

(2) Compute the score metric for the current network `(B,T)`.

(3) For all possible splits on every decision tree `T(i)`.

(3.1) Compute the score metric for the resulting model structure.

(4) If every split fails to improve the current score, finish; otherwise, perform the split that improves the score metric the most and does not introduce a cycle in `B`.

(5) Add the corresponding edge to `B`. Return to step (3).

Figure 4.3: Pseudocode of the greedy algorithm for learning a Bayesian network with decision trees.

the network. For more details on learning BNs with local structures the reader is referred elsewhere [25, 42, 125].

The generation of new solutions is done by sampling from the learned Bayesian network using probabilistic logic samping (PLS) [75]. Briefly, PLS consists in (1) computing an ancestral ordering of the nodes (where each node is preceded by its parents) and (2) generating the values for each variable according to the ancestral ordering and the conditional probabilities (Equation 4.1).

The hierarchical BOA (hBOA) was later introduced by Pelikan and Goldberg [126, 125] and results from the combination of BNs with local structures with a simple yet powerful niching method to maintain diversity in the population, known as restricted tournament replacement (RTR) [64]. hBOA is able to solve hierarchical decomposable problems, where variable interactions are present at more than a single level.

4.3 Modeling Fitness in BOA

Pelikan and Sastry [132] extended the Bayesian networks used in BOA to encode a surrogate fitness model that estimates the fitness for a proportion of the population, thereby reducing the total number of function evaluations. For each possible value x_i of every variable X_i, an estimate of the marginal fitness contribution of a subsolution with $X_i = x_i$ is stored for each instance π_i of X_i's parents Π_i. Therefore, in the binary case, each row of the CPT is extended by two additional entries. The fitness of an individual can then be estimated as

$$f_{est}(X_1, X_2, \ldots, X_\ell) = \bar{f} + \sum_{i=1}^{\ell} \bar{f}(X_i|\Pi_i), \tag{4.5}$$

where $\bar{f}$ is the average fitness of all solutions used to learn the surrogate and $\bar{f}(X_i|\Pi_i)$ is the conditional average fitness of solutions with X_i. Note that

$$\bar{f}(X_i|\Pi_i) = \bar{f}(X_i, \Pi_i) - \bar{f}(\Pi_i), \tag{4.6}$$

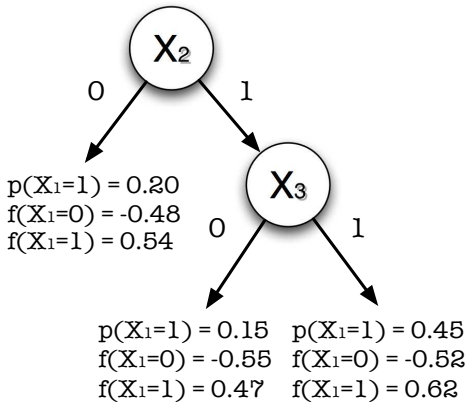
where $\bar{f}(X_i, \Pi_i)$ is the average fitness of solutions with X_i and Π_i, and $\bar{f}(\Pi_i)$ is the average fitness of all solutions with Π_i.

Fitness information can also be incorporated in Bayesian networks with decision trees in a similar way. In this case, the average fitness of each instance for every variable must be stored in every leaf of the decision tree. The fitness averages in each leaf are now restricted to solutions that satisfy the condition specified by the path from the root of the tree to the leaf. Figure 4.4 shows the previous example when storing fitness information as well.

The original proposal for fitness estimation in BOA uses the surrogate to evaluate the fitness of a proportion of the offspring population, which reduces the total

X_2X_3	$p(X_1=1\|X_2X_3)$	$f(X_1=0\|X_2X_3)$	$f(X_1=1\|X_2X_3)$
00	0.20	-0.49	0.53
01	0.20	-0.38	0.51
10	0.15	-0.55	0.47
11	0.45	-0.52	0.62

(a) Conditional probability table



(b) Decision tree

Figure 4.4: Example of a conditional probability table which also stores fitness information for $p(X_1|X_2X_3)$ using the (a) traditional representation and a (b) decision tree.

number of evaluations spent. The information necessary to compute statistics used for fitness estimation is acquired from (1) selected parents that were evaluated using the actual fitness function, and (2) offspring that were evaluated with the actual fitness function. The computation is restricted to these solutions because the current Bayesian network is directly related to this information (learned from the parents and sampled the offspring). Only evaluated solutions from the original fitness function are considered so that the noise coming from the errors in estimating fitness is not propagated through several iterations of the algorithm.

In this thesis the surrogate model will be exclusively used for evaluating the substructural local search steps, therefore the entire parent and offspring population is evaluated with the actual fitness function. By doing this, the population is kept

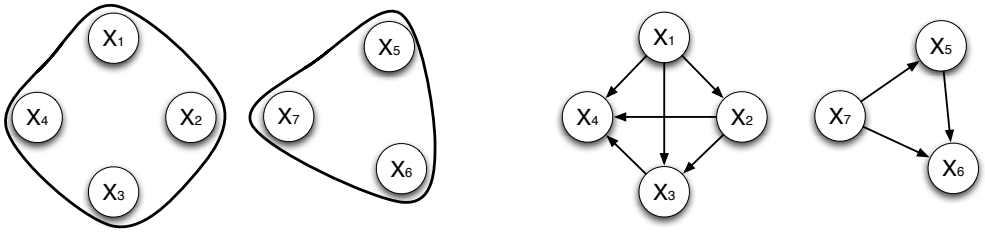
free of noise arising from possible surrogate errors, and performing local search becomes costless while rapidly achieving important subsolutions.

4.4 Substructural Neighborhoods in Bayesian Networks

Recently, it has been shown that a selectomutative algorithm that performs hill-climbing in the substructural space can successfully solve problems of bounded difficulty with subquadratic scalability, as opposed to $\Theta(\ell^k \log \ell)$ scalability of traditional local search methods [150, 115, 149]. The previous chapter showed that when incorporating substructural local search in eCGA, the resulting algorithm is more robust than both single-operator-based approaches [65, 149].

This section concerns about extending the concept of substructural neighborhoods to the case of Bayesian networks [103]. Marginal product models are fundamentally different from Bayesian networks and other graphical models that rely on conditional dependencies. While eCGA groups variables together in a joint distribution, BOA factorizes the problem by making use of conditional dependencies and independencies. The notion of *interaction* is of special relevance here. Two (or more) variables *interact* if they yield more information when considered together than separately. Interaction itself is that dependency that cannot be broken down [81].

Consider the example of a possible factorization shown in Figure 4.5. While eCGA is able to directly represent interactions between variables, BOA implicitly represent the interaction through a product of conditional dependencies. If k variables interact with each other, the Bayesian network should express their joint distribution to be able to maintain k-order statistics. As can be seen in Figure 4.5 (b),



(a) Marginal Product Model (b) Bayesian network

Figure 4.5: Factorization of the probability distribution using a (a) marginal product model and a (b) Bayesian network. While in the first case the factorization is directly expressed as $p(X_1X_2X_3X_4)p(X_5X_6X_7)$, with BNs the factorization is expressed as $p(X_1)p(X_2|X_1)p(X_3|X_1X_2)p(X_4|X_1X_2X_3)p(X_7)p(X_5|X_7)p(X_6|X_5X_7)$.

the joint distribution of variables X_1, X_2, X_3, and X_4 can be expressed in a Bayesian network as

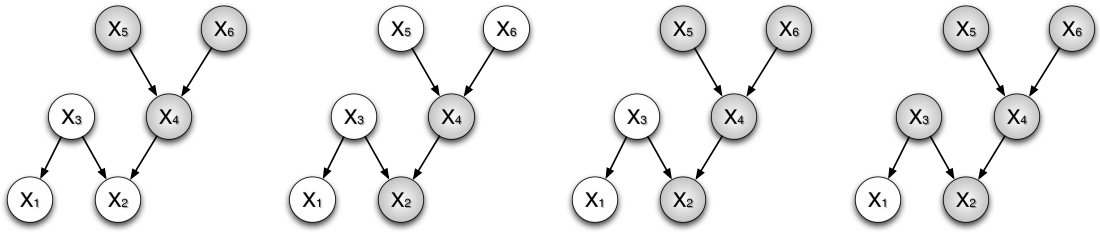
$$p(X_1X_2X_3X_4) = p(X_1)p(X_2|X_1)p(X_3|X_1X_2)p(X_4|X_1X_2X_3), \tag{4.7}$$

or any other permutation of the variables that respect this dependency structure. Ideally, the Bayesian network structure should contain a *clique*[1] between interacting variables, where the direction of the edges is defined in such way that there are no cycles (so that sampling new instances with PLS is feasible). Additionally, the model should not contain edges between non-interacting variables.

Given the structure of a Bayesian network, several neighborhood topologies can be considered to perform random or improvement-guided mutations. For a given variable X_i, the corresponding set of parent nodes Π_i, and set of child nodes Ω_i (nodes to where an edge arrives from node X_i), four different substructural neighborhoods can be defined:

Parental neighborhood considers variable X_i together with parent variables Π_i.

[1] A *clique* is a set of nodes N such that for every two nodes in N, there exist an edge (regardless its direction) connecting them.



(a) Parental (b) Children (c) Parental+Children (d) Markov blanket

Figure 4.6: Topology of the (a) parental, (b) children, (c) parental+children, and (d) Markov blanket neighborhoods for variable X_4.

This neighborhood is therefore defined by $K = 1 + |\Pi_i|$ different variables, resulting in 2^K possible values in the binary realm.

Children neighborhood considers variable X_i together with child variables Ω_i. Thus this neighborhood is defined by $K = 1 + |\Omega_i|$ variables.

Parental+Children neighborhood considers variable X_i together with both parent variables Π_i and child variables Ω_i. This neighborhood is composed by $K = 1 + |\Pi_i| + |\Omega_i|$ variables.

Markov blanket neighborhood same as above, but in addition the remaining parents of the child variables Ω_i are considered as well. This neighborhood is composed by $K = |\Pi_i| + |\Omega_i| + |\Pi_{\Omega_i}| - |\Pi_i \cap \Pi_{\Omega_i}|$ variables.

Figure 4.6 shows the topology of the different neighborhoods for a particular variable. When considering the substructural neighborhood of different variables, these four topologies explore the structure captured by the Bayesian network to different extents. However, looking at the dependency structure exemplified in Figure 4.5 (b), the neighborhoods defined for every variable inside an interaction group are exactly the same for both parental+child and Markov blanket neighborhoods. Take for example the interaction $\{X_5, X_6, X_7\}$. For every of these variables, both

neighborhoods are exactly the same. This leads to a certain redundancy when exploring the neighborhoods for each variable X_i. Therefore, the parental or child neighborhoods are more efficient choices.

The parental neighborhood however presents a clear advantage over the child neighborhood. Because the conditional probabilities and corresponding fitness contribution are already stored in the form $p(X_i|\Pi_i)$ and $\bar{f}(X_i|\Pi_i)$, the information needed to perform SLS is directly accessible when using the parental neighborhood. Thus, this work focuses on the parental neighborhood to guide substructural local search.

A somewhat related approach has been recently proposed by Handa [60], where the traditional bitwise mutation operator is employed in the estimation of Bayesian networks algorithm (EBNA) [37]. Consequently, variables that depend on the mutated node are resampled according to the conditional probabilities for the new instance. Although this mutation operator takes into account the dependencies between variables, it is specifically designed to perturb solutions in order to maintain diversity in the population. The approach proposed in this chapter is to interpret the structure of the Bayesian network as a set of linkage groups to define neighborhoods explored by local search.

4.5 BOA with Substructural Local Search

This section integrates a local search method in BOA that uses the parental neighborhood proposed in the previous section [103]. The substructural local search is performed for a proportion of the population to speedup the convergence to good solutions, as in traditional hybrid GAs. After the offspring population is sampled from the probabilistic model and evaluated, each individual is submitted to substructural

BOA with Substructural Local Search (BOA+SLS)

(1) Create a random population `P` of n individuals.

(2) Evaluate population `P`.

(3) Select `P'` individuals from `P` using a selection procedure.

(4) Model the selected individuals `P'` by learning the most adequate Bayesian network `B`.

(5) Create a new population `O` by sampling from the joint probability distribution of `B`.

(6) Evaluate population `O`.

(7) Submit population `O` to substructural local search with probability p_{ls}.

(8) Replace all (or some) individuals in population `P` by those from `O`.

(9) If stopping criteria are not satisfied, return to step 3.

Figure 4.7: Pseudocode of the Bayesian optimization algorithm with substructural local search (BOA+SLS).

hillclimbing with probability p_{ls}. Figure 4.7 shows an algorithmic description of the resulting algorithm (BOA+SLS).

While in eCGA substructural local search was only applied to the best individual, in BOA it is more efficient to perform local search for several individuals. The first reason for doing so is related to the more complex nature of probabilistic modeling with Bayesian networks, where increasing model complexity can sometimes lead to model overfitting. In addition, the decision making between competing substructures is certainly not error free, due to the noise coming from estimated fitness. To overcome these possibilities, experiments for several proportions of local search are considered. Another important reason is that SLS uses surrogate fitness which substantially decreases the cost of local search in terms of number of function evaluations. The substructural local searcher is described in Figure 4.8.

Substructural Local Search (SLS)

(1) Consider the first variable X_i according with the reverse ancestral ordering of variables in the Bayesian network.

(2) Choose the values (x_i, π_i) associated with the maximal substructural fitness $\bar{f}(X_i, \Pi_i)$.

(3) Set variables (X_i, Π_i) of the considered individual to values (x_i, π_i) if the overall fitness of the individual is improved by doing so, otherwise leave the individual unchanged.

(4) Repeat steps 2-3 for all remaining variables following the reverse ancestral order of variables.

(5) Evaluate the resulting individual.

Figure 4.8: Pseudocode of the substructural local search in BOA.

Some details need further explanation. First, a reverse order of that used to sample the variables of new solutions is used, where each node is preceded by its parents. By doing so, higher-order dependencies within the same linkage/interaction group are *optimized* first. This procedure aims to reduce the possibility of doing incorrect decisions when considering problems whose lower-order statistics can be misleading. Later on, the removal of lower-order dependencies is also considered.

Two different versions of the SLS are tested with experiments, that only differ in step 3. The first version uses the estimated fitness of the individual (Equation 4.5) to decide if the best substructure (according to $\bar{f}(X_i, \Pi_i)$) for a given neighborhood should be accepted, while the second version uses the actual fitness function to make the decision. After performing substructural hillclimbing for all variables, the resulting individual is evaluated with the fitness function before being inserted back into the population. This avoids the propagation of error possibly introduced by using surrogate fitness. Thus, the surrogate is only used to perform local search in the substructural neighborhoods. That said, the additional cost in terms of

evaluations when performing local search can be simply estimated as $n \cdot p_{ls}$ for the first version of the local searcher and $n \cdot p_{ls} \cdot \ell$ for the second version.

It should be noted that searching within the same substructural neighborhoods for different individuals yields results whose similarity increase with the accuracy of the probabilistic model. However, in practice, performing local search on different individuals helps to overcome incorrect biases from possible errors in the learned models.

4.6 Experiments

This section presents the results obtained for varying proportions of local search p_{ls} and empirically analyzes the scalability of the proposed method for increasing problem size. Two different problems are initially used to test substructural local search in BOA: onemax and m-k trap functions. These problems represent two important bounds on a class of additively decomposable problems with bounded difficulty.

Onemax is a simple linear function for which there is no need of linkage learning to be able to solve it efficiently. While the optimization of the onemax problem is easy, the probabilistic models built by EDAs such as eCGA and BOA, however, are known to be only partially correct and include spurious dependencies/interactions. Therefore, the results on this function will indicate if the effect of using partially correct linkage mapping on the accuracy of the surrogate is significant, and consequently, if performing substructural local search under these conditions is still advantageous. A onemax function with size $\ell = 50$ is considered. The second function is m-k trap problem with $m = 10$ and $k = 5$, therefore with $\ell = 50$ as well.

For each problem, experiments are performed for different proportions of local

search p_{ls}, between 0 and 0.2. The minimal number of function evaluations required to obtain the optimal solution is empirically determined using the bisection method over the population size [145, 125]. For each experiment, 10 independent bisection runs are performed. Each bisection run searches for the minimal population size required to find the optimum in 10 out of 10 independent runs. The results for the minimal sufficient population size are therefore averaged over 10 bisection runs, while the results for the number of function evaluations and the number of generations spent are averaged over 100 (10×10) independent runs. For all experiments, binary tournament selection without replacement is used.

4.6.1 Results

The results for onemax and trap problems are shown in figures 4.9 and 4.10. For both problems, the number of evaluations is substantially reduced when using local search that explores substructural neighborhoods. Also, both versions of the substructural hillclimber succeed to reduce the cost of solving the problem. Nevertheless, different dynamics can be observed for each problem.

For onemax, using the actual fitness function for deciding between competing substructures provides slightly better results than using estimated fitness, while the population size required is substantially smaller, in particular for higher proportions of local search. Note that the correctness of the substructural neighborhoods is not crucial when solving onemax using local search because there is no linkage. However, the choice of the best alternative in each neighborhood is based on the substructural fitness contribution that is estimated by the surrogate whose correctness relies on the accuracy of the probabilistic model. But even more important is the acceptance (or not) of the substructures. By using real fitness evaluation in this decision, only those partial solutions that really improve the fitness of the individual are accepted,

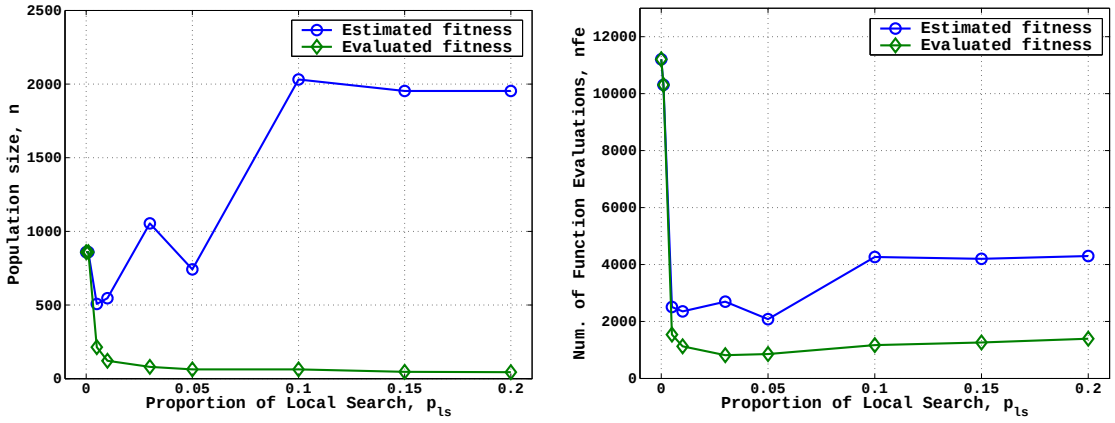


Figure 4.9: Population size and number of function evaluations required to solve the 50-bit onemax problem for different proportions of local search.

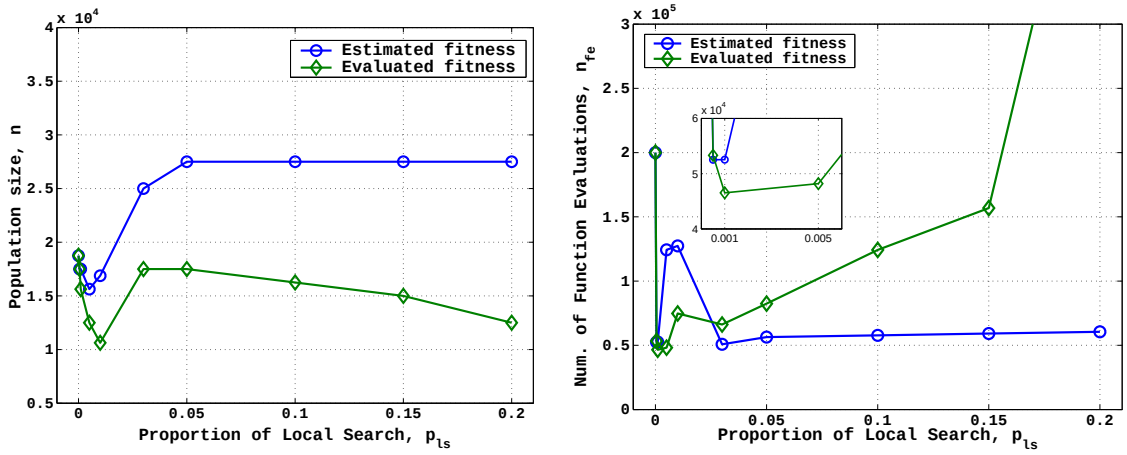


Figure 4.10: Population size and number of function evaluations required to solve the 10x5-bit trap problem for different proportions of local search.

which drastically reduces the need of having an accurate surrogate fitness model (and consequently larger population size). For the local searcher that uses only estimated fitness, the population size required grows even more for higher proportions of local search because the diversity in the population is quickly reduced, which requires the surrogate to be accurate enough to solve the problem in the first generations.

For the trap problem, the correct identification of substructures is essential to solve the problem, which requires the accuracy of the Bayesian network to be higher. Therefore, both hillclimbers perform similar for small proportions of local search.

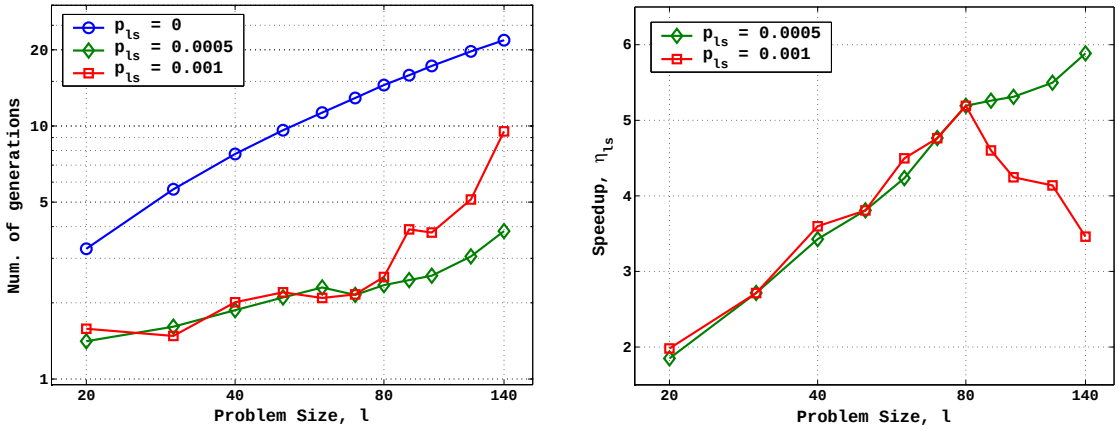


Figure 4.11: Number of generations required to get the optimum and the speedup obtained by performing SLS for the trap-5 problem. For $p_{ls} = 0.0005$ the speedup scales approximately as $\Theta(\ell^{0.45})$, although for larger problem instances the speedup is more moderate. For higher proportions of local search the speedup seems to decrease due to excessive reduction of diversity in the population.

In this case, however, the cost of using fitness function calls at each step of the substructural hillclimber shows to be an expensive overhead for higher values of p_{ls}. Similar to onemax, there is a transition phase in the population size required for the local searcher that uses surrogate fitness. For $p_{ls} \geq 0.05$, the population size stagnates at a value where the model is accurate enough to solve the problem in the first generation by performing SLS.

Figure 4.11 presents the results obtained for increasing instances of the trap problem ($k = 5$ and varying m) with BOA+SLS using estimated fitness. The number of generations required to reach the optimum and the speedup obtained by local search are shown. Note that the speedup is simply the ratio of the number of evaluations required by BOA with and without local search. The population size required (not plotted) scales similarly for all tested p_{ls} values and according with population sizing theory [128, 134].

The results show that while obtaining a significant reduction in the number of generations, substantial speedups are provided by using substructural local search

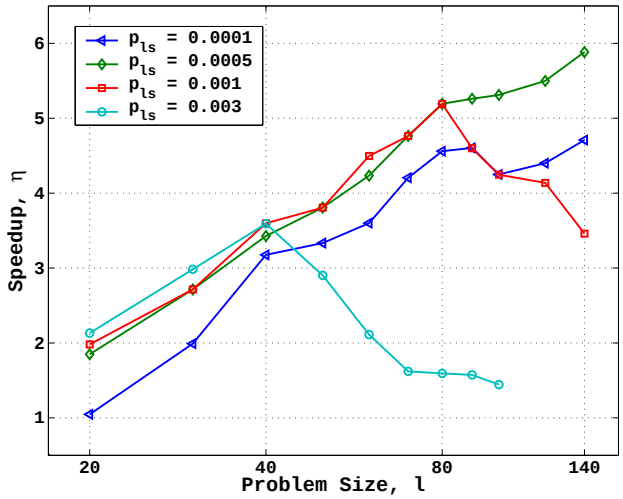


Figure 4.12: Speedup obtained for different proportions of local search in BOA+SLS. For higher proportions of local search the speedup decreases after some problem size ℓ which depends on p_{ls}. Smaller proportions of local search lead to a curve with similar slope to $p_{ls} = 0.0005$ but with inferior speedups.

in BOA. The speedup grows approximately as $\Theta(\ell^{0.45})$, however for larger problem instances the increase in speedup becomes more moderate for $p_{ls} = 0.0005$, while for higher proportions of local search the speedup even decreases. This seems to be due to the population size required for larger problems, increasing the number of individuals that undergo local search for the same value of p_{ls}, and thereby reducing diversity in the population. Note that the resulting individuals from substructural local search are very similar. On the other hand, smaller proportions of local search lead to a curve with similar slope to that obtained for the adequate proportion but with inferior speedups. This can be observed in Figure 4.12.

4.7 Discussion

While using (the right amount of) substructural local search in BOA is shown to speedup the convergence to good solutions, the scalability results demonstrate that the corresponding speedup becomes more moderate for larger problem instances.

The reduction of the slope in the speedup curve, even for adequate values of p_{ls}, seem to be related to problem-structure learning and interpretation with BNs. Two aspects deserve further investigation.

The first point to consider is the exploration of substructural neighborhoods using a reverse ancestral ordering of variables (where each node is preceded by its child). For example, consider that variables X_1, X_2, X_3, X_4, and X_5 interact with each other. Assuming that the joint distribution of these variables is represented by the typical product of conditional dependencies (see Equation 4.7), the resulting substructural neighborhoods will be explored in the following order:

1. $\{X_1, X_2, X_3, X_4, X_5\}$
2. $\{X_1, X_2, X_3, X_4\}$
3. $\{X_1, X_2, X_3\}$
4. $\{X_1, X_2\}$
5. $\{X_1\}$

As mentioned before, this order allows higher-order dependencies within the same linkage/interaction group to be optimized first. While this procedure reduces the chance of doing incorrect decisions (in particular for problems whose lower-order statistics may mislead the search), it does not avoid it completely. An efficient solution for this problem is to skip substructural neighborhoods that are *subsets* of previously visited neighborhoods. By proceeding in this way only the first neighborhood would be visited in the previous example. Note that the fitness information stored in the model already contains fifth-order statistics for the term $\bar{f}(X_1, X_2, X_3, X_4, X_5)$, therefore all the required information to make a correct decision is present. While this is valid for evaluating a substructure[2], to estimate the overall fitness of a solution all the terms are still required (as in Equation 4.5).

[2]Notice that when comparing competing subsolutions, it is not relevant if the corresponding

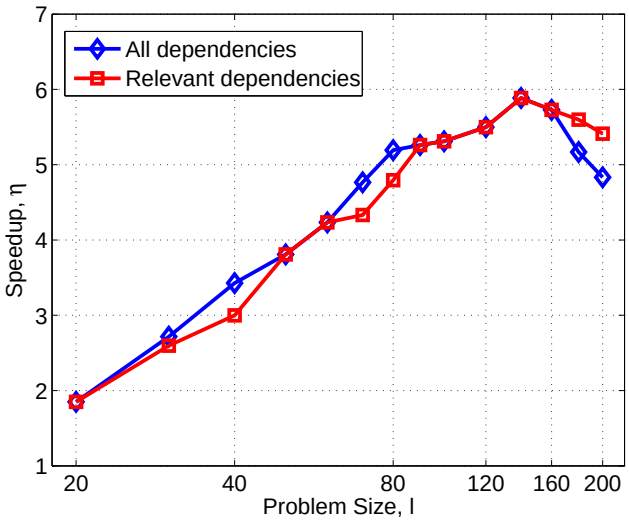
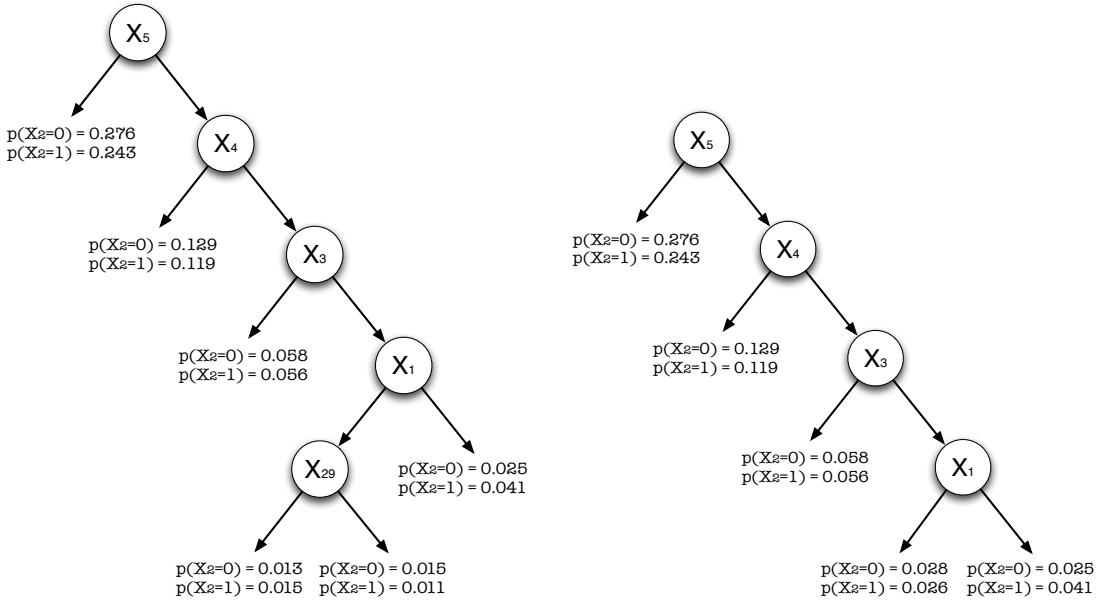


Figure 4.13: Speedup obtained for BOA+SLS when exploring all dependency groups or only relevant ones. Both versions of SLS fail to maintain the speedup for larger instances.

Figure 4.13 presents the speedup obtained when using only relevant dependencies compared to exploring all ℓ neighborhoods. Note that now instances up to $\ell = 200$ are tested. Both versions of SLS fail to maintain the speedup for larger instances. Although the speedup seems to decrease slowly when using only relevant dependencies, for $\ell > 140$, both alternatives reduce their efficiency.

The second aspect to investigate is the structural accuracy of the probabilistic models. Analyzing the dependency groups learned by the Bayesian network, it can be observed that the number and size of spurious linkages increases with problem size. Spurious linkage can be defined as additional variables that are considered together with a correct linkage/interaction group. Consequently, local search deals with substructural neighborhoods of excessive size and complexity, that do not entirely correspond to the underlying structure of the problem. Although the structure of the Bayesian network captures these spurious dependencies, the conditional proba-

estimated fitness is normalized or not.



(a) Spurious linkage (b) Correct linkage

Figure 4.14: Example of (a) spurious and (b) correct linkage identification with decision tree learning for a trap-5 subfunction (located at $\{X_1,X_2,X_3,X_4,X_5\}$). The decision tree encodes the parameters for X_2. Instead of conditional probabilities, joint probabilities are presented to demonstrate the low significance of the spurious variable X_{29}. The addition of less significant dependencies depends on the scoring-metric sensitivity to noise.

bilities nearly express independency between spurious and linked variables, therefore not affecting the capability of sampling such variables as if they were independent.

Figure 4.14 shows an example of spurious linkage identification with decision tree learning. The decision tree represents the parameters corresponding to variable X_2, which is known to be correlated with X_1, X_3, X_4, and X_5 (learned structure of a 5-bit subfunction). Instead of conditional probabilities, joint probabilities are presented to demonstrate the low significance of the spurious variable X_{29}. Notice that the correlation detected for X_{29} is the least significant among all variables in the decision tree. Because the score metric accepts every split whose corresponding gain is greater than zero, additionally spurious variables can later be added[3] to

[3]In the last iterations of BN learning.

the already correct linkage group. The addition (or not) of these less significant dependencies greatly depends on the sensitivity of the scoring metric to noise.

In eCGA, while model overfitting can also be an issue, this is usually not the case for problems like the (non-overlapping) m-k trap, because MPMs have an "adequate" complexity for representing such problem structure. Notice that MPMs group variables into non-overlapping clusters. Therefore, assigning a spurious variable to a particular cluster, requires the corresponding metric value to be better than having that variable assigned to the cluster where it actually belongs. On the other hand, and because BNs allow the representation of overlapping interactions, BOA can assign a spurious variable to some interacting group of variables and still have that variable interacting with highly-correlated variables. In this case, the only condition is that the corresponding metric gain is greater than zero, which can often happen due to stochastic noise.

Therefore, to efficiently tackle this issue the accuracy of Bayesian networks in BOA has to be addressed to understand in which conditions the model quality is inferior, and how it can be improved. This is the topic of the next chapter.

4.8 Summary

This chapter introduces substructural local search in BOA—an EDA which uses Bayesian networks to model promising solutions and sampling new ones. The concept of substructural neighborhood is extended from eCGA to BOA, which uses more complex models than MPMs. For that, four different substructural neighborhoods are considered, being the parental neighborhood identified as the most suitable one. In addition, a surrogate fitness model is used to evaluate competing substructures while performing local search, so that the corresponding cost in terms

of function evaluations is substantially reduced. The results show that incorporating substructural local search in BOA can lead to a significant reduction in the number of generations necessary to solve the problem, providing relevant speedups in terms of number of evaluations. However, for larger instances of the trap problem the efficiency decreases, which suggests further analysis to identify the reason behind such behavior. While a more efficient exploration of the substructural neighborhoods fails to improve the results for larger instances, the root of the problem is associated with the structural accuracy of the probabilistic models in BOA. The next chapter addresses this issue in detail.

Chapter 5

Model Accuracy in BOA

5.1 Introduction

Estimation of distribution algorithms are typically classified according to the complexity of the probabilistic models they rely on. Simpler EDAs use a model of simple and fixed structure, and only learn the corresponding parameters. At the other side of the spectrum, are the Bayesian EDAs which use Bayesian networks to model complex multivariate interactions.

While Bayesian EDAs are able to solve a broad class of nearly decomposable and hierarchical problems in a reliable and scalable manner, their probabilistic models oftentimes do not exactly reflect the problem structure [96, 71, 35, 116, 101, 98, 99]. Because these models are learned from a sample of limited size (population of individuals), particular features of the specific sample are also encoded, which act as noise when seeking for generalization. This is a well-known problem in machine learning, known as *model overfitting*. However, in the context of EDAs model overfitting is double-sided. While the goal is to model promising solutions rather than the entire search space, focusing on an excessively narrowed portion of this space

might not reveal meaningful information about the underlying problem structure, and even reduce the probability of finding the optimum.

In many situations, the knowledge of the problem structure can be as valuable as a high-quality solution to the problem. This is the case for several model-based efficiency enhancement techniques [154, 132, 153, 149, 103, 104, 146, 176, 72, 69, 97] developed for EDAs that yield super-multiplicative speedups. Another important situation is the *offline interpretation* of the probabilistic models [173, 174] to help develop fixed but structure-based operators for specific instances or classes of problems that have similar structure. In this case the EDA can act as a *data miner* to gain insight about the problem. The importance of analyzing the resulting probabilistic models in EDAs has also been recently highlighted by others [144, 171, 27, 35, 71, 96, 116].

Bayesian network learning is an active topic of research in machine learning, as the choice of the search procedure can have a great influence on model accuracy. However, the problem of finding the best network has been proven to be NP-complete for most scoring metrics [24]. Therefore, in Bayesian EDAs a simple local search procedure is typically used for a good compromise between search efficiency and model quality [127, 125, 37, 119], given the high computational cost of considering more sophisticated alternatives [35]. Note that the Bayesian networks (or any other probabilistic model for that matter) are used as an auxiliary tool in the optimization process, thus it is a good practice to keep the search complexity as simple as possible. On the other hand, this work focuses on the best way to integrate BNs within the evolutionary computation framework to improve the expressiveness of learned models.

This chapter investigates model structural accuracy in the Bayesian optimization algorithm [98, 99, 101, 96], giving particular emphasis to the relationship between

the underlying problem structure and the learned Bayesian network structure. The chapter also addresses the selection operator as a source of overfitting in Bayesian EDAs. First, a detailed analysis of model learning in BOA is performed to better understand how the problem structure is learned and when inaccuracies are introduced in the network. Next, the role of selection in BN learning is investigated by looking at selection as the mating pool distribution generator, which turns out to have a great impact on model structural accuracy. Particularly, it is shown that tournament selection generates the mating pool according to a power distribution that leads to model overfitting. However, if the metric that scores networks takes into account the resampling bias induced by tournament selection, the model quality can be highly improved and comparable to that of truncation selection which generates a uniform distribution, more suitable for BN learning.

The chapter is organized as follows. The next section presents a brief survey on previous work related to model accuracy in EDAs. Section 5.3 analyzes in detail how the model is learned in BOA, while Section 5.4 investigates the role of selection in model learning and overfitting. Section 5.5 models the scoring metric gain when overfitting with tournament selection. In Section 5.6, an adaptive scoring metric is proposed to avoid overfitting, which is shown to considerably improve model accuracy. The chapter ends with a summary.

5.2 Related Work

Although the main feature of BOA and other EDAs is to perform efficient mixing of key substructures or BBs, they also provide additional information about the problem being solved. The probabilistic model of the population, that represents (in)dependencies among decision variables, is an important source of information

that can be exploited to enhance the performance of EDAs even more, or to assist the user in a better interpretation and understanding of the underlying structure of the problem. Examples of using structural information from the probabilistic model for another purpose beyond mixing include the design of structure-aware crossover operators [173], fitness estimation [154, 132, 153], induction of global neighborhoods for mutation operators [149, 103, 60, 135], hybridization and adaptive time continuation [104, 103], substructural niching [146, 97], offline [174] and online [176] population size adaptation, and speeding up the model building itself [72, 69]. Therefore, it is important to understand in which conditions the structural accuracy of the probabilistic models in BOA and other structure-learning EDAs can be maximized. Recently, some studies have been done in this direction [144, 171, 27, 35, 71, 116]. The remainder of this section takes a brief look at these works.

Santana, Larrañaga, and Lozano [144] analyzed the effect of selection on bivariate interactions between single variables for random functions. They showed that for these functions, independence relationships not represented by the function structure are likely to appear in the probabilistic model. The authors also noted that even if the function structure plays an important role in the creation of dependencies, this role is mediated by selection [144]. Additionally, an EDA that only used a subset of the dependencies that exist in the data (malign interactions) was proposed. Some preliminary experiments showed that these approximations of the probabilistic model can in certain cases be applied to EDAs.

Wu and Shapiro [171] investigated the presence of overfitting when learning the probabilistic models in BOA and its consequences in terms of overall performance for solving random 3-SAT problems. CPTs (to encode the conditional probabilities) and the corresponding BIC metric were used. The authors concluded that overfitting does take place and that there is some correlation between this phenomenon and

performance. The reduction in overfitting was proposed by using an early stopping criterion (based on cross entropy) for the learning process of BNs, which gave some improvement in performance.

The tradeoff between model complexity and performance in BOA was also studied by Correa and Shapiro [27]. They looked at the performance achieved by BOA as a function of a parameter that determines the maximum number of incoming edges for each node. This parameter puts a limit on the number of parents for each variable, simplifying the search procedure for a model structure. This parameter was found to have a strong effect on the performance of the algorithm, for which there is a limited set of values where the performance can be maximized. These results were obtained using CPTs and the corresponding K2 metric. It should be noted that in fact this parameter is crucial if CPTs are used with the K2 metric, however this is not the case for more sophisticated metrics that efficiently incorporate a complexity term to introduce pressure toward simpler models. This can be done better with the BIC metric for CPTs, or with the K2 metric for the case of decision trees [125].

Echegoyen *et al.* [35] applied new developments in exact BN learning into the EDA framework to analyze the consequent gains in optimization. While in terms of convergence time the gain was marginal, the models learned by EBNA were more closely related to the underlying structure of the problem. However, the computational cost of learning exact BNs is only manageable for relatively small problem instances (experiments were made for a maximum problem size of 20).

Hauschild *et al.* [71] made an empirical analysis of the probabilistic models built by hierarchical BOA for several test problems. The authors verified that the models learned closely correspond to the problem structure and do not change much over consequent iterations. They have also concluded that creating adequate probabilistic models for the 2D Ising spin glasses problem by hand is not straightforward, even

with complete knowledge of the problem. While in that work Hauschild *et al.* used truncation selection, this chapter demonstrates that the results from [71] do not carry over to other selection methods that assign several copies of the same individual to the mating pool according to a non-uniform distribution.

Recently, Muhlenbein [116] investigated the Bayesian networks learned for LFDA and BOA when solving a trap-5 decomposable function. He found that in order to find the optimum about $80 - 90\%$ of the edges have to be correctly identified. Also, the penalty factor used for the BIC metric was shown to have influence on the network density. Although these results are relevant to better understand Bayesian EDAs, there is a fundamental difference from the study presented in this chapter—the maximum number of incoming edges was set according to the problem structure—which reduces dramatically the overfitting phenomenon. In real-world optimization this is not typically the case, therefore the task of learning the adequate complexity is left to the algorithm itself. Another important difference is that both LFDA and BOA used CPTs to encode the model parameters, as opposed to decision trees used in this thesis. Additionally, while LFDA used truncation selection, BOA was paired with tournament selection, resulting in worse model quality when compared to LFDA [116]. The author however did not make any remarks about the reason of such quality difference.

On the contrary, this work demonstrates that the difference in model quality is actually related with the selection procedure rather than the algorithm as a whole. Furthermore, it is shown when and why overfitting is related to the selection operator and a method to counterbalance this feature is proposed. The results presented in this chapter clarifies previous empirical comparisons between different Bayesian EDAs, which are known to mainly differ on the choice of the selection operator and scoring metric.

5.3 Analyzing Model Expressiveness

This section analyzes model learning in BOA when solving a problem of known structure and where that knowledge is crucial to solve it efficiently. It starts by introducing the experimental setup used along the chapter and then proceeds to a detailed analysis of the learning process of BNs in BOA.

5.3.1 Experimental Setup for Measuring Structural Accuracy of Probabilistic Models

At this point, it is adequate to clarify some terms that are relevant to the scope of this chapter.

Definition 5.3.1 *The* model structural accuracy *(MSA) is defined as the ratio of correct edges over the total number of edges in the Bayesian network.*

Definition 5.3.2 *An* edge *is* correct *if it connects two variables that are linked according to the objective function definition.*

Definition 5.3.3 Model overfitting *is defined as the inclusion of* incorrect (or unnecessary) edges *to the Bayesian network, which leads to excessive complexity.*

To investigate the MSA in BOA, this chapter focuses on solving problems of known structure, where it is clear which dependencies must be discovered (for successful tractability) and which dependencies are unnecessary (reducing the interpretability of the models). The first problem considered is the $m-k$ trap function. If k variables interact with each other, the probabilistic model should express their joint distribution to be able to maintain k-order statistics. Figure 5.1 shows an ideal BN for the $m-k$ trap problem with $k=4$. As mentioned before, the ideal Bayesian

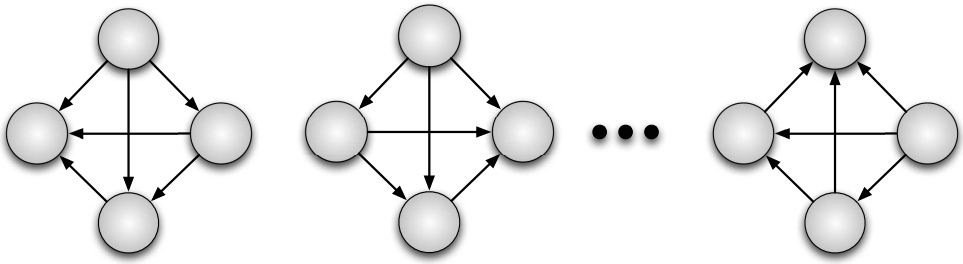


Figure 5.1: An ideal Bayesian network to solve the $m - k$ trap problem with $k = 4$. A clique is formed for each set of variables corresponding to a trap subfunction, while between different traps there are no dependencies.

network structure should contain a *clique* between interacting variables, where the order of the edges is defined in such way that there are no cycles. Additionally, the model should not contain dependencies between different trap subfunctions.

Later, in Section 5.6, the experiments are extended to onemax and hierarchical trap problems to validate the proposed approach for different types of variable interactions.

Like in the previous chapter, the minimal population size required to solve the problem in 10 out of 10 independent runs (success rate of 100%) is used. The population size is obtained by performing 10 independent bisection runs [145, 125]. Therefore, the total number of function evaluations is averaged over 100 (10×10) runs. To focus on the influence of selection in model quality, the replacement strategy is kept as simple as possible, where the offspring fully replace the parent population.

5.3.2 A Closer Look at Model Learning

When analyzing the dependencies captured by the Bayesian networks in BOA, it can be observed that while all important linkages are detected—given a sufficient population size [134, 125, 177]—spurious dependencies are also incorporated in the model. Remember that spurious dependencies are those incorrect (or unnecessary)

edges added to the network (according to the problem definition), which lead to model overfitting. On the other hand, the corresponding conditional probabilities oftentimes nearly express independence between spurious variables and correct linkage groups, therefore not affecting the capability of sampling such variables as if they were almost independent. Nevertheless, as model overfitting increases with problem size and selection pressure [96], the interpretation of the resulting models becomes meaningless.

To better understand the excessive complexity of the learned models in BOA, a detailed analysis is made to the model learning process. Figures 5.2 and 5.3 show the scoring metric gain obtained in model building for a typical run of BOA with K2 and BIC metrics, respectively. For each learning step (edge addition), the corresponding gain in the scoring metric is plotted, as well as if the edge inserted is correct (upper dots) or spurious (lower dots). The first, middle, and last generations of the run are plotted using binary tournament selection. Clearly, the K2 metric produces more accurate models than the BIC metric, when using BNs with local structures. Looking at the results, one can easily conclude that the K2 metric is better for learning the underlying problem structure because it introduces much less spurious dependencies than the BIC metric. Nevertheless, some incorrect edges are also inserted in the network for the K2 metric, particularly at the end of the learning process.

The number of correct edges (n_{ce}) in a Bayesian network for the $m - k$ trap problem is given by

$$n_{ce} = \sum_{i=1}^{m} \frac{k_i(k_i-1)}{2}, \quad \text{for } k \geq 2, \tag{5.1}$$

where m is the number of subfunctions and k_i is the size (number of interacting variables) of the i^{th} subfunction. Therefore, for the trap problem with $m = 10$ and $k = 5$ there is a total of 100 correct edges. While for the K2 metric at most 20

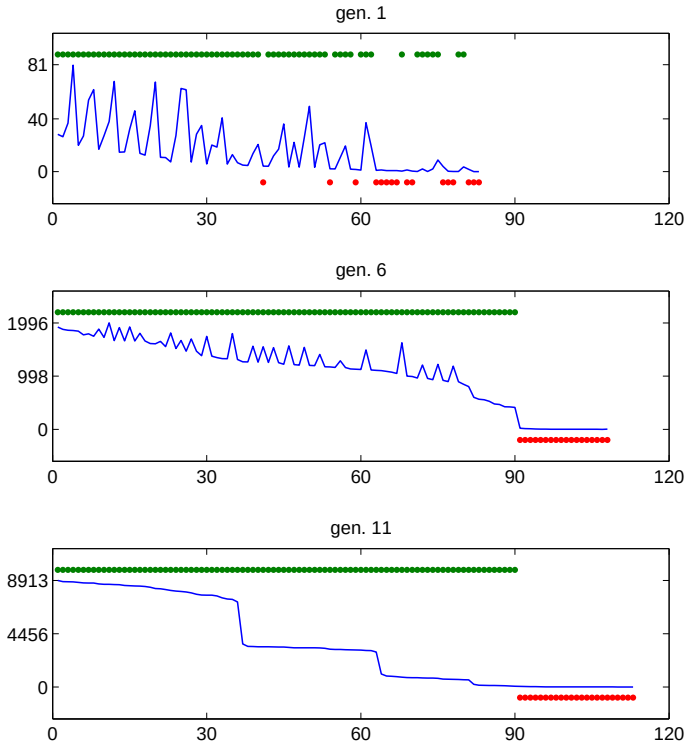


Figure 5.2: Metric gain vs. learning step in Bayesian network learning for a typical run with BOA using the K2 metric. The upper dots are correct edge additions while the lower ones are incorrect edges. The problem is the trap-5 with $m = 10$ ($\ell = 50$), for which there is a total of 100 correct edges. The first, middle, and last generations of the run are plotted.

incorrect edges are inserted in the network, for the BIC metric a maximum of 350 incorrect edges (generation 6) are learned! Even so, BOA with the BIC metric is able to solve the problem, although the model building phase takes more time. For the K2 metric, it can also be seen that while only 90% of the correct edges are learned, the problem can still be solved (recall however that this refers to a single run). This result agrees with the observations made elsewhere [116].

Analyzing BOA with the more robust K2 metric, it can be seen that the metric gain is higher at the beginning of the learning process and decreases towards zero—which is the threshold for accepting a modification in the Bayesian network. Additionally, the metric gain magnitude increases towards the end of the run (note

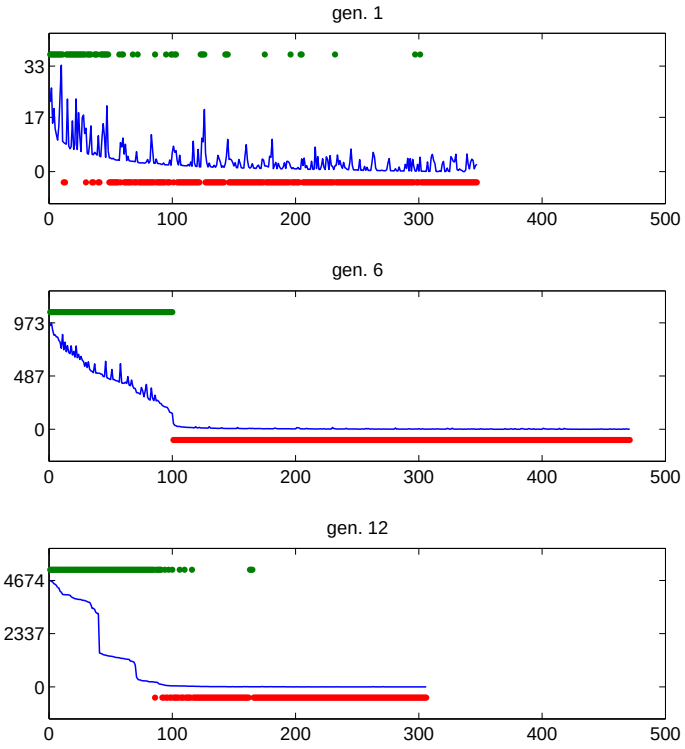
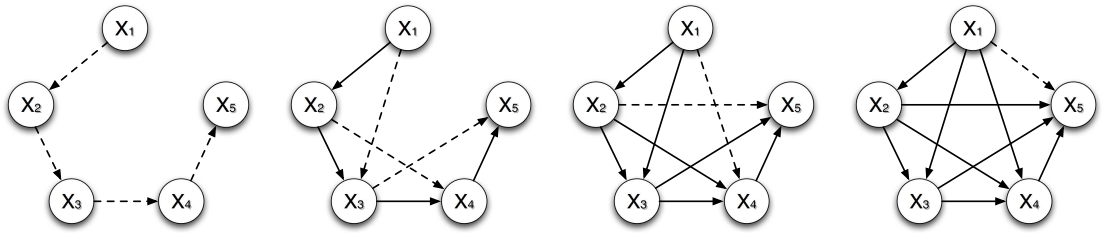


Figure 5.3: Metric gain vs. learning step in Bayesian network learning for a typical run with BOA using the BIC metric. The upper dots are correct edge additions while the lower ones are incorrect edges. The problem is the trap-5 with $m = 10$ ($\ell = 50$), for which there is a total of 100 correct edges. The first, middle, and last generations of the run are plotted.

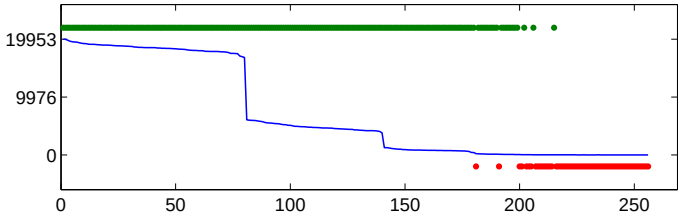
that the maximum metric gain at generation 1 is 81, while at generation 11 is 8913). This is due to the fact that as the search focuses on specific regions of the search space (loss in diversity) the marginal likelihood of the model increases. Indeed, for the last generation the shape of the metric gain function is less noisy when compared to the first and middle generations. With respect to the correctness of the edges, it is clear that the overwhelming majority of the spurious edges are inserted in the network at the end of the learning procedure. This suggests that an earlier stopping criterion or a higher acceptance threshold (note that spurious edges have a metric gain quite small) in the learning procedure could avoid the acceptance of a large part of incorrect edges.



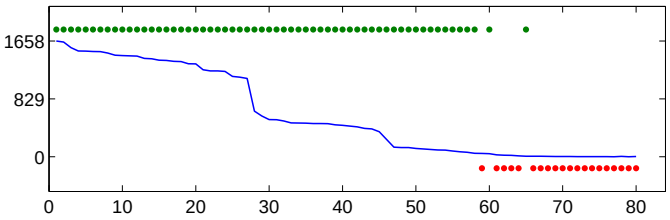
(a) Stage 1 (b) Stage 2 (c) Stage 3 (d) Stage 4

Figure 5.4: Different stages of Bayesian network learning in BOA when solving a trap function with $k = 5$. The order in which dependencies are learned is determined by their corresponding metric gain. Remember that the greedy algorithm for learning the network structure accepts at each step the edge that improves the scoring metric the most. Also note that the metric gain corresponding to adding an edge from a node X_a to another node X_b is inversely proportional to the number of parent nodes that X_b already contains [2]. Therefore, the magnitude of the metric gain for the $k-1$ different stages will differ. Typically, incorrect edges are added to the network at the later stages of learning, when the metric gain is smaller.

Before discussing the role of selection in model quality, it is relevant to look at the particular shape of the metric gain function observed for the last generation in Figure 5.2 (more clear for the K2 metric). This function appears to have the shape of several decreasing steps, where the metric gain drops considerably at particular points in BN learning. These are indeed different stages of dependency learning. Figure 5.4 shows an example with the different stages in Bayesian network learning when solving a trap-5 function. Four different stages can be identified, where the order in which edges are inserted into the network is determined by their corresponding metric gain. Remember that the greedy algorithm for learning the network structure accepts at each step the edge that improves the scoring metric the most. Note also that the metric gain corresponding to adding an edge from a given node X_a to another node X_b is inversely proportional to the number of parent nodes that X_b already contains [2]. This can also be concluded from the population size requirements for adding a correct edge in BOA [134, 125], which scales as $O(2^{\beta}\ell^{1.05})$, where β is the number of parent nodes that X_b already contains. Therefore, the



(a) $k = 5$ and $m = 20$



(b) $k = 4$ and $m = 10$

Figure 5.5: Metric gain vs. learning step in Bayesian network learning for a typical run with BOA using the K2 metric. The last generation is plotted for a trap problem with (a) $k = 5, m = 20$ and (b) $k = 4, m = 10$, which have a total of 200 and 60 correct edges, respectively. The decrease in metric gain closely match the different stages of BN learning.

magnitude of the metric gain for the $k - 1$ different stages will decrease towards later stages.

Looking at the metric gain function for generation 11 with the K2 metric (see Figure 5.2), this conjecture can be confirmed. For $m = 10$ trap functions of $k = 5$, there should be $10 \times 4 = 40$ edges for the first stage, $10 \times 3 = 30$ edges for the second stage, $10 \times 2 = 20$ edges for the third stage, and $10 \times 1 = 10$ edges for the fourth and last stage of BN learning. These estimated stages closely match the observed shape for the metric gain function. Note that some of the edges might not be learned (about 10% in this case) or learned at a different stage from the expected one, due to noise originated from learning with a finite population size; however, the above estimates are still quite accurate. To confirm this rationale, Figure 5.5 plots the last generation of a run with BOA+K2 when solving the same problem with different number of subfunctions ($k = 5, m = 20$) and different trap size ($k = 4, m = 10$).

Table 5.1: Equivalent tournament size (s) and truncation threshold (τ) for the same selection intensity (I).

I	0.56	0.84	1.03	1.16	1.35	1.54	1.87
s	2	3	4	5	7	10	20
$\tau(\%)$	66	47	36	30	22	15	8

Once again, the metric gain decreases more or less at the expected points where there is a stage shift.

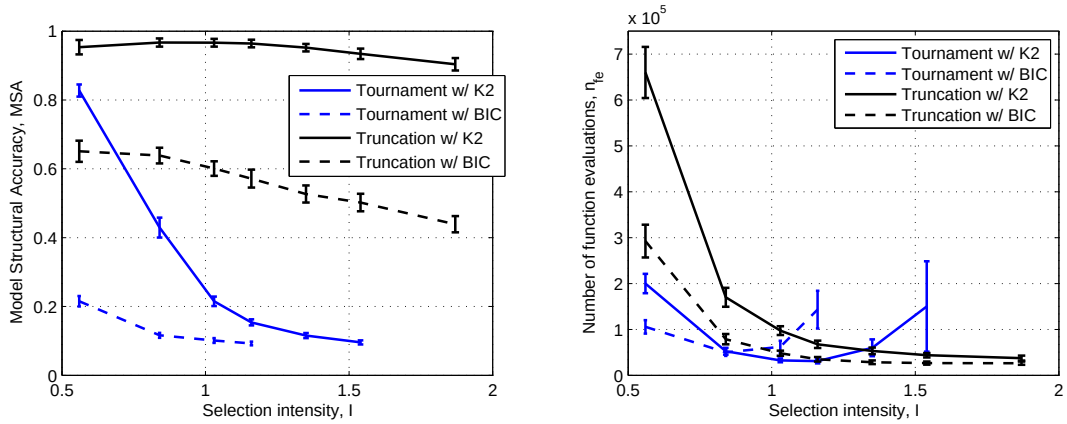
Typically, incorrect edges are added to the network at the later stages of learning, when the metric gain is smaller. This means that model overfitting comes mainly in the form of excessive number of incoming parents for the nodes that depend on some other interacting variables. For example, in stage 4 (Figure 5.4 (d)), when learning the dependency $X_1 \rightarrow X_5$, additional spurious dependencies are oftentimes added to the network, such as $X_6 \rightarrow X_5$, $X_7 \rightarrow X_5$, etc.

5.4 The Role of Selection

When EDAs model the set of promising solutions to guide the search, these are identified by a selection operator, which can have a great influence on their performance [67, 84, 144, 96, 177]. This section pays special attention to the role of selection in BOA. In particular, two widely used selection schemes in EDAs are considered: Tournament and truncation selection.

In order to compare the two selection operators on a fair basis, different configurations for both methods with equivalent selection intensity are considered. The relation between selection intensity I, tournament size s, and truncation threshold τ is taken from [14] and is shown in Table 5.1.

The influence of the selection strategy in BOA has been discussed in detail else-



(a) Model structural accuracy (b) Num. of function evaluations

Figure 5.6: Model structural accuracy and number of function evaluations for different selection-metric combinations when solving the trap-5 problem of size $\ell = 50$.

where [96]. Here, essential findings for the purpose of studying model overfitting are reviewed and extended to the BIC metric. Figure 5.6 shows the model quality and number of function evaluations for different combinations of selection methods and scoring metrics. From a model quality perspective, it is clear that (1) truncation selection performs better than tournament selection and (2) K2 metric performs better than BIC metric. Note that with tournament selection, while for small values of s the number of evaluations decreases, after some value of s, the number of evaluations starts to increase again. Curiously, this happens when the MSA approaches 0.1 (meaning that 90% of the edges are incorrect or unnecessary).

5.4.1 Selection as the Mating Pool Distribution Generator

Like in traditional genetics, the selection mechanism is responsible for ensuring the survival of the fittest in the population. In the context of EDAs, this is one of the most important components inherited from the evolutionary computation framework. However, in EDAs, which have a strong connection with data mining and classification, the selection operator can also be viewed as the *generator* of the

data set used to learn the probabilistic model at each generation. Since in EDAs one is interested in modeling the set of promising solutions, the selection operator indicates which individuals have relevant features to be modeled and propagated in the solution set. Before proceeding to the study of the selection strategy as the data set generator for learning the BNs, a simple analysis of the selection operators considered is made.

In terms of creating duplicate individuals in the population there are two responsible mechanisms. The selection operator explicitly assigns several copies of the same individual to the mating pool, where the number of copies is somewhat proportional to their fitness rank. This is the case for tournament, ranking, and proportional selection. Additionally, the model sampling procedure generates with a certain probability duplicates of the same individual, although selection implicitly controls how often this happens. Note that this probability will increase in time as the EDA starts focusing on more concrete regions of the search space. Clearly, the selection operator has some influence on this phenomenon as it explicitly regulates the convergence speed of the algorithm. Without loss of generality, consider that the replication of individuals done explicitly by the selection operator is the main source of duplicates in the population.

For the sake of simplicity, let us assume that all individuals have different fitness. Ordering the population by fitness, where the worst individual has rank 1 and the best has rank n, the probability that an individual with rank i wins a given tournament of size s is, for $i \geq s$, given by

$$p_i = \frac{\binom{i-1}{s-1}}{\binom{n-1}{s-1}} = \frac{(i-1)!(n-s)!}{(i-s)!(n-1)!} = \prod_{j=1}^{s-1} \frac{i-j}{n-j}, \qquad \text{for } s \geq 2. \tag{5.2}$$

Note that the worst $s-1$ individuals will never win a tournament, therefore for

$i < s$, $p_i = 0$.

Given that in tournament selection without replacement each individual participates in exactly s tournaments, the expected number of copies (c_i) in the mating pool for an individual of rank i is simply

$$c_i = s\ p_i. \tag{5.3}$$

For $i \gg s$, and consequently $n \gg s$, the distribution of the expected number of copies c_i can be approximated by a power distribution [7] with p.d.f.,

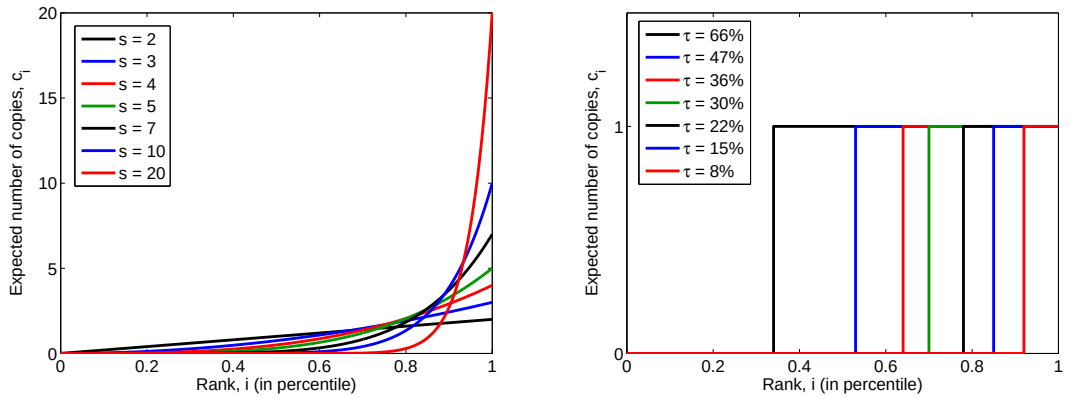
$$f(x) = \alpha\ x^{\alpha-1}, \quad 0 < x < 1, \alpha = s. \tag{5.4}$$

In this way, the distribution of c_i can be expressed for any population size, where the relative rank is given by $x = i/n$. Note that as the relative rank slightly decreases from 1 the corresponding number of expected copies rapidly decreases. This is particularly true for higher tournament sizes, when increasing the exponent of the power factor.

On the other hand, in truncation selection the expected number of copies for the selected individuals is one, which follows a uniform distribution with p.d.f.,

$$c_i = \begin{cases} 0, & \text{if } i < n\,(1 - (\tau/100)) \\ 1, & \text{otherwise.} \end{cases} \tag{5.5}$$

Figure 5.7 illustrates the distributions of the expected number of copies for each individual with rank expressed in percentile. The difference between the two selection methods is remarkable. While tournament selection assigns increasing relevance to top-ranked individuals according to a power distribution, truncation selection



(a) Tournament (b) Truncation

Figure 5.7: Distribution of the expected number of copies in the mating pool for (a) tournament and (b) truncation selection with different selection intensity values. Note that s and τ values generate the same selection intensity. Rank is expressed in percentile.

gives no particular preference to any of the selected individuals, all having the same frequency in the learning data set.

The differences between tournament and truncation distributions stress out two relevant features of any given selection method:

1. *Window size* which determines the proportion of unique individuals that are included in the mating pool.

2. *Distribution shape* which determines the relevance of each selected individual in the mating pool, in terms of the number of copies.

These features in a certain way control the tradeoff between exploration and exploitation in model structural learning in EDAs.

Clearly, tournament and truncation selection differ in both features. While the window size is deterministically defined in truncation selection—solutions above the threshold are included in the selected set and solutions below are not—in tournament selection, the choice of which individuals to include in the mating pool is a stochastic

process (except for the best solution and the worst $s-1$), but also guided by fitness rank. The probability of inclusion rapidly decreases with rank, particularly for larger tournament sizes, as can be seen in Figure 5.7 (a). In terms of distribution shape, the two selection methods also differ significantly. Tournament selection gives higher emphasis to top-ranked solutions according to a power distribution with $\alpha = s$. This means that the best solutions get approximately s copies in the mating pool, which forces the learned models to focus on particular features of these individuals, which contain good substructures, but also undesirable components due to stochastic noise[1].

Another way to look at tournament selection in comparison with truncation selection as the mating pool generator, is recognizing that this selection procedure acts as a biased data resampling on a uniform data set. The uniform data set is the set of unique selected solutions (solutions that will win at least one tournament), similar to what happens in truncation, while the resampling is performed when top-ranked individuals participate in more than one tournament. This sort of resampling is clearly biased by fitness.

A related topic in data mining is the generalization of features with low-density in the learning data set, generally known as learning from imbalanced data sets [82, 93, 170, 136]. One way to achieve this is to artificially create additional data instances that appears to have the feature of interest, but in a way that does little impact to the distribution of the population as a whole. The last remark however can be quite difficult to ensure, as the question of what is the "natural" distribution of the data does not have a straightforward answer. Quoting from [136]:

> When added to the original data set, these now appear as more instances with the feature, increasing the apparent count and increasing the feature

[1]Notice that selection operates at the entire solution level, making decisions on the basis of the overall fitness, and therefore can not "judge" single substructures or subsolutions.

> density in the overall data set. The added density means that mining tools will generalize their predictions from the multiplied data set. A problem is that any noise or bias present will be multiplied too.

This is indeed the problem with tournament selection when trying to model the overall problem structure.

5.5 Modeling Metric Gain when Overfitting

To analyze the effect of tournament size on resampling bias one must look at the cumulative distribution function (c.d.f.) of the power distribution, which is given by

$$F(x) = x^s, \quad 0 < x < 1, s \geq 1. \tag{5.6}$$

Note that $s = 1$ generates an uniform distribution (no resampling), as the mating pool becomes a complete copy of the population. For $s \geq 2$, the proportion of individuals in the mating pool with rank equal or less than x can be obtained by simply calculating $F(x)$, or alternatively, the market share of the $(1-x)$ top-ranked individuals given by $1 - F(x)$ (corresponding to the right-side area of the c.d.f.).

The overfitting due to noise coming from top-ranked individuals is certainly more likely to happen if considering a fairly small percentage of the population. Said differently, the smaller this proportion is, the more likely these individuals will contain the same misleading features that are induced by noise. On the other hand, this proportion should be significant enough in terms of relative frequencies so that it can influence the metric component that scores the likelihood of the model with respect to the data. How large or small should this proportion be depends obviously on the tournament size. For larger tournament sizes, this proportion is expected to be inferior to the case of smaller tournament sizes because the number of copies

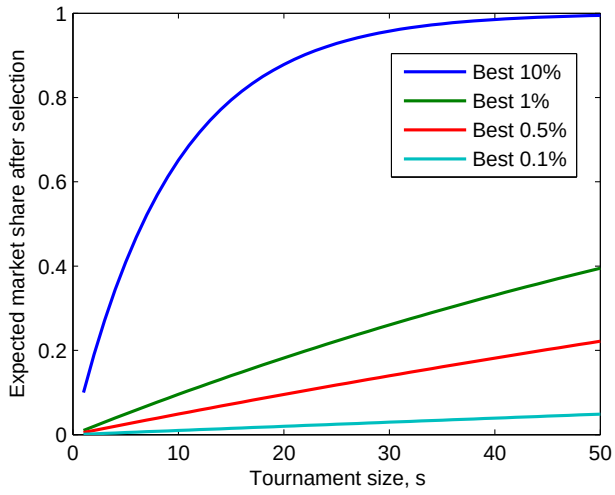


Figure 5.8: Expected market share of top-ranked individuals included in the mating pool after selection for infinite population size. For small proportions ($\leq 1\%$) of top-ranked individuals the relation between tournament size and the expected proportion in the mating pool after selection is approximately linear.

assigned to top individuals increases considerably. Therefore, it is recognized that this proportion should be small, but the exact proportion will differ from situation to situation.

To better ilustraste this argument, Figure 5.8 shows the power c.d.f. for several proportions of top-ranked individuals. It can be seen that for small proportions ($\leq 1\%$) of top-ranked individuals, the expected proportion in the mating pool after selection grows approximately linearly with the tournament size. Note that as the proportion considered is more elitist, the slope of the linear relationship approaches the proportion itself. For example, when considering the best 0.1%, the market share after selection with $s = 50$ is $4.88\% \approx 0.1\% \times 50$.

The bottom line of this rationale is to verify that, in the worst case, the noise in terms of counts or relative frequencies coming from the replication of top-ranked individuals grows linearly with the tournament size.

Consider now the possibility of adding an edge from a variable X_2 to another

variable X_1 due to nonlinearities introduced by tournament selection, knowing that these two variables are in fact independent from each other. To investigate the influence of the resampling done by successive tournaments, the score metric for the network where an edge is added from X_2 to X_1 must be derived. Given that both MDL and Bayesian metrics are decomposable, it is sufficient to look at the term corresponding to the node X_1. The metric gain G_{metric} obtained by splitting a leaf on X_2 in the tree encoding the parameters of X_1, and adding the corresponding edge to the network, is given by

$$G_{metric} = ScoreAfter - ScoreBefore - ComplexityPenalty, \tag{5.7}$$

where *ScoreAfter* is the metric score obtained after splitting the leaf into two new ones, *ScoreBefore* is the score obtained before the split (keeping X_1 independent from X_2), and *ComplexityPenalty* is the penalty associated with the increased complexity of adding one leaf to the tree. In BOA, if this gain is positive the split is accepted and the corresponding edge is inserted in the Bayesian network.

Due to its simplicity (compared with the K2 metric), the BIC metric is considered in the following calculations. The metric gain corresponding to adding an edge from X_2 to X_1 is

$$\begin{aligned} G_{BIC} = \quad & m(X_1X_2 = 00) \log_2 \left(\frac{m(X_1X_2 = 00)}{m(X_2 = 0)} \right) \\ & + m(X_1X_2 = 10) \log_2 \left(\frac{m(X_1X_2 = 10)}{m(X_2 = 0)} \right) \\ & + m(X_1X_2 = 01) \log_2 \left(\frac{m(X_1X_2 = 01)}{m(X_2 = 1)} \right) \\ & + m(X_1X_2 = 11) \log_2 \left(\frac{m(X_1X_2 = 11)}{m(X_2 = 1)} \right) \\ & - m(X_1 = 0) \log_2 \left(\frac{m(X_1 = 0)}{n} \right) \end{aligned} \tag{5.8}$$

$$
\begin{aligned}
&- m(X_1 = 1) \log_2 \left(\frac{m(X_1 = 1)}{n} \right) \\
&- \frac{1}{2} \log_2 (n) ,
\end{aligned}
$$

where $m(X_1X_2 = x_1x_2)$ is the number of individuals in the population with $X_1 = x_1$ and $X_2 = x_2$. Note that the first four terms correspond to *ScoreAfter*, the fifth and sixth terms express *ScoreBefore*, and the final term penalizes the score because of the complexity added to the network. Denoting $m(X_1X_2 = x_1x_2)$ by $m_{x_1x_2}$ and recognizing that $m(X_1 = x_1) = m(X_1X_2 = x_10) + m(X_1X_2 = x_11)$, as well as $n = m_{00} + m_{01} + m_{10} + m_{11}$, the previous expression can be expressed as

$$
\begin{aligned}
G_{BIC} = \quad & m_{00} \log_2 \left(\frac{m_{00}}{m_{00} + m_{10}} \right) + m_{10} \log_2 \left(\frac{m_{10}}{m_{00} + m_{10}} \right) \\
& + m_{01} \log_2 \left(\frac{m_{01}}{m_{01} + m_{11}} \right) + m_{11} \log_2 \left(\frac{m_{11}}{m_{01} + m_{11}} \right) \\
& - (m_{00} + m_{01}) \log_2 \left(\frac{m_{00} + m_{01}}{m_{00} + m_{01} + m_{10} + m_{11}} \right) \\
& - (m_{10} + m_{11}) \log_2 \left(\frac{m_{10} + m_{11}}{m_{00} + m_{01} + m_{10} + m_{11}} \right) \\
& - \frac{1}{2} \log_2 (m_{00} + m_{01} + m_{10} + m_{11}) .
\end{aligned}
\tag{5.9}
$$

Expressing in terms of relative frequencies, the gain can be formulated as

$$
G_{BIC} = n \; G'_{BIC} - \frac{1}{2} \log_2(n), \tag{5.10}
$$

where

$$
\begin{aligned}
G'_{BIC} = \quad & p_{00} \log_2 \left(\frac{p_{00}}{p_{00} + p_{10}} \right) + p_{10} \log_2 \left(\frac{p_{10}}{p_{00} + p_{10}} \right) \\
& + p_{01} \log_2 \left(\frac{p_{01}}{p_{01} + p_{11}} \right) + p_{11} \log_2 \left(\frac{p_{11}}{p_{01} + p_{11}} \right) \\
& - (p_{00} + p_{01}) \log_2 (p_{00} + p_{01}) - (p_{10} + p_{11}) \log_2 (p_{10} + p_{11}) .
\end{aligned}
\tag{5.11}
$$

The next step is to model the deviation from the actual frequencies, in a uniformly distributed mating pool (in terms of copies), to biased frequencies towards the noise induced by the replication of top-ranked individuals (power distribution). First, consider the frequencies on the uniform mating pool to be $p_{00} = p_{01} = p_{10} = p_{11} = 0.25$, which reveals independence between X_1 and X_2. Then, and without loss of generality, it is assumed that these frequencies are deviated towards equally increasing p_{00}, p_{11} and equally decreasing p_{01}, p_{10}. This assumption relies on the fact that the decrease in entropy (corresponding to an increase in score) will be achieved faster than for other possible configurations of pairwise frequency deviation. In this way, the case that can upper bound other possible deviations is analyzed.

Assuming that the deviation of the "true" frequencies is linear with respect to the tournament size, as argued before, the frequency deviation can be expressed as

$$\begin{aligned} p_{00} &\approx 0.25 + \Delta(s-1), \\ p_{01} &\approx 0.25 - \Delta(s-1), \\ p_{10} &\approx 0.25 - \Delta(s-1), \\ p_{11} &\approx 0.25 + \Delta(s-1), \end{aligned} \tag{5.12}$$

where Δ is the slope of the linear relationship plotted in Figure 5.8, therefore the exact value will depend on the proportion considered. Replacing (5.12) into (5.11) and denoting $(s-1)$ by s',

$$\begin{aligned} G'_{BIC} \approx \quad & (0.25 + \Delta s') \log_2 \left(\frac{0.25 + \Delta s'}{0.5} \right) + (0.25 - \Delta s') \log_2 \left(\frac{0.25 - \Delta s'}{0.5} \right) \\ & + (0.25 - \Delta s') \log_2 \left(\frac{0.25 - \Delta s'}{0.5} \right) + (0.25 + \Delta s') \log_2 \left(\frac{0.25 + \Delta s'}{0.5} \right) \\ & - 0.5 \log_2 (0.5) - 0.5 \log_2 (0.5) . \end{aligned} \tag{5.13}$$

Simplifying the previous equation,

$$G'_{BIC} \approx \quad (0.5 + 2\Delta s') \log_2 \left(\frac{0.25 + \Delta s'}{0.5} \right) + (0.5 - 2\Delta s') \log_2 \left(\frac{0.25 - \Delta s'}{0.5} \right) + 1. \tag{5.14}$$

Using the logarithm property $\log(a/b) = \log(a) - \log(b)$ and simplifying again,

$$G'_{BIC} \approx \quad (0.5 + 2\Delta s') \log_2(0.25 + \Delta s') + (0.5 - 2\Delta s') \log_2(0.25 - \Delta s') + 2. \tag{5.15}$$

Dividing both terms by 2,

$$\frac{1}{2} G'_{BIC} \approx \quad (0.25 + \Delta s') \log_2(0.25 + \Delta s') + (0.25 - \Delta s') \log_2(0.25 - \Delta s') + 1. \tag{5.16}$$

Looking at the function $x \log_2(x)$ for the interval $[0, 0.5]$, one can see that the first term in Equation 5.16 is relatively constant around -0.5. Therefore,

$$\frac{1}{2} G'_{BIC} \approx (0.25 - \Delta s') \log_2(0.25 - \Delta s') + 0.5, \tag{5.17}$$

or alternatively,

$$G'_{BIC} \approx 2 \, (0.25 - \Delta s') \log_2 \, (0.25 - \Delta s') + 1. \tag{5.18}$$

The approximate expression for the metric gain G'_{BIC} due to overfitting of top-ranked individuals in tournament selection is plotted in Figure 5.9. A value of $\Delta = 0.001$ is used (best 0.1%). Since the proportions considered will vary from 0.25 to 0 or 0.5, the Δ value will basically define the increment/decrement step of that same proportions. For example, for a higher $\Delta = 0.005$ the approximate expression

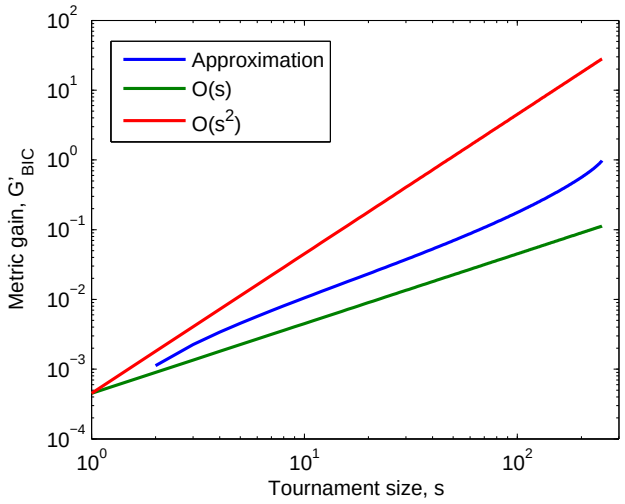


Figure 5.9: Approximated metric gain G'_{BIC} due to overfitting of top-ranked individuals in tournament selection. A value of $\Delta = 0.001$ is used (best 0.1%). The growth of the metric gain is somewhere between linear and quadratic, but closer to linear.

would be defined only for $s = [1, 50]$, instead of the plotted $s = [1, 250]$.

As can be seen, the metric gain scales close to linear in log-log scale, with the exception made for lower and higher values of s. This means a polynomial growth in linear scale, somewhere between linear and quadratic, which can be confirmed by comparison with reference curves. While the metric gain G'_{BIC} does not account for the factor n (population size) and the complexity penalty term $0.5 \log_2(n)$, it does tell us about the way the gain grows with respect to the tournament size s.

5.6 Improving Model Accuracy

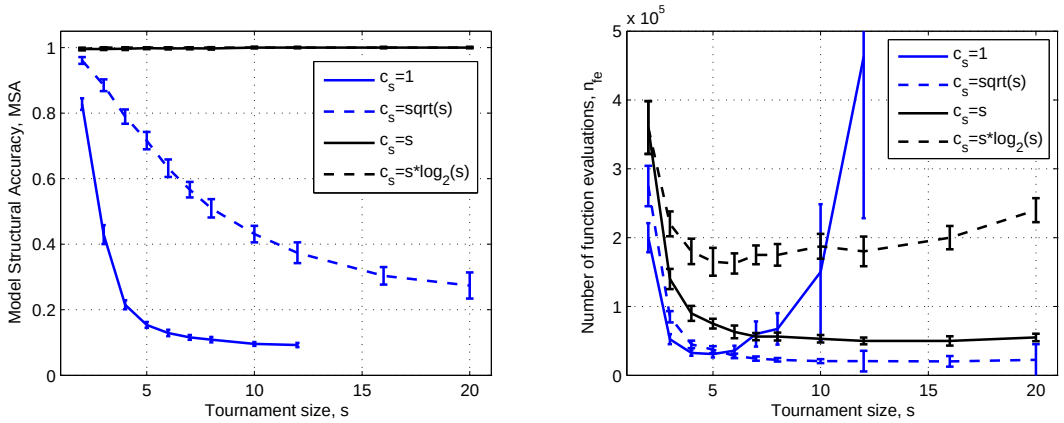
In this section, the complexity penalty of the scoring metric is changed in order to account for the power distribution of tournament selection. First, the efficacy of different penalty factors that are functions of the tournament size s is investigated. Second, the penalty found to be the most appropriate is validated for two additional problems that pose new sources of difficulty to model learning in Bayesian EDAs.

5.6.1 Adaptive Scoring Metric

While the metrics considered have different backgrounds, the penalty associated with each leaf addition is exactly the same: $0.5 \log_2(n)$ [125]. This becomes clear if one compares the logarithm of the K2 metric with the BIC metric. Additionally, the K2 metric has some implicit "penalty" for complex models, which comes from the marginal likelihood function. However, this is insufficient on its own for making models simple enough in Bayesian EDAs (both with standard CPTs and local structures). In fact, this extra penalty seems to be the reason for the K2 metric producing less complicated models than the BIC metric.

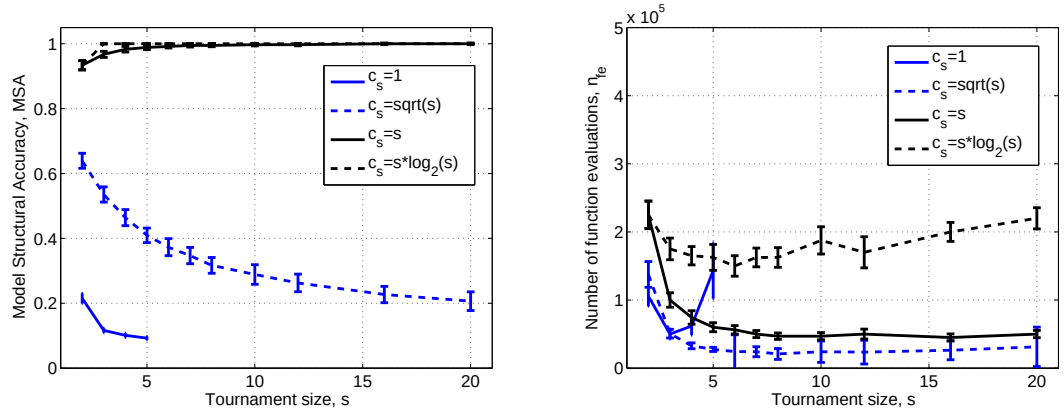
To compensate for the resampling bias of tournament selection, the complexity penalty is aggravated by a factor c_s that depends on the tournament size s, using $0.5 c_s \log_2(n)$ instead of the standard penalty. In this way, the greater the number of copies of top-ranked individuals in the mating pool, the more demanding is the scoring function in accepting an edge/leaf addition. From the previous section, it is known that the metric gain due to overfitting grows approximately as $\Theta(s)$, therefore different c_s values around that estimate are tested to investigate the corresponding response in terms of MSA and number of function evaluations. In particular, experiments for $c_s = \{\sqrt{s}, s, s \log_2(s)\}$ are performed and compared to the original penalty correction ($c_s = 1$).

Experiments for both BIC and K2 metrics were performed, although BIC was shown to produce excessively complex models (both for tournament and truncation selection). Figures 5.10 and 5.11 (for K2 and BIC metrics, respectively) show the model quality and corresponding evaluations for BOA with tournament selection using different complexity penalties. Despite the difference observed between metrics, their behavior when increasing the complexity penalty is somewhat equivalent. Already for $c_s = \sqrt{s}$, the model quality improves with respect to the standard case



(a) Model structural accuracy (b) Num. of function evaluations

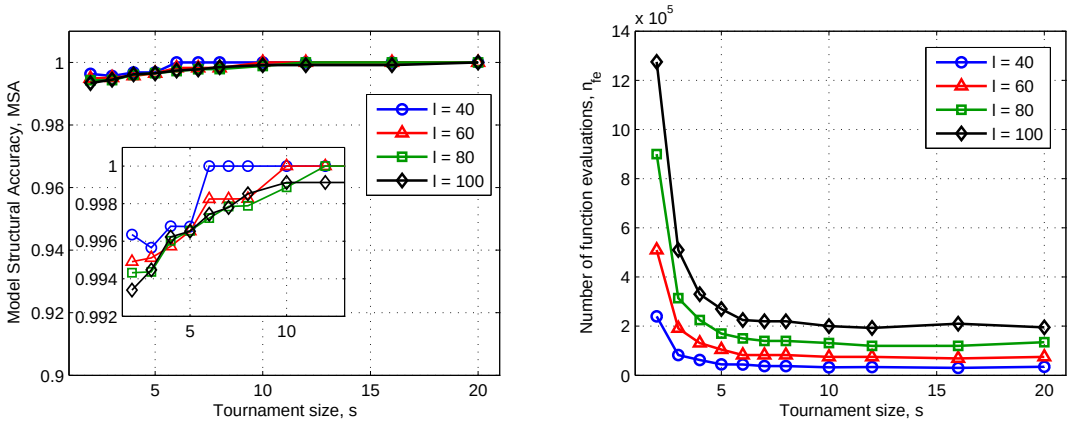
Figure 5.10: Model quality and number of function evaluations for different penalty correction values $c_s = \{1, \sqrt{s}, s, s\log_2(s)\}$ with the K2 metric, when solving the trap problem with $k = 5$ and $m = 10$.



(a) Model structural accuracy (b) Num. of function evaluations

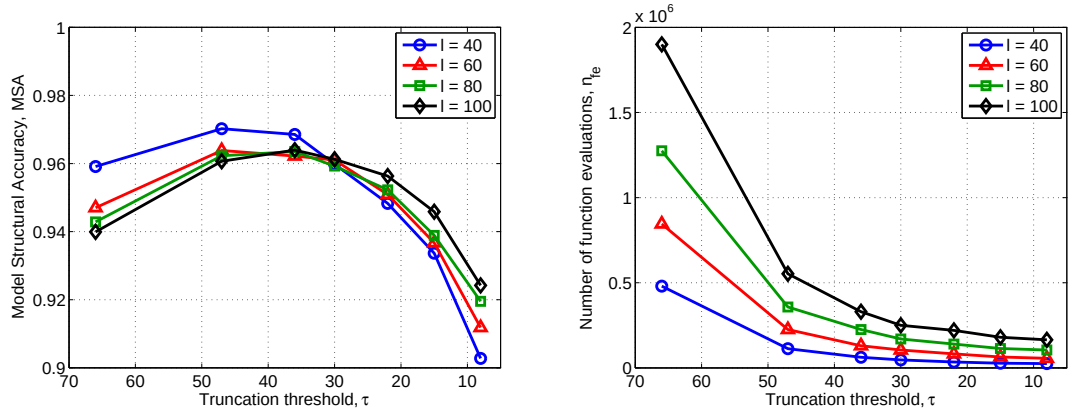
Figure 5.11: Model quality and number of function evaluations for different selection-dependent values $c_s = \{1, \sqrt{s}, s, s\log_2(s)\}$ with the BIC metric, when solving the trap problem with $k = 5$ and $m = 10$.

$c_s = 1$, but when considering $c_s = s$ and $c_s = s\log_2(s)$ the improvement is much better. Increasing the penalty by a factor of s or higher takes model structural accuracy very close to 100%. However, looking at the number of evaluations spent by each penalty, it is clear that $c_s = s\log_2(s)$ is too strong as a penalty because for larger s values it takes too many evaluations and the situation gets worse with



(a) Model structural accuracy (b) Num. of function evaluations

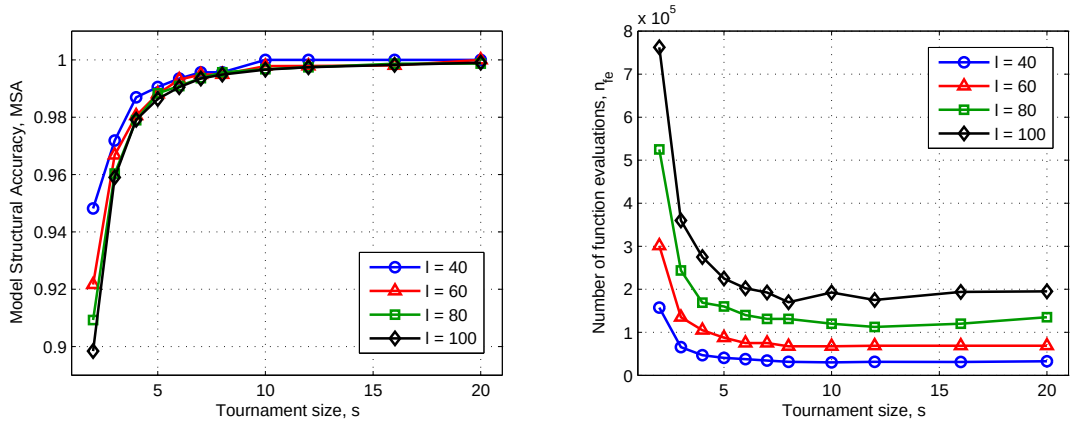
Figure 5.12: Model quality and number of function evaluations for tournament selection with the K2 metric and $s-$penalty ($c_s = s$). The problem is the same trap-5 with different sizes $\ell = \{40, 60, 80, 100\}$.



(a) Model structural accuracy (b) Num. of function evaluations

Figure 5.13: Model quality and number of function evaluations for truncation selection with the K2 metric. The problem is the same trap-5 with different sizes $\ell = \{40, 60, 80, 100\}$.

increasing s. On the other hand, the $s-$penalty ($c_s = s$) shows to be an adequate penalty because while obtaining high-quality models the number of evaluations is kept constant after some tournament size. This point us out to another advantage of the $s-$penalty, because it allows to have a wider range of s values for which BOA performs well and at a relatively low cost.



(a) Model structural accuracy (b) Num. of function evaluations

Figure 5.14: Model quality and number of function evaluations for tournament selection with the BIC metric and $s-$penalty ($c_s = s$). The problem is the same trap-5 with different sizes $\ell = \{40, 60, 80, 100\}$.

It is time now to look at the behavior of tournament selection with the $s-$penalty for different problem sizes and compare it to truncation selection with the standard penalty. Figures 5.12 and 5.13 show BOA using the K2 metric with tournament and truncation selection, respectively. Clearly, tournament selection with the $s-$penalty obtains better model quality than truncation selection with the standard penalty. Notice, however, that model quality is now plotted between 90% and 100%, because both methods obtain models of much better quality than tournament selection with the standard penalty. In terms of number of evaluations, tournament selection is still less expensive than truncation selection, but as selection intensity increases their costs become comparable.

Figure 5.14 shows BOA with tournament selection but now using the BIC metric and $s-$penalty. The results for truncation selection with the standard penalty are not plotted as their quality is much inferior to those with the $s-$penalty, with MSA typically varying between $40-70\%$ (see Figure 5.6). For this metric, the $s-$penalty also improves dramatically the model quality obtained with the standard penalty; however, the results are still not as good as for the K2 metric. For higher selection

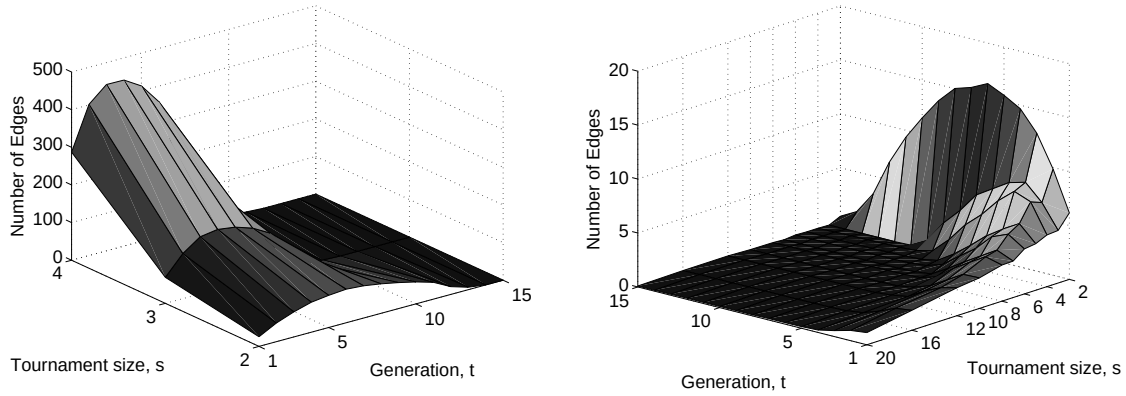
pressures the model quality is close to 100%, but as tournament size decreases the MSA degrades, which becomes more evident for increasing problem size. This confirms the tendency of the BIC metric to generate more dense networks.

5.6.2 Method Validation

While the previous results showed a significant improvement of model quality for BOA with the $s-$penalty, it is important to validate the proposed methodology when solving problems that pose new challenges in model learning for BOA and other Bayesian EDAs. Specifically, the onemax and hierarchical trap problems are considered.

While the optimization of onemax is quite easy for univariate EDAs, the probabilistic models built by multivariate EDAs are known to easily introduce unnecessary dependencies, which lead to increased population-sizing requirements. On the other hand, the hierarchical trap problem poses a more difficult challenge to EDAs, as important variable interactions are expressed at more than a single level. For these problems, the interactions at an upper level are too weak to be detected unless all lower levels are already solved.

A 27-bit hierarchical trap with three levels and $k = 3$ is considered (see Figure 2.3). Recall that subsolutions 000 and 111 are equally good except for the top level. However, the local optimum 000 is easier to climb, which requires the maintenance of all alternatives until a decision can be reached at the top level. Therefore, for this problem the hierarchical version of BOA is used by employing restricted tournament replacement to insert the offspring into the original population. Note that for hierarchical problems the probabilistic model must be capable of representing *chunks* of solutions from lower levels in a compact way so that only relevant features are considered [125].



(a) Standard penalty (b) $s-$penalty

Figure 5.15: Number of edges in the Bayesian network when solving the onemax problem with $\ell = 50$. Note that the ideal BN should not contain edges as the problem variables can be optimized independently from each other. The results for the (a) standard penalty and the (b) $s-$penalty are shown from a different angle so that they can be better observed. The $s-$penalty considerably reduces the number of edges captured by the network.

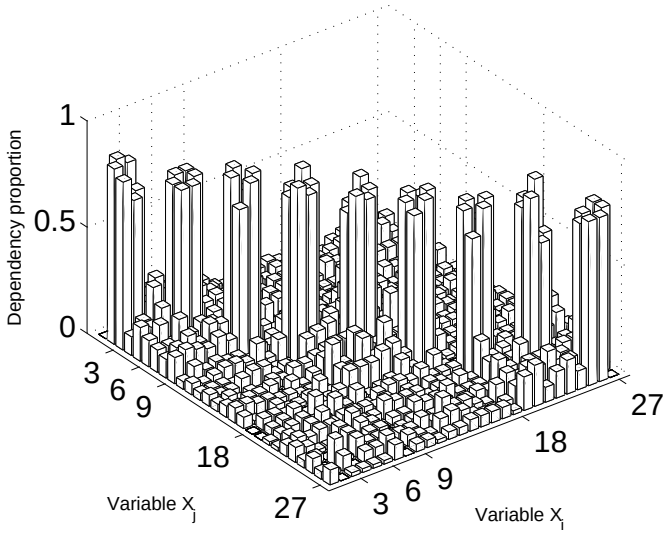
For the onemax problem, experiments are made for a string length of $\ell = 50$, while the number of edges present in the Bayesian network is recorded. Note that the ideal model should not contain any edges because the variables are independent from each other. Therefore, the fewer edges the more accurate is the model. Figure 5.15 compares the model quality of the standard penalty with the $s-$penalty, during the run and for different tournament sizes. The plots are shown from a different angle to better observe the results. The number of edges in the networks for the standard penalty is considerably higher than that observed for the proposed penalty. Furthermore, when increasing selection pressure for the standard penalty the resulting models become considerably more complex, which requires exponentially increasing population sizes with respect to tournament size s (to be able to solve the problem). This seems to reveal some sensitivity of BOA with tournament selection and the standard penalty in correctly modeling linear or approximately linear functions. On the contrary, when using the $s-$penalty the number of edges is drastically re-

duced, while the population-sizing requirements are considerably smaller, scaling approximately as $\Theta(s)$.

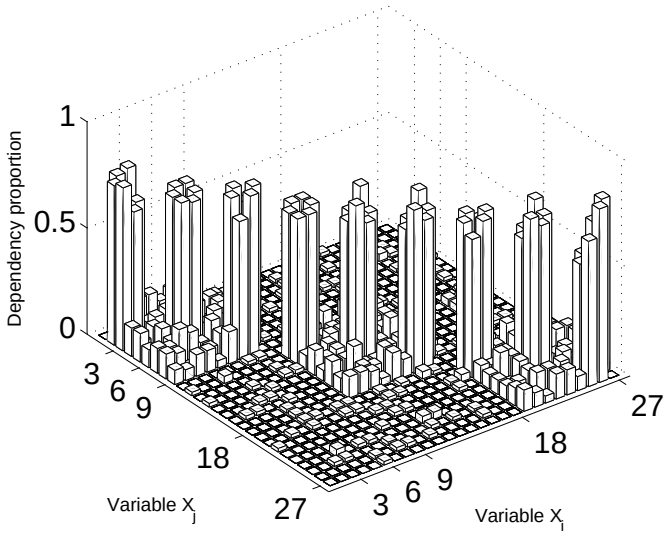
For the hierarchical trap problem the definition of correct/incorrect edges is not so clear. While the dependencies at the lower level are clearly correct and necessary dependencies, it is not clear when the dependencies at the higher levels should be captured. That will depend on whether the problem is already solved at the lower level. Therefore, instead of looking at the MSA, the results explicitly plot bivariate statistics relative to the dependencies learned by the Bayesian network.

Figure 5.16 plots the proportion of pairwise dependencies between variables for the 27-bit hierarchical trap captured by the Bayesian network in the last generation (when the optimum is found). The dependency proportion between any two variables refers to edges in both directions, which means that the graph is symmetrical with respect to the diagonal between points $(1, 1)$ and $(27, 27)$. The z-axis indicates the proportion of runs (out of 100) in which a certain dependency was learned by the Bayesian network. For example, a dependency proportion of 0.87 between X_1 and X_2 means that in 87 out of 100 runs there was a dependency $X_1 \rightarrow X_2$ or $X_2 \rightarrow X_1$. Clearly, the graph obtained for the $s-$penalty is more informative about the underlying structure of the hierarchical problem than the one obtained for the standard penalty. Note that stronger dependencies (nine linkage groups of order $k = 3$) reveal the base level structure, while more weak dependencies (three linkage groups of order $k = 9$) denote the structure of the middle level. Because the problem is solved at the higher level, the optimum is found before the model can capture the next level structure.

With respect to truncation selection using the standard penalty, the conclusions are similar to those made before. Truncation selection achieves comparable (although marginally inferior) model quality to tournament with the $s-$penalty. In



(a) Standard penalty



(b) $s-$penalty

Figure 5.16: Pairwise dependencies between variables for the 27-bit hierarchical trap problem, captured by Bayesian network in the last generation (when the optimum is found). The statistics presented do not consider the direction of the dependencies, which means that the graph is symmetrical with respect to the diagonal between points (1,1) and (27,27). The z-axis indicates the proportion of runs (out of 100) in which a dependency between X_i and X_j is learned by the Bayesian network.

terms of function evaluations requirements, tournament selection is still less expensive than truncation, but for higher selection intensities their cost become similar.

Together, these results demonstrate that tournament selection is an efficient selection method for Bayesian EDAs, as it is for genetic algorithms, as long as the complexity penalty of the scoring metric takes into account the power distribution in the mating pool. The greater the tournament size is, the more demanding the scoring metric should be in accepting edge/leaf additions.

5.7 Summary

While the main goal of BOA and other Bayesian EDAs is to perform efficient mixing of good substructures, it also provides additional information about the problem. The Bayesian networks learned during the run, which represent probabilistic dependencies between variables, is an important source of information that can be exploited to improve performance, or to get a better insight on the problem itself. Thus, it is important to investigate the relationship between the learned probabilistic models and the underlying problem structure.

This chapter makes a contribution towards understanding and improving model accuracy in the Bayesian optimization algorithm. Three main issues are addressed. First, a careful analysis of model building in BOA is made to understand how the problem structure is captured by learned dependencies in the BNs, as well as when incorrect or unnecessary dependencies are introduced. Specifically, it is shown that the Bayesian-Dirichlet (K2 version) scoring metric produces more accurate models than the Bayesian information criterion metric, and that spurious dependencies are learned mainly at the end of the network construction. Additionally, it has been identified the existence of $k-1$ different stages when learning strong interactions of

order k, due to their different contributions to the scoring metric.

The role of selection in Bayesian network learning is also investigated by looking at selection as the mating-pool distribution generator, which is found to have a great impact on model structural accuracy. Empirically, it has been shown that truncation selection produces considerably more accurate models in BOA than tournament selection. Intrigued by these results, a theoretical analysis of both selection methods is made to understand the reason behind such quality difference. The outcome of this study is that while tournament selection generates the mating pool according to a power distribution, which leads to model overfitting, truncation selection generates a more suitable uniform distribution for learning (using standard scoring metrics).

Finally, the metric gain originated from the resampling bias of tournament selection is modeled so that a quantitative measure can be obtained. Based on these results, the complexity penalty used in the scoring metrics is changed to counterbalance the corresponding metric gain in the same order of magnitude. The $s-$penalty which penalizes each edge/leaf addition by an additional factor of s (tournament size) is found to significantly improve the model structural accuracy. In essence, the greater the tournament size s is, the more demanding the metric should be in accepting edge/leaf additions. The proposed scoring metric is tested on the onemax, $m - k$ trap, and hierarchical trap problems, which present different challenges to probabilistic modeling in Bayesian EDAs. The results obtained for these problems show that the interpretability of the corresponding models is significantly improved with respect to the standard penalty.

These results demonstrate that tournament selection is an efficient selection method for Bayesian EDAs, as it is for traditional genetic algorithms, as long as the scoring metric is adjusted according to its natural distribution in the mating pool. Additionally, truncation selection is found to be more appropriate when using

standard scoring metrics. However, the corresponding model quality is inferior to the one obtained for tournament with the $s-$penalty. In terms of function evaluations requirements, tournament selection is less expensive than truncation, particularly for lower values of selection intensity (typical used values).

Overall, this work makes a step towards understanding and interpreting the probabilistic models in BOA, providing more interpretable models to assist efficiency enhancement techniques and human researchers. While other selection operators such as ranking or proportionate selection are not considered, the methodology developed in the chapter should provide enough guidelines to account for the non-uniform distributions generated by these operators.

Chapter 6

Improved Substructural Local Search in BOA

6.1 Introduction

While substructural local search in the Bayesian optimization algorithm is shown to speedup convergence to optimal solutions, its efficiency reduces for larger problem instances. The previous chapter illustrated that model accuracy can vary significantly if one does not take into account the duality between the selection operator and the scoring metric which guides model search.

This chapter proposes new developments to improve the efficiency of substructural local search in BOA. First, the utility of the $s-$penalty is investigated in terms of local search performance. Second, the presence of overlapping and hierarchical interactions is introduced. Overlapping difficulty is an important problem feature because many problems can have different interactions that share common components. Since the effect of overlapping variable interactions is similar to that of exogenous noise [51], recombination is likely to be more efficient than mutation-based

methods to solve problems with overlapping substructures [150]. Finally, to make SLS more effective for problems with substantial overlap between subproblems, a new SLS method is proposed based on the principles of loopy belief propagation. The utility of such local searcher is discussed for problems with overlapping and hierarchical interactions.

The next section reveals the influence of model accuracy when performing substructural local search in BOA, while Section 6.3 investigates the efficiency of substructural local search for the trap problem with overlapping interactions. Section 6.4 presents a brief introduction to belief propagation in graphical models and relevant applications for function optimization. To better deal with overlapping difficulty, Section 6.5 presents a new SLS method based on loopy belief propagation. In Section 6.6, the performance of the loopy local searcher is also tested for hierarchical problems. The chapter ends with a discussion about the efficiency of loopy SLS for different types of problems and with a brief summary.

6.2 Influence of Model Accuracy

The accuracy of model structures in BOA and other EDAs is important not only for their performance but also for a number of model-based efficiency enhancement techniques and for the offline interpretation of probabilistic models, which gives a further insight about the problem being solved. In this section, the influence of such model accuracy is demonstrated when performing substructural local search in BOA. Essentially, the speedup obtained for BOA with SLS using the proposed $s-$penalty (tailored for tournament selection) is compared to that obtained for the standard score metric (previous results). Figure 6.1 illustrates the results. The experimental setup is exactly the same as that in Chapter 4.

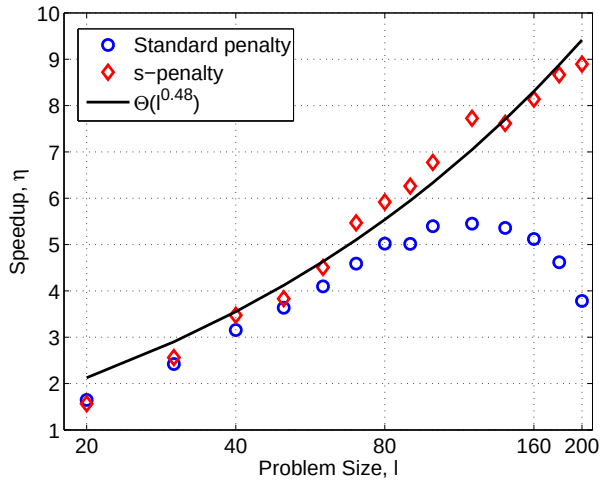


Figure 6.1: Speedup obtained when performing substructural local search in BOA with the standard penalty and the $s-$penalty for the trap-5 problem. The $s-$penalty approximates the theoretical bound for the speedup, which is given by the number of generations necessary to solve the problem with standard BOA: $\Theta(\sqrt{\ell})$.

While the standard penalty fails to maintain increasing speedups for larger problem instances, the $s-$penalty consistently presents substantial speedups—up to 9 for $\ell = 200$. In fact, the speedup approximates the theoretical upper bound of $O(\sqrt{\ell})$, which is the expected number of generations t_c for a properly designed GA to solve boundedly difficult problems [121, 51]. Note that the number of function evaluations is given by the product

$$n_{fe} = n \cdot t_c. \tag{6.1}$$

If a problem can be solved by only inspecting the initial population, the corresponding speedup in terms of function evaluations is given by

$$\eta = \frac{\Theta(n\sqrt{\ell})}{\Theta(n)} = \Theta(\sqrt{\ell}). \tag{6.2}$$

Despite the optimistic context of this speedup, it demonstrates that if crucial information about the problem is present in the initial populations, the corresponding

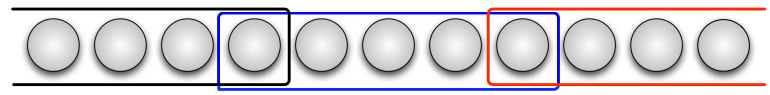
speedups can be up to $O(\sqrt{\ell})$. Clearly, this is not the case for exponentially-scaled or highly-noisy problems, in which situation iterative recombination of multiple solutions is a more efficient search approach.

6.3 Overlapping Difficulty

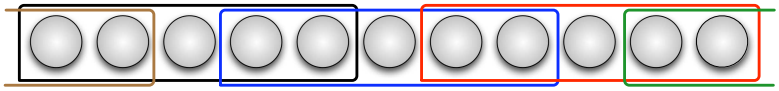
Up to this point in the thesis, only problems with non-overlapping interactions have been considered for evaluating the utility of SLS. While eCGA is unable to map such kind of interactions through probabilistic modeling, that is not the case with BOA. Bayesian networks are known to be able to express interactions of subproblems that share common variables by making use of conditional probabilities.

After ensuring that SLS can be efficient for exploring non-overlapping substructures, it is now time to address its applicability in more complex settings. Because the effect of overlapping variable interactions is similar to that of exogenous noise [51], mutation-based search methods are known to have difficulty in dealing with such problems [150]. Nevertheless, this section investigates the performance of BOA with SLS for several overlapping trap-5 problems. Figure 6.2 shows the variable mapping of different subfunctions for the overlapping version of the problem. Notice that for the case of $o = 3$, every other variable is associated with three different subfunctions, and each subfunction overlaps with another four, which adds substantial noise to the decision-making for each subproblem.

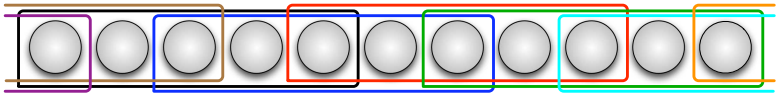
Figure 6.3 details the performance of substructural local search in BOA for the trap-5 problem with overlapping subfunctions ($o = \{1, 2, 3\}$). Essentially, when the degree of overlapping between different subfunctions increases, the efficiency of performing local search reduces substantially. Even for small overlap ($o = 1$), the speedup obtained is quite inferior to the non-overlapping case. These results



(a) One-overlapping variable ($o = 1$)



(b) Two-overlapping variables ($o = 2$)

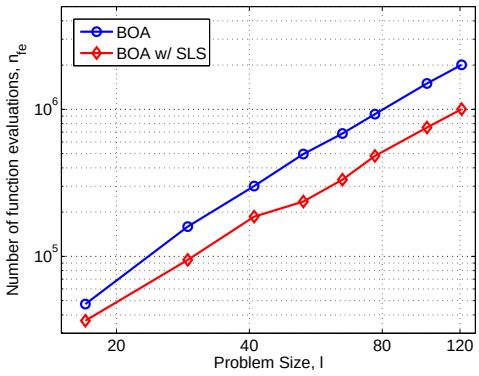


(c) Three-overlapping variables ($o = 3$)

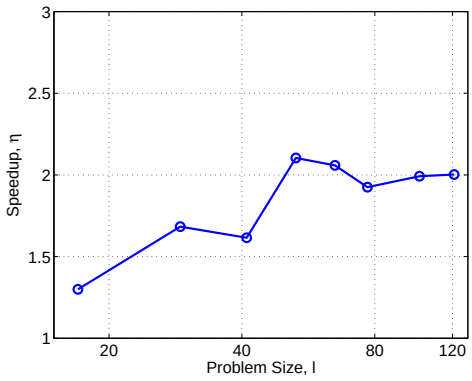
Figure 6.2: Variable mapping for the overlapping trap-5 problem. Notice that for $o = 3$, one in every two variables is associated with three different subfunctions, and each subfunction overlaps with another four, introducing a significant amount of noise when solving a particular subproblem.

are not surprising given the nature of the local searcher. When searching for the best substructure at a given subproblem, the decision-making does not take into account the corresponding context. Because different subproblems are solved in a particular sequence, the best subsolution for a subproblem considered in isolation might not be the best choice when considering other subproblems that overlap with the first. While this is not the case for the overlapping trap-5 problem, because all subproblems have the same global optimum at 11111, the local searcher can still be deceived.

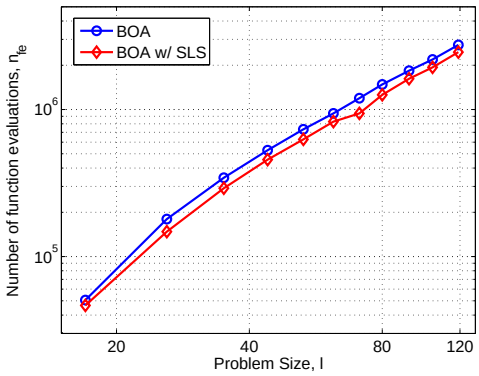
Consider the following example, where two different trap-5 subproblems overlap in two variables (X_4 and X_5), being the total problem size $\ell = 8$. When performing local search, the initial solution 00000000 has fitness $f = 4 + 4 = 8$, but when considering the best substructure for the first partition 11111000 the corresponding total fitness decreases to $f = 5 + 2 = 7$. While locally the best substructure is identified, the decrease in the overall fitness will not accept the move. Even if the order of visit for the neighborhoods is randomly shuffled each time local



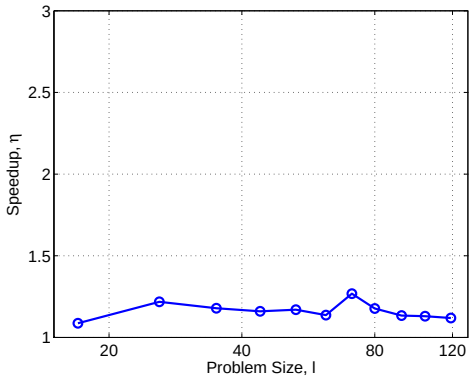
(a) Num. of function evaluations, $o = 1$



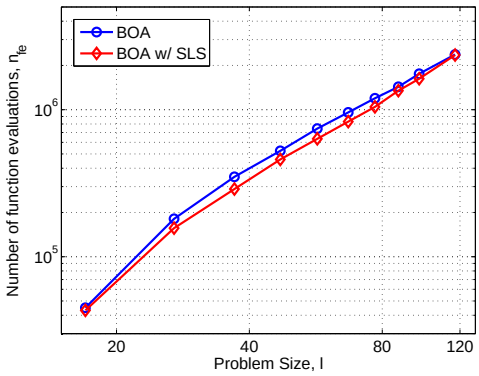
(b) Speedup, $o = 1$



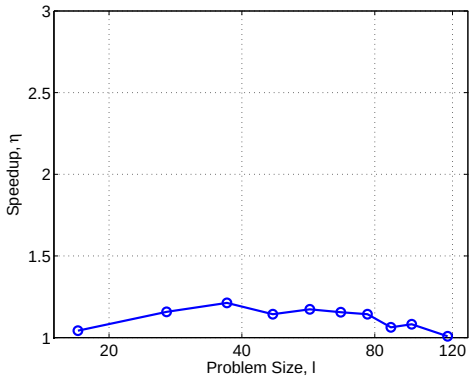
(c) Num. of function evaluations, $o = 2$



(d) Speedup, $o = 2$



(e) Num. of function evaluations, $o = 3$



(f) Speedup, $o = 3$

Figure 6.3: Number of function evaluations (left) and corresponding speedup (right) for BOA with SLS, when solving the trap-5 problem containing several overlapping variables between subfunctions. Overlap of $o = 1$ (top), $o = 2$ (middle), and $o = 3$ (bottom) variables are considered.

search is performed, there is no guarantee that all possibilities are covered for highly overlapping problems.

Therefore, the context of a given variable should be addressed to make a correct decision about the best value for that variable. This can be done for instance with dynamic programming or loopy belief propagation. The latter is introduced in the next section.

6.4 Loopy Belief Propagation

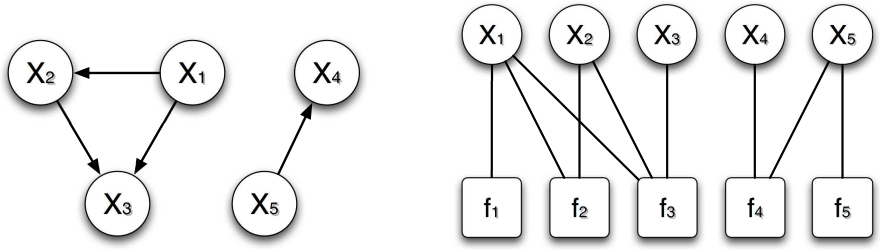
Belief propagation (BP) [124] is a method for performing exact and approximate inference in graphical models, which has enjoyed increasing popularity over the last years. Although BP has been reinvented several times in different fields [92, 113], it is mainly applied to two tasks: (1) obtaining marginal probabilities for some of the variables, or (2) finding the most probable explanation or instance for the graphical model. These two versions are known as the sum-product and max-product algorithms.

6.4.1 A Brief Introduction to Belief Propagation

BP algorithms are typically applied to factor graphs [92], which can be seen as a unifying representation for both Bayesian networks and Markov networks [88]. Factor graphs explicitly express the factorization structure of the corresponding probability distribution. Consider a function $g(X)$ whose joint probability distribution can be factorized in several local functions, such that

$$g(x_1, x_2, \ldots, x_\ell) = \frac{1}{Z} \prod_{I \in \mathcal{F}} f_I(x_{N_I}), \tag{6.3}$$

where $Z = \sum_x \prod_{I \in \mathcal{F}} f_I(x_{N_I})$ is a normalization constant, I is the factor index, N_I is



(a) Bayesian network (b) Factor graph

Figure 6.4: Example of a (a) Bayesian network and its equivalent representation as a (b) factor graph. Note that each factor corresponds to a conditional probability table, therefore the number of variable and factor nodes is the same.

the subset of variable indices associated with factor I, and factor f_I is a nonnegative function. Note that for a Bayesian network each factor corresponds to a conditional probability table.

A factor graph 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 $i \in N_I$, meaning that factor f_I depends on x_i. The neighbors of a factor node I are precisely variables N_I, while the neighbors N_i of a variable node i are the factors that depend on that variable. Factor nodes are typically represented as squares and variable nodes as circles.

An example of a Bayesian network, along with the corresponding representation as a factor graph, is presented in Figure 6.4. The factor graph represents the following factorization

$$g(x_1, x_2, x_3, x_4, x_5) = \frac{1}{Z} f_1(x_1) f_2(x_1, x_2) f_3(x_1, x_2, x_3) f_4(x_4, x_5) f_5(x_5). \tag{6.4}$$

If one substitutes the factor functions by the corresponding conditional probabilities, the joint probability distribution of a Bayesian network is obtained.

When BP is applied to cyclic graphs it is often referred as *loopy* belief propagation (LBP). In this situation, the convergence to exact beliefs can not be guaranteed as it is for acyclic graphs (without loops). However, empirical studies have shown that good approximate beliefs can be obtained for several domains (see [113] for an extensive list).

The inference performed by BP is done by message-passing between the nodes of the graphical model. Each node sends and receives messages from its neighbors until a stable state is reached. The outgoing messages are functions of incoming messages at each node. This iterative process is repeated according to some schedule that describes the sequence of message updates in time [113].

When performing BP in factor graphs, there are two types of messages: messages $m_{I\to i}$, sent from factors $I \in \mathcal{F}$ to neighboring variables $i \in N_I$, and messages $m_{i\to I}$, sent from variables $i \in \mathcal{V}$ to neighboring factors $I \in N_i$. The new messages m' are given in terms of the incoming messages by the following update rules:

$$m'_{i\to I}(x_i) = \prod_{J\in N_i\setminus I} m_{J\to i}(x_i) \qquad \forall i \in \mathcal{V},\ \ \forall I \in N_i, \tag{6.5}$$

$$m'_{I\to i}(x_i) = \sum_{x_{N_I\setminus i}} f_I(x_{N_I}) \prod_{j\in N_I\setminus i} m_{j\to I}(x_j) \qquad \forall I \in \mathcal{F},\ \ \forall i \in N_I, \tag{6.6}$$

$$m'_{I\to i}(x_i) = \max_{x_{N_I\setminus i}} \left(f_I(x_{N_I}) \prod_{j\in N_I\setminus i} m_{j\to I}(x_j) \right) \qquad \forall I \in \mathcal{F},\ \ \forall i \in N_I, \tag{6.7}$$

where $N_i\setminus I$ represents the set of neighboring factor nodes of variable node i excluding node I, $N_I \setminus i$ represents the set of neighboring variable nodes of factor node I excluding node i, and $x_{N_I\setminus i}$ stands for a possible combination of values that all

variables but X_i in X_{N_I} can take while variable X_i remains instantiated with value x_i.

For the sum-product algorithm (calculation of marginal probabilities), equations 6.5 and 6.6 are used, while for the max-product algorithm (most probable configuration) equations 6.5 and 6.7 should be used instead. When messages stop changing over a certain period of time, the BP algorithm has converged and marginal functions (sum-product) or max-marginals (max-product) can be obtained as the normalized product of all messages received for X_i:

$$g_i(x_i) \propto \prod_{I \in N_i} m_{I \to i}(x_i). \tag{6.8}$$

For the max-product algorithm, the most probable configuration (MPC) for each variable X_i is obtained by assigning the value associated with the highest probability at each max-marginal:

$$MPC(X_i) = \arg\max_{x_i} \left(g_i(x_i) \right). \tag{6.9}$$

When applying BP algorithms, three types of parameters need to be defined [111]:

Scheduling policy considers the way messages are spread through the nodes of the factor graph. A synchronous model can be used, where a clock triggers when messages should be sent. Alternatively, rule-based scheduling can be employed, in which case the message-sending is triggered by the number of messages received by a particular node or a set of nodes [111].

Stopping criteria consider different criteria to stop the BP algorithm when convergence can not be guaranteed. This is particularly important when per-

forming BP in cyclic graphs where good results can still be obtained without reaching a stable state.

Initial settings consist of a number of initial parameters that need to be set by the user and may depend upon the graph itself. The parameters include values for the initial messages, an allowed difference between two messages to declare them similar, the maximum number of messages before stopping the algorithm, number of message comparisons needed to identify a stable state, and others.

For more details about parameter setting in BP algorithms the reader is referred elsewhere [111, 113, 124, 92].

6.4.2 Message-Passing Techniques for Optimization

Several message-passing algorithms have been developed and applied to different optimization problems. The idea is to associate a probability distribution to the function to be optimized in such a way that the most probable value of the distribution is reached for the solution(s) that optimize the function [111].

Warning and survey propagation have been proposed for solving satisfiability problems [17, 38]. The constraint satisfaction problem is mapped into a graph factor where factor nodes represent the clauses to be satisfied. Message passing between variables and clauses is performed with the goal of finding satisfiable assignments for all clauses. The max-product algorithm has also been used for finding the maximum weight matching in a bipartite graph [11]. Essentially, the authors consider a weighted complete bipartite graph and define a probability distribution on it, whose MPC corresponds to the maximum weight matching in that graph.

Ochoa *et al.* [123] introduced BP to the polytree distribution algorithm (PADA). The objective is to construct higher-order marginal distributions from the bivariate

probabilities corresponding to a polytree distribution. These results were later extended [78] to the FDA. The calculation of the most probable configuration has been also used with UMDA and EDAs based on trees and polytrees [161]. These results were extended to pairwise Markov networks as well [143]. Recent developments in BP have also been applied to EDAs for obtaining higher order consistent marginal probabilities [117], and for calculating the MPCs for models based on Kikuchi approximations [79]. Nevertheless, for all these cases, the probabilistic models are simpler than Bayesian networks and factor graphs, or the structure of the model has to be known a priori.

Recognizing the potential of BP for Bayesian EDAs, Mendiburu *et al.* [110] introduced belief propagation to the estimation of Bayesian networks algorithm (EBNA), which is very similar to BOA. The idea is to combine probabilistic logic sampling (PLS) with loopy belief propagation to sample the offspring population. Specifically, $n-1$ individuals are sampled through PLS and the remaining individual is instantiated with the most probable configuration for the current Bayesian network. The Bayesian network is mapped into an equivalent factor graph so that the max-product algorithm can be applied to obtain the new individual. Although the authors concluded that this modification allowed an improvement in the optimization capabilities of EBNA, the results fail to demonstrate great improvements both in solution quality and number of function evaluations spent.

While the calculation of the most probable configuration of the Bayesian network at each generation is expected to generate a good solution, its relative quality is strongly dependent upon the current stage of the search. It seems clear that high-quality solutions can only be generated by LBP when BOA starts focusing on more concrete regions of the search space. On the other hand, instead of performing loopy belief propagation based on the conditional probabilities, substructural fitness

information can be used for the factor nodes. Although probabilities represent likely substructures, using the associated fitness provides more direct information when looking for solutions with high quality. That is what is proposed in the next section.

6.5 BOA with Loopy Substructural Local Search

6.5.1 Method Description

This section describes a substructural local searcher based on loopy belief propagation, and how it can be integrated in BOA. The resulting method which is named as loopy substructural local search (loopy SLS) uses substructural fitness information $\bar{f}(X_i, \Pi_i)$ to guide the max-product algorithm in finding the MPC, which is the solution that is expected to maximize fitness based on the contribution of its substructures.

Regarding the parameterization of BP, the maximum number of iterations that the algorithm is allowed to run is set to 2ℓ, while the allowed difference when comparing two messages is of at least 10^{-6} (otherwise messages are considered to be similar). These are typical parameter values from the literature. The update schedule used is the maximum residual updating [36], which calculates all residuals (difference between updated and current messages) and updates only the message with the largest residual. Consequently, only the residuals that depend on the updated message need to be recalculated.

If the factor graph is acyclic, BP will converge towards a unique fixed point within a finite number of iterations, while the beliefs can be shown to be exact. However, if the factor graph contains loops, which is the typical situation when translating a Bayesian network from BOA, the result can be only interpreted as an approximated solution. Therefore, two different situations can arise when performing loopy SLS:

(1) the max-product algorithm might not converge to a stable point and (2) even in case of convergence, the solution can present ties for certain positions.

If the LBP algorithm does not converge to a stable state, the configuration found after the maximum number of iterations (2ℓ) is still used as the result of loopy SLS. While this solution is not guaranteed to be the MPC, it is certainly a local optimum[1] and therefore should be inserted in the population. Another situation that can happen is the presence of ties for certain variables, where the most probable configuration can not be decided between 0 and 1. Typically, for problems tested with BOA, this occasionally occurs but for very few variables. Therefore, when the MPC presents ties, the loopy SLS enumerates all possible configurations and insert them in the population as the result of local search. To account for the rare case where the number of ties n_t is beyond reasonable, the maximum number of possible configurations/individuals returned by local search is set to ℓ, in which case the configurations chosen are randomly selected from all possible 2^{n_t}.

The loopy SLS method presents several differences from the proposal by Mendirubu *et al.* [110]. The most significative difference is that the factor nodes use fitness information instead of the traditional approach in BP which is to use the conditional probabilities stored in CPTs. The motivation for doing so is discussed later with a detailed example. By using fitness information, the algorithm becomes a local search method based on loopy message-passing principles— therefore the name of loopy substructural local search. Another important innovation is the selection of relevant factor nodes to perform loopy SLS. Essentially, factor nodes (and corresponding edges) whose variable set is a subset of another factor are removed. Consider the previous example of a factor graph in Figure 6.4. The relevant factor nodes are f_3 and f_4 because the variable domain of the remaining factors is already

[1] Assuming that the optimal configuration is found for at least one subproblem.

Loopy Substructural Local Search (Loopy SLS)

(1) Map the current Bayesian network `B` into a factor graph `F`, where factor nodes store substructural fitness information $\bar{f}(X_i, \Pi_i)$.

(2) Remove factor nodes (and corresponding edges) whose variable set is a subset of another factor in `F`.

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

(4) **If** $n_t = 0$, instantiate an individual with the values from `MPC`;
Else If $2^{n_t} \leq \ell$, enumerate all possible 2^{n_t} configurations and instantiate them in 2^{n_t} different individuals;
Else enumerate ℓ randomly chosen configurations out of 2^{n_t} and instantiate them in ℓ different individuals.

(5) Evaluate the resulting individuals.

Figure 6.5: Pseudocode of the loopy substructural local search in BOA.

included in these factors. Note that this simplification of the BN is possible because $\bar{f}(X_1, X_2, X_3)$ (stored in factor f_3) is more informative than both $\bar{f}(X_1, X_2)$ and $\bar{f}(X_1)$ (stored in factors f_2 and f_1). In the same way, $\bar{f}(X_4, X_5)$ already contains information from $\bar{f}(X_5)$. This straightforward procedure simplifies and improves the information exchange between nodes in the factor graph. In addition, the method for dealing with ties is also a novel contribution.

Figure 6.5 presents the pseudocode for the loopy substructural local searcher in BOA. The algorithm starts by mapping the current Bayesian network to a factor graph which stores fitness information in the factor nodes. The method proceeds by removing all factor nodes that are not relevant to local search, simplifying the search complexity for the MPC. The max-product algorithm is then applied to the resulting graph and the result is inserted in the population. Depending upon the number of possible ties for certain variable positions, up to ℓ different individuals can be inserted in the population. This is a reasonable number in terms of population-sizing

which is known to scale as $\Theta(\ell \log \ell)$ [177]. Finally, it is important to mention that the loopy local search takes place between steps 5 and 6 in BOA+SLS pseudocode, previously detailed in Figure 4.7.

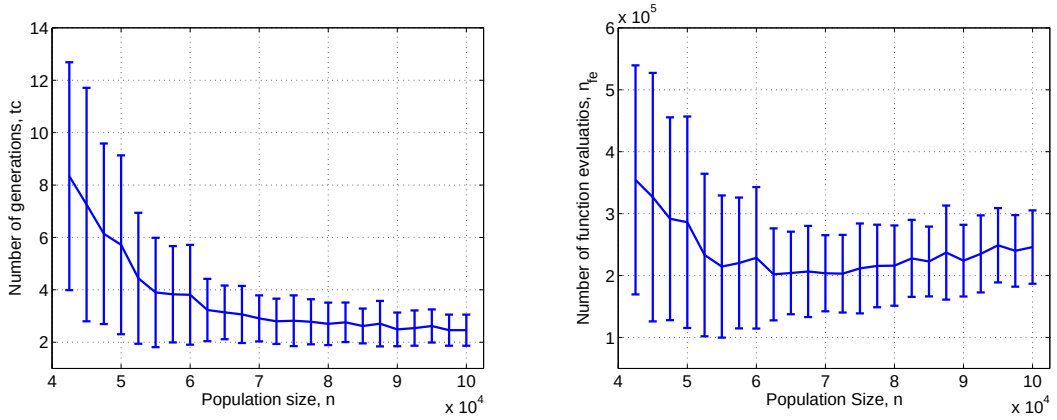
6.5.2 Results

This section presents the results obtained for loopy SLS in both overlapping and non-overlapping trap-5 problems. Apart from loopy SLS, the proposal by Mendiburu *et al.* [110] is also considered in the experiments. It should be clear that the only difference between the two alternatives is that loopy SLS uses (1) substructural fitness information instead of conditional probabilities for the factor nodes and (2) removal of non-relevant factors. Other aspects such as message-scheduling, ties management, and parameters are set similarly (as described for loopy SLS), to focus the experimental comparison on the capability for generating high-quality solutions, rather than comparing different configurations of the max-product algorithm.

Experimental Setup

While for the previous version of the local searcher, the bisection method returned the adequate population size for both standard BOA and BOA+SLS, when applying loopy SLS this method seems not to be the most adequate. This is particularly true for overlapping problems. Figure 6.6 shows the influence of the population size in the number of generations and the corresponding number of evaluations to solve an instance of the overlapping trap-5 problem ($o = 2$).

Clearly, the minimal population size required to solve the problem is not the most adequate in terms of minimizing evaluations of the fitness function. As the population size increases the number of generations necessary to converge to global optima reduces substantially. However, because increasing population size is a cost

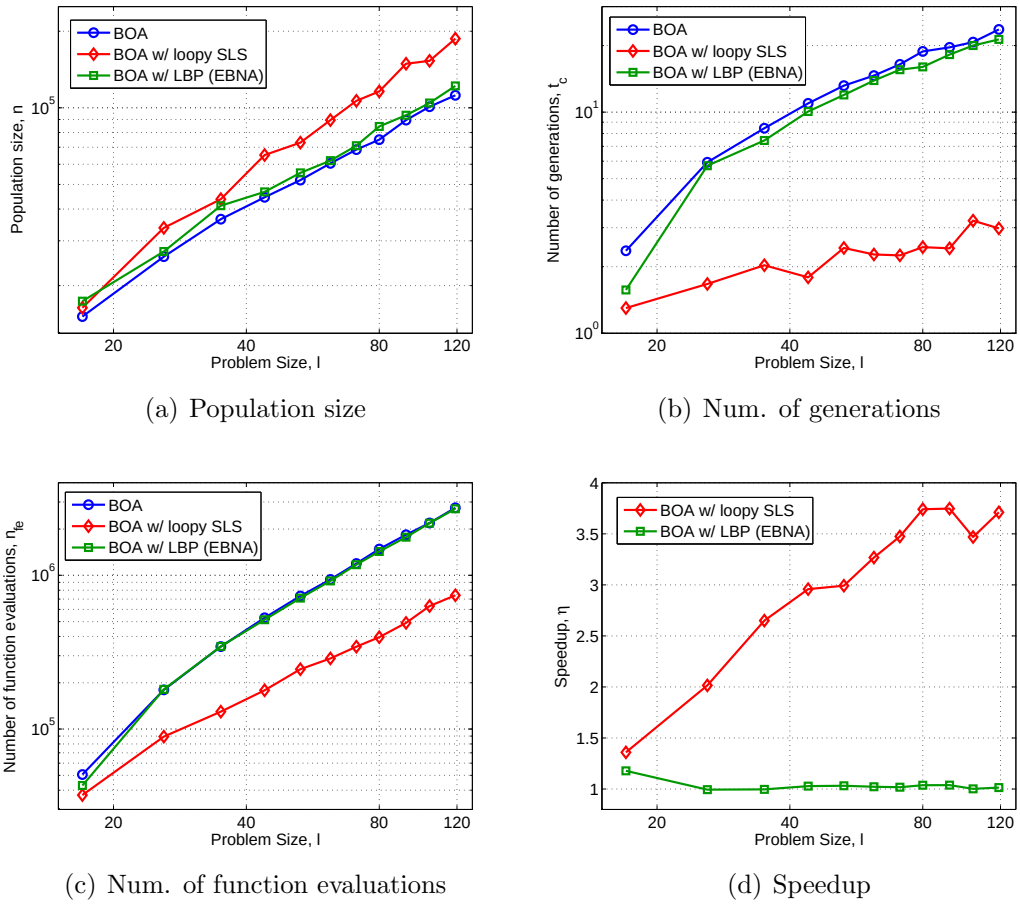


(a) Num. of generations (b) Num. of function evaluations

Figure 6.6: Influence of population size in the number of generations and function evaluations required by BOA with loopy SLS to solve an instance of the overlapping trap-5 problem (with $o = 2$ and $\ell = 44$).

payed in function evaluations (larger population sizes require more evaluations), there is an optimal population size. When the decrease in number of iterations slows down (after reducing significantly), the increase in population size seems no longer to pay off in terms of function evaluations ($n > 72500$).

Therefore, in addition to the bisection method, a simple population-size tuning method is used. Initially, the traditional bisection method is performed to obtain the minimal required population size. Afterwards, the population is increased as long as the number of function evaluations spent decreases. To make the method more robust to noise, it is required that two consecutive population sizes do not improve the result to stop the tuning method. For example, searching in the plot of Figure 6.6 (b), the method would start at $n = 42500$ and finish with $n = 55000$. Note that the number of evaluations have been reduced from 350000 to 220000. It can also be seen that the number of evaluations could be further optimized at $n = 62500$; however, this is directly related to the interval size used for the tuning method. Smaller intervals for the population size may have to deal with more noisy decisions, while larger intervals may miss important gains. Consequently, the



(a) Population size

(b) Num. of generations

(c) Num. of function evaluations

(d) Speedup

Figure 6.7: Results for BOA with both standard LBP and loopy SLS when solving the two-overlapping trap-5 problem ($o = 2$).

incremental interval for the population size is set according to the smallest interval used by bisection, which in this case is $n = 5000$, therefore finding $n = 62500$ as the most appropriate value.

Loopy SLS *versus* standard LBP

The results for BOA with both standard loopy BP (as proposed for EBNA) and loopy SLS are presented in Figure 6.7. The two alternatives present a very different behavior. While the LBP algorithm behaves very similarly to the original BOA, preferring smaller population sizes but taking more iterations and consequently eval-

uations, the loopy SLS seems to take advantage from using larger population sizes. Note that both alternatives use the same method to tune the population size. Nevertheless, increasing the population size for standard LBP does not reduce the number of generations necessary to solve the problem. Although minor gains are obtained with LBP for some problem instances, the corresponding speedup is very close to one.

Loopy SLS can effectively take advantage from larger populations to gather more accurate information to speedup the solution of the problem. More accurate fitness information allows the loopy local searcher to converge faster to optimal solutions. If conditional probabilities are used instead (as in standard LBP), the algorithm requires a certain number of generations for selecting and propagating the best substructures until their sampling probability becomes significant enough. For example, consider the trap-5 function with the local optimum at 00000 and the global optimum at 11111. Initially, the local optimum will dominate the population because it's much easier to climb. Later on, when both optima are the most frequent alternatives, the selection process starts propagating 11111 over 00000. Only at this stage, the max-product algorithm based of conditional probabilities is expected to return 11111 as the most probable configuration. On the other hand, when using substructural fitness information, once the fitness surrogate is accurate enough to identify 11111 as a better alternative than 00000, the MPC is expected to return the optimal solution. Consequently, BOA with loopy SLS takes advantage from using larger populations by building a more accurate fitness surrogate model.

Another important difference between the two approaches is the removal of factors which are not relevant to the MPC search. This is directly related to the dependency structure used by Bayesian networks to represent interactions among several variables. While this structure is required to be able to sample new instances

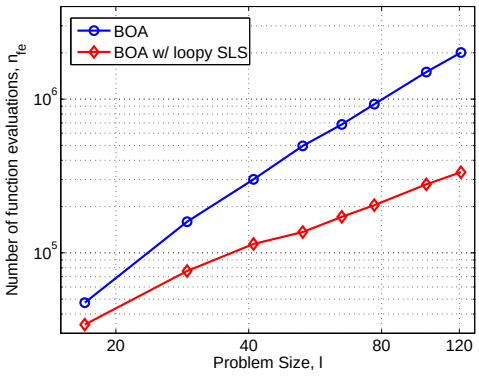
with PLS, it is not necessary or even desirable when using BP methods. Given that

$$\prod_{i=1}^{k} p(X_i|X_{i+1}, X_{i+2}, \ldots, X_k) = p(X_1, X_2, \ldots, X_k), \tag{6.10}$$

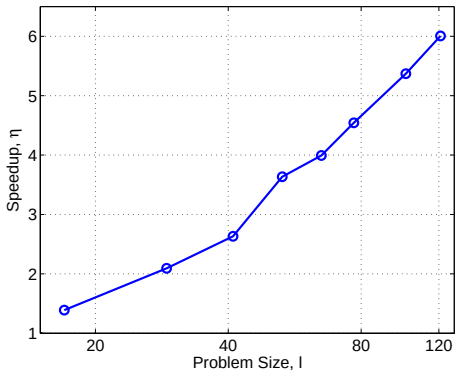
if a factor node relating k interacting variables stores the joint probability $p(X_1, X_2, \ldots, X_k)$, there is no need of having k factors, one for each conditional probability in the product above. In this case, the presence of k factors can be even prejudicial if the lower-order statistics are misleading, which is the case for deceptive problems. The factors corresponding to lower-order statistics will make judgments based on local/deceptive information, somehow discrediting the information sent by the factor nodes with $k-$order statistics. Only when lower-order statistics start guiding the search towards 11111 (as mentioned above), this configuration will get enough recommendations to set the MPC with the optimal substructure.

Loopy SLS for overlapping trap problems

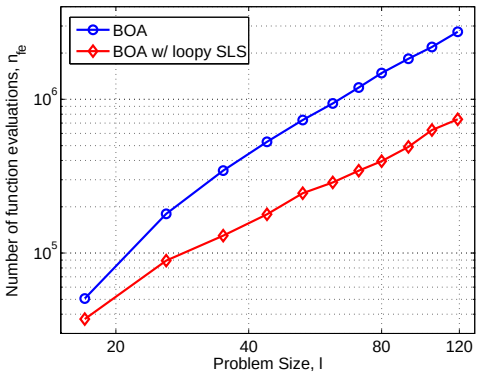
Figure 6.8 presents the results for the trap-5 problem with several degrees of overlapping ($o = \{1, 2, 3\}$). By using loopy substructural local search the savings in function evaluations are much better than those obtained by the previous local searcher (see Figure 6.3). For 1-variable overlap ($o = 1$), the speedup grows up to 6, behaving very similarly to the non-overlapping case. For 2-variable overlap ($o = 2$), the speedup also increases with the problem size but with a more moderate slope. Finally, for 3-variable overlap, the speedup grows with even a more moderate slope, while for larger problem instances the speedup seems to stagnate. Notice that for a trap subfunction with $k = 5$ and $o = 3$, three out of five variables (60%) are shared with each of the two neighboring subfunctions, and each subfunction overlaps with another four on at least one variable. This translates into a considerable amount of



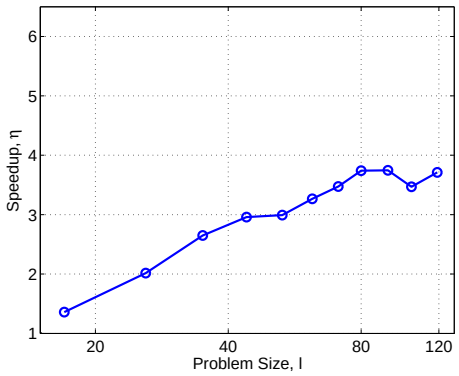
(a) Num. of function evaluations, $o = 1$



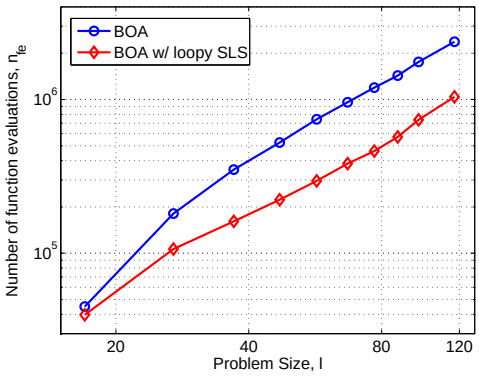
(b) Speedup, $o = 1$



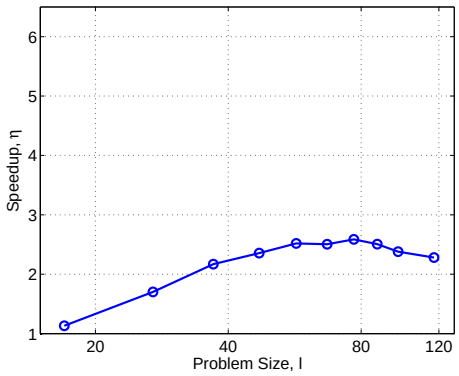
(c) Num. of function evaluations, $o = 2$



(d) Speedup, $o = 2$



(e) Num. of function evaluations, $o = 3$



(f) Speedup, $o = 3$

Figure 6.8: Number of function evaluations (left) and corresponding speedup (right) for BOA with loopy SLS, when solving the overlapping trap-5 problem. Variable overlapping between subproblems of $o = 1$ (top), $o = 2$ (middle), and $o = 3$ (bottom) are considered.

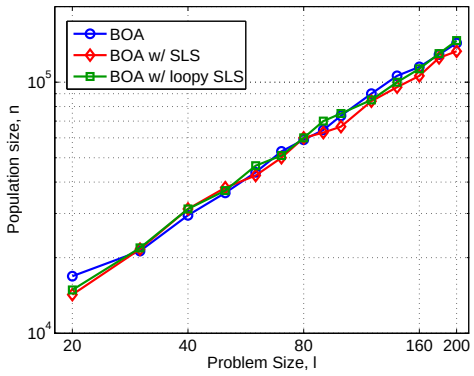
noise at the decision-making for each subproblem, when looking for the best subsolution. Therefore, it is expected that loopy SLS decreases its efficiency for higher values and occurrence of overlapping.

It is interesting to note that this behavior is similar to that obtained with local search in eCGA for noisy problems. For increasing amounts of noise, the performance of SLS reduces while the recombination capabilities of eCGA excel. The same tradeoff seems to happen here. For problems with highly overlapping subfunctions, PLS which simulates recombination is expected to be the main responsible component for efficiently generating good solutions. Notice that standard BOA deals quite well with overlapping problems, as the number of required evaluations is barely affected with increased overlap. Nevertheless, the speedups obtained with loopy SLS for problems with overlap are still substantial for considerable proportions of overlap.

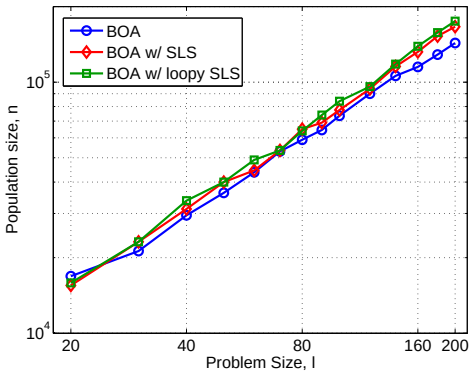
As mentioned before, when applying belief propagation to graphs with loops there is no guarantee of convergence to a fixed point. Therefore, increasing the number of loops (by increasing the proportion of overlap) will affect the chances of converging to high-quality solutions with loopy SLS. Looking in detail at the loopy belief propagation algorithm, it can be seen that for highly overlapping trap-5 problems ($o \geq 3$) the number of ties becomes larger at initial generations delaying the ability to find good solutions from the start. This makes BOA with loopy SLS require more generations to solve the problem.

Loopy SLS for non-overlapping trap problems

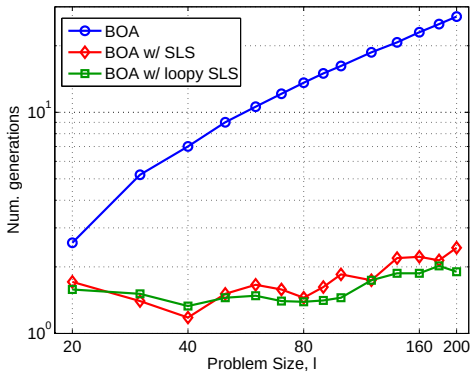
Apart from being able to tackle overlapping problems, it is important that the loopy substructural local searcher can deal with the non-overlapping case and deliver the same speedups obtained for the previous substructural local searcher. To address



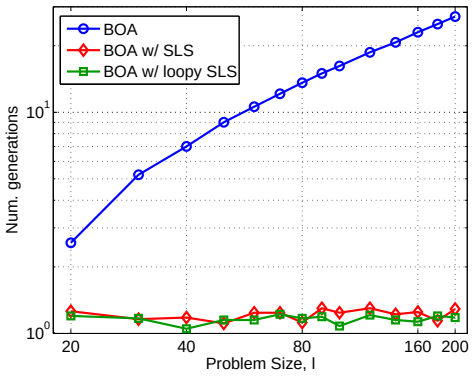
(a) Population Size, n_0



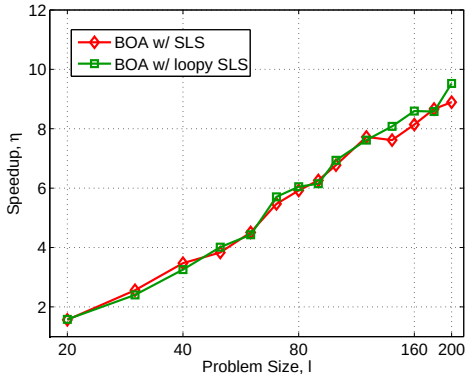
(b) Population Size, n_1



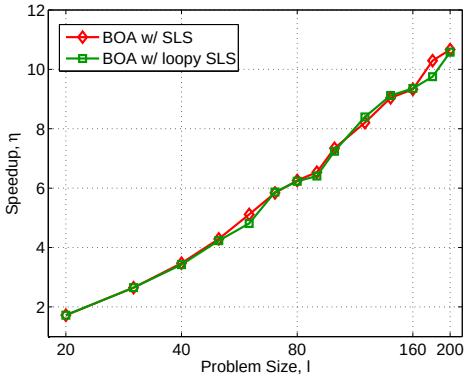
(c) Num. of generations, n_0



(d) Num. of generations, n_1



(e) Speedup, n_0



(f) Speedup, n_1

Figure 6.9: Results for BOA with loopy SLS when solving the non-overlapping trap-5 problem. These results are compared with the previous substructural local searcher. The results from the left side are obtained with the traditional bisection method (n_0), while for the right side the modified bisection (n_1) is used.

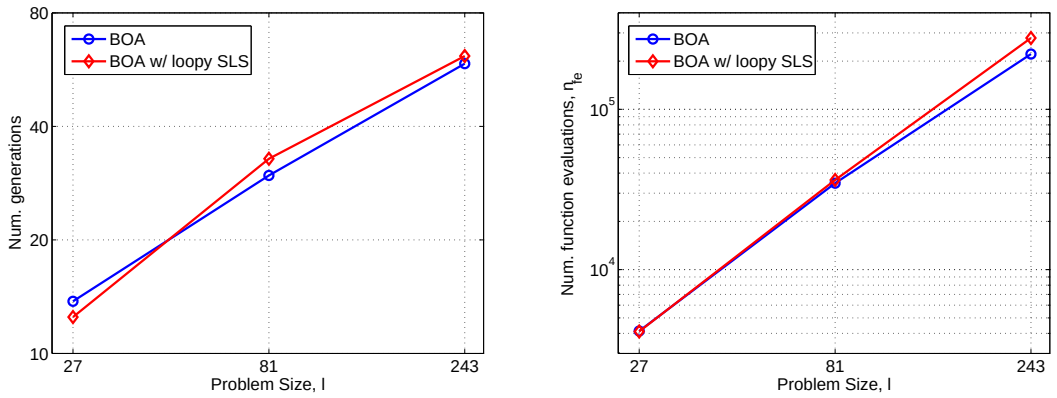
this issue, experiments for BOA with loopy SLS on the non-overlapping trap-5 problem are performed. The corresponding results are plotted in Figure 6.9. Results obtained through both the traditional bisection method and the refined method are shown and compared to the results obtained for the previous SLS version. For both population-sizing methods, the loopy SLS obtains similar speedups to the previous local searcher. Also, both versions of local search slightly improve with a marginally larger population size (n_1), particularly for larger instances of the problem. However, the impact of the population size on performance is much greater when addressing the overlapping case.

6.6 Hierarchical Difficulty

The hierarchical difficulty is a relevant feature that should be addressed to model complex real-world problems. As previously mentioned, hierarchical problems contain interactions that are present at more than a single level or context. The test problem considered is the hierarchical trap-3 function. Figure 6.10 shows the results obtained for loopy substructural local search.

For the three instances of the problem tested (with 3,4, and 5 hierarchical levels), the incorporation of substructural local search does not seem to bring any advantage over the traditional BOA. Both versions of the algorithm perform similar for this particular problem. The population size and number of generations taken by the two approaches is similar, and consequently the number of function evaluations spent is about the same.

Because there is no way of deciding whether 000 is better than 111 for any subproblem on all levels except for the top one, the local searcher can not speedup the identification of optimal subsolutions. For this problem, the preservation of different



(a) Num. of generations (b) Num. of function evaluations

Figure 6.10: Number of generations and function evaluations required for BOA with loopy SLS for solving three different instances of the hierarchical trap-3 problem.

alternatives and consequent recombination between them, lead to the discovery of optimal settings for the upper level, until the top level is reached and a decision can be made based on the fitness signal between the string of ones and zeros. Essentially, the problem needs to be solved iteratively because the information gathered by the probabilistic model about the underlying problem structure is not enough to accelerate the decision-making between competing optima. In addition, the problem is highly multimodal at the lower levels, which reduces the impact of local search.

An interesting local search approach to tackle hierarchical problems have been recently proposed [80]. The method consists of using self organizing maps (SOM) [90] to map variables into substructural partitions which are then explored locally. For each module at a given level, the most two significant subsolutions are mapped into a single variable, while the remaining $2^k - 2$ subsolutions are discarded, therefore reducing the dimensionality of the problem. This method is able to solve hierarchical problems quite efficiently using incremental model learning, while avoiding large samples of solutions to infer about the problem structure.

Although this approach is efficient for hierarchical problems, it relies on a num-

ber of assumptions that are not appropriate for broadly applicable optimization techniques. The first limitation of the method is that it relies on a map of fixed size (grid of 5×5 nodes), which can not learn an adequate complexity by itself. This is somewhat equivalent of using BOA with a predefined number of incoming edges. Another drawback of this approach is the assumption that the problem at each level only has two relevant subsolutions. Furthermore, encapsulating several variables into a single one raises the question of how do deal with overlapping interactions.

Clearly, the method ends up incorporating specific knowledge about the problem itself, therefore cannot be considered as a black-box solver. This approach contrasts with the work presented in the thesis, as the methods proposed herein have in mind a more general range of application. Nevertheless, the methodology proposed in [80] is interesting and very much related to the ideas presented in this thesis.

6.7 Discussion

Considering the behavior of BOA with loopy SLS on boundedly difficult problems with non-overlapping, overlapping, and hierarchical interactions, distinct dynamics can be observed for each case. For the non-overlapping trap problem, substructural local search is shown to substantially reduce the number of function evaluations, providing speedups superior to 10 for a problem size of $\ell = 200$. This translates into one order of magnitude less evaluations to solve the same problem. More importantly, the speedup consistently increases with problem size approximately as $\Theta(\sqrt{\ell})$.

For the overlapping trap problem, BOA with loopy SLS maintains the substantial speedups from the non-overlapping case, but as the dimension of overlapping increases (making the problem more noisy), its efficiency is reduced. Nevertheless,

local search still succeeds in saving a significant number of function evaluations when compared to standard BOA. Speedups of 6, 3.75, and 2.5 were obtained for proportions of overlap of 20%, 40%, and 60%, respectively.

When solving the hierarchical trap problem, the local search approach has not been successful in reducing the number of evaluations. The reason for SLS not providing any benefits is related to the amount of information about the problem captured by the Bayesian network. While interactions at the base level are captured and exploited by local search, further interactions from superior levels can only be detected by careful preservation and combination of subsolutions from the lower levels. Therefore, the number of generations required by recombination seem to be necessary to *mine* the crucial structure of the problem.

Essentially, in order to take full advantage from substructural local search, crucial information about the problem should be *minable* from the set of solutions in the initial generations. In some cases, particularly for the overlapping trap problem, the choice of the population size can have an important role in capturing the necessary information to perform SLS with success. While a smaller population size can still solve the problem through iterative recombination of subsolutions, a larger population can excel the SLS capabilities to quickly solve the problem. Once more, this goes back to the topic of adaptive time continuation discussed previously in Section 3.6. The incorporation of substructural local search in BOA implicitly allows to switch between a global and a local search operator based on problem features, being the population size responsible for "choosing" one over the other.

6.8 Summary

This chapter starts by applying the developments from the previous chapter to improve the efficiency of substructural local search in BOA. The $s-$penalty is shown to improve significantly the savings obtained with local search, allowing the corresponding speedup to scale approximately as $\Theta(\sqrt{\ell})$ for larger problem instances. The presence of overlapping interactions as a new source of problem difficulty is investigated while performing SLS. Although the effect of overlapping variable interactions is similar to that of exogenous noise, which is known to be extremely hard for local search, a new SLS method based on loopy belief propagation is proposed. The loopy SLS is shown to efficiently solve problems with both overlapping and non-overlaping interactions. The utility of such local searcher is also discussed for problems with hierarchical interactions, where it fails to improve the results obtained for standard BOA. The chapter ends by recognizing that integrating loopy SLS in BOA enables the resulting algorithm to have the best of both worlds—efficient mutation and recombination—for problems with non-ovelapping, overlapping, and hierarchical interactions.

Chapter 7

Future Work and Conclusions

This final chapter highlights a number of extensions that deserve further exploration and presents the major conclusions from this dissertation.

7.1 Future Work

Following the work presented in this thesis, a number of topics deserve further investigation:

Application to real-world problems. While the artificial problems used in the thesis cover different types of variables interactions and problem difficulty dimensions, it is important to investigate the utility of SLS in real-world problems. The main motivation for designing adversarial problems with bounded difficulty is to achieve competence when solving real-world problems that are bounded in difficulty by this kind of functions. Therefore, experiments on real-world problems should be performed to confirm the proposed approach as a robust real-world solver.

Automatic population-sizing. The incorporation of SLS in both eCGA and

BOA implicitly allows to switch between a global and local search operator based on problem requirements. However, such behavior is mediated by the population size. While SLS requires more accurate substructural information to quickly solve a problem, standard EDAs through iterative recombination of solutions can afford less accurate models and consequently smaller population sizes. One possible approach is to adapt online the population size using model information and theoretical population-sizing models [176, 177].

Consider other fitness estimators. The key idea behind fitness modeling with Bayesian networks (and other multivariate EDAs) is to use the model structure to induce the functional form of the surrogates and combine with techniques for estimating the coefficients of the induced surrogates. The thesis considered a method based on the schema theorem for fitting coefficients of the surrogate function; however, other more robust methods from system identification, estimation, and regression, can be used to estimate the same coefficients. A first step in this direction has already been made with promising results [153].

Extension for multi-objective optimization. While this work considers single-objective problems, many interesting problems have multiple conflicting objectives. The goal in multi-objective optimization is to find the solutions that form the Pareto-optimal front (solutions which are not dominated in at least one objective). Consequently, it would be interesting to investigate if SLS can help EDAs to converge to the optimal front. However, probabilistic modeling for multi-objective optimization purposes is not straightforward, as the problem decomposition might differ for different objectives. One possible solution is to build different models for each objective and generate solutions by considering all models [173]. In this case, substructural neighborhoods from different

models can be combined to perform SLS.

7.2 Conclusions

This thesis investigates substructural local search for discrete estimation of distribution algorithms. Particularly, the extended compact genetic algorithm and the Bayesian optimization algorithm are considered. For both algorithms it is shown that incorporating SLS is advantageous for several types of boundedly-difficult problems. Empirical results demonstrate that SLS can substantially reduce the total number of function evaluations required to solve such problems, providing speedups superior to 10. This translates into one order of magnitude less evaluations to solve the same problem. More importantly, the speedups obtained scale with problem size as $O(\sqrt{\ell})$.

Looking at the behavior of both eCGA and BOA when incorporating SLS, distinct dynamics can be observed for different problems. In eCGA, using SLS is shown to be beneficial for deterministic and low-noise problems with uniformly-scaled BBs. For problems with moderate-to-high amounts of exogenous noise or with exponentially-scaled BBs, SLS in eCGA can not improve the search for optimal solutions, performing similar to the standard eCGA which is the more adequate approach. On the other hand, BOA with SLS considerably improves the results obtained for problems with both non-overlapping and overlapping interactions. For increasing overlapping between subfunctions (becoming the problem more noisy), SLS reduces its efficiency, but still succeeds in saving a significant number of evaluations compared to standard BOA. For hierarchical problems, BOA with and without SLS perform similarly, indicating that SLS brings no additional value for solving these type of problems.

Essentially, in order to take full advantage from SLS, relevant information about the problem should be *minable* from the set of solutions in the initial generations. The choice of the population size have an important role in capturing the necessary information to perform SLS with success. While a smaller population size can still solve the problem through iterative recombination of subsolutions, a larger population can excel the SLS capabilities to quickly solve the problem. This tradeoff can be seen as a type of adaptive time continuation. The incorporation of substructural local search in eCGA and BOA implicitly allows to switch between a global and a local search operator based on problem features. This adaptive behavior is however mediated by the population size.

Different ways of incorporating substructural local search in EDAs are also proposed. While for eCGA only the best individual is considered for local search, in BOA several individuals are obtained after performing SLS. This difference is directly related to the cost of performing local search, which is significantly reduced when using a surrogate fitness model instead of actual function evaluations. The more complex nature of the probabilistic models in BOA, and corresponding neighborhoods, also motivate the incorporation of several solutions obtained through local search. Nevertheless, eCGA and BOA can use both approaches. Therefore, the choice of incorporating local search results into the population (or into the probabilistic model) should depend on the particular approach. For certain optimization problems where diversity in the population is extremely important, the strategy for incorporating SLS should take that fact into account.

While the analysis of model structural accuracy in BOA was not one of the initial objectives for the thesis, this study has been crucial to improve SLS in BOA. Furthermore, this work takes a step towards understanding and interpreting the Bayesian networks in BOA, providing more interpretable models to assist efficiency-

enhancement techniques and human researchers. The study identifies both the selection operator and the scoring metric (that guides model search) as the main factors[1] to influence model quality in Bayesian EDAs. These results demystifies previous empirical comparisons between different Bayesian EDAs, which mainly differ on the choice of the selection operator and scoring metric.

More generally, the importance of exploiting model-based information in EDAs to further improve the search has been demonstrated with SLS. The study of model quality in BOA will be also important for situations where the knowledge of the problem structure can be as valuable as a high-quality solution to the problem. This is the case for several other model-based efficiency enhancement techniques [154, 132, 153, 146, 97, 176, 174, 72, 69], or simply for the offline interpretation of probabilistic models [173, 174] to gain insight on the problem.

[1]Given a sufficient population size (learning data set).

Bibliography

[1] Ackley, D. H. *A connectionist machine for genetic hill climbing*. Kluwer Academic, Boston, 1987.

[2] Ahn, C. W., and Ramakrishna, R. S. On the scalability of the real-coded Bayesian optimization algorithm. *IEEE Transactions on Evolutionary Computation 12*, 3 (2008), 307–322.

[3] Arnold, D. V. *Noisy Optimization with Evolution Strategies*. Kluwer Academic Publishers, Norwell, MA, 2002.

[4] Bäck, T. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence* (1994), pp. 57–62.

[5] Bäck, T. Generalized convergence models for tournament—and (μ, λ)—selection. In *Proceedings of the 6th International Conference on Genetic Algorithms* (San Francisco, CA, USA, 1995), Morgan Kaufmann, pp. 2–8.

[6] Bäck, T. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[7] Balakrishnan, N., and Nevzorov, V. B. *A Primer on Statistical Distributions*. Wiley, 2003.

[8] Baluja, S. 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.

[9] Baluja, S., and Davies, S. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proceedings of the 14'th International Conference on Machine Learning* (1997), Morgan Kaufman, pp. 30–38.

[10] Barnes, J. W., Dimova, B., and Dokov, S. P. The theory of elementary landscapes. *Applied Mathematical Letters 16* (2003), 337–343.

[11] Bayati, M., Shah, D., and Sharma, M. Max-product for maximum weight matching: Convergence, correctness, and LP duality. *Information Theory, IEEE Transactions on 54*, 3 (March 2008), 1241–1251.

[12] Bertsekas, D. P. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1995.

[13] Beyer, H.-G. Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation 3*, 3 (1996), 361–394.

[14] Blickle, T., and Thiele, L. A comparison of selection schemes used in genetic algorithms. *Evolutionary Computation 4*, 4 (1997), 311–347.

[15] Blum, C., and Dorigo, M. Search bias in ant colony optimization: on the role of competition-balanced systems. *Evolutionary Computation, IEEE Transactions on 9*, 2 (2005), 159–174.

[16] Bosman, P. A. N. *Design and Application of Iterated Density-Estimation Evolutionary Algorithms*. PhD thesis, University of Utrecht, The Netherlands, 2003.

[17] Braunstein, A., Mezard, M., and Zecchina, R. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms 27*, 2 (2005), 201–226.

[18] Brindle, A. *Genetic Algorithms for Function Optimization*. PhD thesis, University of Alberta, Edmonton, Canada, 1981. Unpublished doctoral dissertation.

[19] Cantú-Paz, E. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis 10*, 2 (1998), 141–171.

[20] Cantú-Paz, E. Migration policies and takeover times in parallel genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference* (San Francisco, CA, 1999), Morgan Kaufmann, p. 775.

[21] Cantú-Paz, E. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[22] Chen, Y.-P. *Extending the scalability of linkage learning genetic algorithms: Theory and practice*. Springer, 2005.

[23] Chen, Y.-P., Yu, T.-L., Sastry, K., and Goldberg, D. E. A survey of linkage learning techniques in genetic and evolutionary algorithms. IlliGAL Report No. 2007014, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2007.

[24] Chickering, D. M., Geiger, D., and Heckerman, D. Learning Bayesian Networks is NP-Hard. Technical Report MSR-TR-94-17, Microsoft Research, Redmond, WA, 1994.

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

[26] Cooper, G. F., and Herskovits, E. H. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning 9* (1992), 309–347.

[27] Correa, E. S., and Shapiro, J. L. Model complexity vs. performance in the Bayesian optimization algorithm. In *PPSN IX: Parallel Problem Solving from Nature, LNCS 4193* (2006), T. P. Runarsson et al., Eds., Springer, pp. 998–1007.

[28] De Bonet, J. S., Isbell, C. L., and Viola, P. MIMIC: Finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems* (1997), M. C. Mozer, M. I. Jordan, and T. Petsche, Eds., vol. 9, The MIT Press, Cambridge, p. 424.

[29] De Jong, K. A. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.

[30] Deb, K. *Optimization for Engineering Design: Algorithms and Examples.* Prentice-Hall of India, Privated Limited, New Delhi, India, 1995.

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

[32] Dorigo, M. *Optimization, Learning and Natural Algorithms (in Italian).* PhD thesis, Politecnico di Milano, Milan, Italy, 1992.

[33] Dorigo, M., and Caro, G. D. The ant colony optimization meta-heuristic. In *New ideas in optimization.* McGraw-Hill, 1999, pp. 11–32.

[34] Dorigo, M., Maniezzo, V., and Colorni, A. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B 26* (1996), 29–41.

[35] Echegoyen, C., Lozano, J. A., Santana, R., and Larrañaga, P. Exact bayesian network learning in estimation of distribution algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2007), IEEE Press, pp. 1051–1058.

[36] Elidan, G., Mcgraw, I., and Koller, D. Residual belief propagation: Informed scheduling for asynchronous message passing. In *Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI)* (2006).

[37] Etxeberria, R., and Larrañaga, P. Global optimization using Bayesian networks. In *Second Symposium on Artificial Intelligence (CIMAF-99)* (Habana, Cuba, 1999), A. A. O. Rodriguez et al., Eds., pp. 332–339.

[38] Feige, U., Mossel, E., and Vilenchik, D. Complete convergence of message passing algorithms for some satisfiability problems. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, LNCS 4110*. Springer, 2006, pp. 339–350.

[39] Fernandes, C., Lima, C. F., and Rosa, A. UMDAs for dynamic optimization problems. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO-2008)* (2008), C. Ryan et al., Eds., ACM Press, pp. 399–406.

[40] Fernandes, C., Rosa, A., and Ramos, V. Binary ant algorithm. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO-2007)* (2007), D. Thierens et al., Eds., ACM Press, pp. 41–48.

[41] Fitzpatrick, J. M., Grefenstette, J. J., and Gucht, D. V. Image registration by genetic search. In *Proceedings of the IEEE Southeast Conference* (1984), pp. 460–464.

[42] Friedman, N., and Goldszmidt, M. Learning Bayesian networks with local structure. *Graphical Models* (1999), 421–459.

[43] Glover, F. Future paths for integer programming and links to artificial intelligence. In *Computers and Operations Research* (1986), pp. 533–549.

[44] Glover, F. Tabu Search: Part I. In *ORSA Journal of Computing* (1989), vol. 1, pp. 190–206.

[45] Glover, F. Tabu Search: Part II. In *ORSA Journal of Computing* (1990), vol. 2, pp. 4–32.

[46] Goldberg, D. E. Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems 3* (1989), 153–171.

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

[48] Goldberg, D. E. Sizing populations for serial and parallel genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms* (San Mateo, CA, 1989), J. D. Schaffer, Ed., Morgan Kaufman, pp. 70–79. (Also TCGA Report 88004).

[49] Goldberg, D. E. The race, the hurdle, and the sweet spot. In *Evolutionary Design by Computers*, P. J. Bentley, Ed. Morgan Kaufmann, San Francisco, CA, 1999, pp. 105–118.

[50] Goldberg, D. E. Using time efficiently: Genetic-evolutionary algorithms and the continuation problem. In *Proceedings of the Genetic and Evolutionary Computation Conference* (San Francisco, CA, 1999), Morgan Kaufmann, pp. 212–219.

[51] Goldberg, D. E. *The Design of Innovation - Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, 2002.

[52] Goldberg, D. E., and Deb, K. A comparative analysis of selection schemes used in genetic algorithms. *Proceedings of the First Workshop on Foundations of Genetic Algorithms 1* (1991), 69–93.

[53] Goldberg, D. E., Deb, K., and Clark, J. H. Genetic algorithms, noise, and the sizing of populations. *Complex Systems 6* (1992), 333–362.

[54] Goldberg, D. E., Deb, K., Kargupta, H., and Harik, G. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms* (San Mateo, CA, 1993), S. Forrest, Ed., Morgan Kaufmann, pp. 56–64.

[55] Goldberg, D. E., Deb, K., and Thierens, D. Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers 32*, 1 (1993), 10–16.

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

[57] Goldberg, D. E., and Rudnick, M. Genetic algorithms and the variance of fitness. *Complex Systems 5*, 3 (1991), 265–278.

[58] Goldberg, D. E., and Sastry, K. A practical schema theorem for genetic algorithm design and tuning. In *Proceedings of the Genetic and Evolutionary Computation Conference* (San Francisco, CA, 2001), Morgan Kaufmann, pp. 328–335.

[59] Goldberg, D. E., Sastry, K., and Latoza, T. On the supply of building blocks. In *Proceedings of the Genetic and Evolutionary Computation Conference* (San Francisco, CA, 2001), Morgan Kaufmann, pp. 336–342.

[60] Handa, H. The effectiveness of mutation operation in the case of estimation of distribution algorithms. *Journal of Biosystems 87*, 2-3 (2007).

[61] Hansen, N., and Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation 9*, 2 (2001), 159–195.

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

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

[64] Harik, G. R. Finding multimodal solutions using restricted tournament selection. *Proceedings of the Sixth International Conference on Genetic Algorithms* (1995), 24–31.

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

[66] Harik, G. R., and Goldberg, D. E. Learning linkage. *Foundations of Genetic Algorithms 4* (1997), 247–262. Also IlliGAL Report No. 96006.

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

[68] Hart, W. E. *Adaptive global optimization with local search.* PhD thesis, University of California, San Diego, San Diego, CA, 1994.

[69] Hauschild, M., and Pelikan, M. Enhancing efficiency of hierarchical BOA via distance-based model restrictions. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature* (2008), Springer-Verlag, pp. 417–427.

[70] Hauschild, M., Pelikan, M., Lima, C. F., and Sastry, K. Analyzing probabilistic models in hierarchical BOA on traps and spin glasses. In *Proceedings of the ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2007)* (2007), D. Thierens et al., Eds., ACM Press, pp. 523–530.

[71] Hauschild, M., Pelikan, M., Lima, C. F., and Sastry, K. Analyzing probabilistic models in hierarchical BOA. *IEEE Transactions on Evolutionary Computation (accepted for publication)* (2008).

[72] Hauschild, M., Pelikan, M., Sastry, K., and Goldberg, D. E. Using previous models to bias structural learning in the hierarchical BOA. In *Proceedings of the ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2008)* (New York, NY, USA, 2008), ACM, pp. 415–422.

[73] Heckendorn, R. B., and Wright, A. H. Efficient linkage discovery by limited probing. *Evolutionary Computation 12*, 4 (2004), 517–545.

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

[75] Henrion, M. Propagation of uncertainty in Bayesian networks by logic sampling. In *Uncertainty in Artificial Intelligence*, J. F. Lemmer and L. N. Kanal, Eds. Elsevier, 1988, pp. 149–163.

[76] Holland, J. H. Genetic algorithms and the optimal allocation of trials. *SIAM Journal of Computing 2*, 2 (June 1973), 88–105.

[77] Holland, J. H. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[78] Höns, R. *Estimation of Distribution Algorithms and Minimum Relative Entropy.* PhD thesis, University of Bonn, Bonn, Germany, 2006.

[79] Höns, R., Santana, R., Larrañaga, P., and Lozano, J. A. Optimization by max-propagation using Kikuchi approximations. Tech. Rep. EHU-KAT-IK-2-07, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 2007.

[80] Iclănzan, D., and Dumitrescu, D. Large-scale optimization of non-separable building-block problems. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature* (2008), Springer-Verlag, pp. 899–908.

[81] Jakulin, A., and Bratko, I. Testing the significance of attribute interactions. In *Proceedings of the International Conference on Machine Learning (ICML-2004)* (2004), ACM Press, pp. 409–416.

[82] Japkowicz, N., and Stephen, S. The class imbalance problem: A systematic study. *Intelligent Data Analysis 6*, 5 (2002), 429–450.

[83] Jin, Y., and Branke, J. Evolutionary optimization in uncertain environments—A survey. *IEEE Transactions on Evolutionary Computation 9*, 3 (2005), 303–317.

[84] Johnson, A., and Shapiro, J. The importance of selection mechanisms in distribution estimation algorithms. In *Proceedings of the 5th European Conference on Artificial Evolution, LNCS Vol. 2310* (London, UK, 2001), Springer-Verlag, pp. 91–103.

[85] Juels, A., Baluja, S., and Sinclair, A. The equilibrium genetic algorithm and the role of crossover, 1993. Unpublished manuscript.

[86] Kargupta, H. The gene expression messy genetic algorithm. In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation* (Piscataway, NJ, 1996), IEEE Press, pp. 814–819.

[87] Kern, S., Müller, S. D., Hansen, N., Bche, D., and Koumoutsakos, P. Learning probability distributions in continuous evolutionary algorithms: A comparative review. *Natural Computing 3* (2004), 77–112.

[88] Kindermann, R., and Snell, J. L. *Markov Random Fields and Their Applications*. American Mathematics Society, Providence, RI, 1980.

[89] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983 220, 4598* (1983), 671–680.

[90] Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics 43* (1982), 59–69.

[91] Krasnogor, N. *Studies on the Theory and Design Space of Memetic Algorithms*. PhD thesis, University of the West of England, Bristol, England, 2002.

[92] Kschischang, F., Frey, B., and Loeliger, H.-A. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on 47*, 2 (Feb 2001), 498–519.

[93] Kubat, M., and Matwin, S. Addressing the curse of imbalanced training sets: one-sided selection. In *Proc. 14th International Conference on Machine Learning* (1997), Morgan Kaufmann, pp. 179–186.

[94] Kullback, S., and Leibler, R. A. On information and sufficiency. *Annals of Mathematical Statistics 22* (1951), 79–86.

[95] Larrañaga, P., and Lozano, J. A., Eds. *Estimation of distribution algorithms: a new tool for Evolutionary Computation*. Kluwer Academic Publishers, Boston, MA, 2002.

[96] Lima, C. F., , Goldberg, D. E., Pelikan, M., Lobo, F. G., Sastry, K., and Hauschild, M. Influence of selection and replacement strategies on linkage learning in BOA. In *IEEE Congress on Evolutionary Computation (CEC-2007)* (2007), K. C. Tan et al., Eds., IEEE Press, pp. 1083–1090.

[97] Lima, C. F., Fernandes, C., and Lobo, F. G. Investigating restricted tournament selection in ECGA for non-stationary environments. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO-2008)* (2008), C. Ryan et al., Eds., ACM Press, pp. 439–446.

[98] Lima, C. F., Lobo, F. G., and Pelikan, M. From mating pool distributions to model overfitting. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO-2008)* (2008), C. Ryan et al., Eds., ACM Press, pp. 431–438.

[99] Lima, C. F., Lobo, F. G., Pelikan, M., and Goldberg, D. E. Towards model accuracy in the Bayesian optimization algorithm (submitted), 2009.

[100] Lima, C. F., Lobo, F. G., Pelikan, M., Goldberg, D. E., and Sastry, K. Substructural local search for estimation of distribution algorithms (in preparation), 2009.

[101] Lima, C. F., Pelikan, M., Goldberg, D. E., Lobo, F. G., Sastry, K., and Hauschild, M. Linkage learning accuracy in the Bayesian optimization algorithm. In *Linkage in Evolutionary Computation*, Y.-P. Chen and M.-H. Lim, Eds. Springer, 2008, pp. 87–107.

[102] Lima, C. F., Pelikan, M., Lobo, F. G., and Goldberg, D. E. Loopy substructural local search for the Bayesian optimization algorithm (in preparation), 2009.

[103] Lima, C. F., Pelikan, M., Sastry, K., Butz, M., Goldberg, D. E., and Lobo, F. G. Substructural neighborhoods for local search in the Bayesian optimization algorithm. In *PPSN IX: Parallel Problem Solving from Nature, LNCS 4193* (2006), T. P. Runarsson et al., Eds., Springer, pp. 232–241.

[104] Lima, C. F., Sastry, K., Goldberg, D. E., and Lobo, F. G. Combining competent crossover and mutation operators: a probabilistic model building approach. In *Proceedings of the ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2005)* (2005), H. Beyer et al., Eds., ACM Press, pp. 735–742.

[105] Lobo, F. G., Goldberg, D. E., and Pelikan, M. Time complexity of genetic algorithms on exponentially scaled problems. In *Proceedings of the Genetic and Evolutionary Computation Conference* (San Francisco, CA, 2000), Morgan Kaufmann, pp. 151–158.

[106] Lourenço, H. R., Martin, O., and Stützle, T. A beginner's introduction to iterated local search. In *Proceedings of MIC'2001 – 4'th Metaheuristics International Conference* (Porto, Portugal, 2001), vol. 1, Kluwer Academic Publishers, pp. 1–6.

[107] Lourenço, H. R., Martin, O., and Stützle, T. Iterated local search. In *Handbook of Metaheuristics* (Norwell, MA, 2002), F. Glover and G. Kochenberger, Eds., Kluwer Academic Publishers, pp. 321–353.

[108] Lozano, J. A., Larrañaga, P., Inza, I., and Bengoetxea, E., Eds. *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*. Springer, Berlin, Germany, 2006.

[109] Marascuilo, L. A., and McSweeney, M. *Nonparametric and distribution-free methods for the social sciences*. Brooks/Cole Publishing Company, 1977.

[110] Mendiburu, A., Santana, R., and Lozano, J. A. Introducing belief propagation in estimation of distribution algorithms: A parallel approach. Tech. Rep. EHU-KAT-IK-11-07, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 2007.

[111] Mendiburu, A., Santana, R., Lozano, J. A., and Bengoetxea, E. A parallel framework for loopy belief propagation. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation* (New York, NY, USA, 2007), ACM, pp. 2843–2850.

[112] Miller, B. L., and Goldberg, D. E. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation 4*, 2 (1996), 113–131.

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

[114] Moscato, P. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Tech. Rep. C3P 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, CA, 1989.

[115] Mühlenbein, H. How genetic algorithms really work: I.Mutation and Hillclimbing. In *Parallel Problem Solving from Nature 2* (Amsterdam, The Netherlands, 1992), R. Männer and B. Manderick, Eds., Elsevier Science, pp. 15–25.

[116] Mühlenbein, H. Convergence of estimation of distribution algorithms for finite samples. *Unpublished Manuscript* (2008).

[117] Mühlenbein, H., and Höns, R. The factorized distributions and the minimum relative entropy principle. In *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*. Springer, 2006, pp. 11–38.

[118] Muhlenbein, H., Mahnig, T., and Rodriguez, A. O. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics 5* (1999), 215–247.

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

[120] Mühlenbein, H., and Paass, G. From recombination of genes to the estimation of distributions I. Binary parameters. In *Parallel Problem Solving from Nature – PPSN IV* (Berlin, 1996), H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds., Kluwer Academic Publishers, pp. 178–187.

[121] Mühlenbein, H., and Schlierkamp-Voosen, D. Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation 1*, 1 (1993), 25–49.

[122] Munetomo, M., and Goldberg, D. E. Linkage identification by nonmonotonicity detection for overlapping functions. *Evolutionary Computation 7*, 4 (1999), 377–398.

[123] Ochoa, A., Höns, R., Soto, M., and Mühlenbein, H. A maximum entropy approach to sampling in EDA: The single connected case. In *Progress in Pattern Recognition, Speech and Image Analysis, LNCS 2905*. Springer, 2003, pp. 683–690.

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

[125] Pelikan, M. *Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms*. Springer, 2005.

[126] Pelikan, M., and Goldberg, D. E. Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* (San Francisco, CA, 7-11 July 2001), L. Spector et al., Eds., Morgan Kaufmann, pp. 511–518.

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

[128] Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. Bayesian optimization algorithm, population sizing, and time to convergence. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)* (Las Vegas, Nevada, 10-12 July 2000), D. Whitley et al., Eds., Morgan Kaufmann, pp. 275–282.

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

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

[131] Pelikan, M., and Mühlenbein, H. The bivariate marginal distribution algorithm. In *Advances in Soft Computing – Engineering Design and Manufacturing*, R. Roy, T. Furuhashi, and P. K. Chawdhry, Eds. Springer–Verlag, London, 1999, pp. 531–535.

[132] Pelikan, M., and Sastry, K. Fitness inheritance in the bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Part II, LNCS 3103* (2004), K. Deb et al., Eds., Springer, pp. 48–59.

[133] Pelikan, M., Sastry, K., and Cantú-Paz, E., Eds. *Scalable Optimization via Probabilistic Modelling: From Algorithms to Applications*. Springer, 2006.

[134] Pelikan, M., Sastry, K., and Goldberg, D. E. Scalability of the Bayesian optimization algorithm. *International Journal of Approximate Reasoning 31*, 3 (2003), 221–258.

[135] Pereira, F. B., Machado, P., Costa, E., Cardoso, A., Ochoa-rodriguez, A., Santana, R., and Soto, M. Too busy to learn. In *Proceedings of the Congress on Evolutionary Computation (CEC-2000)* (Piscataway, NJ, 2000), IEEE Press, pp. 720–727.

[136] Pyle, D. *Data Preparation for Data Mining*. Morgan Kaufmann, San Francisco, CA, 1999.

[137] Quinlan, J. R. Combining instance-based and model-based learning. In *Proceedings of the Tenth International Conference on Machine Learning (ICML-93)* (1993), Morgan Kaufmann, pp. 236–243.

[138] Rechenberg, I. *Evolutionsstrategie Optimierung technischer systeme nach prinzipien der biologischen evolution*. Friedrich Frommann Verlag, Stuttgart-Bad Cannstatt, 1973.

[139] Reeves, C. R. Using genetic algorithms with small populations. In *Proceedings of the 5th International Conference on Genetic Algorithms* (San Francisco, CA, USA, 1993), Morgan Kaufmann, pp. 92–99.

[140] Rissanen, J. J. Modelling by shortest data description. *Automatica 14* (1978), 465–471.

[141] Robbins, H., and Monro, S. A stochastic approximation method. *Annals of Mathematical Statistics 22* (1951), 400–407.

[142] Rubinstein, R. Y. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability 1*, 2 (1999).

[143] Santana, R. *Advances in Graphical Models for Optimization and Learning: Applications in Protein Modeling*. PhD thesis, University of the Basque Country, San Sebastian, Spain, 2006.

[144] Santana, R., Larrañaga, P., and Lozano, J. A. Interactions and dependencies in estimation of distribution algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2005), IEEE Press, pp. 1418–1425.

[145] Sastry, K. Evaluation-relaxation schemes for genetic and evolutionary algorithms. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 2001.

[146] Sastry, K., Abbass, H. A., Goldberg, D. E., and Johnson, D. D. Sub-structural niching in estimation distribution algorithms. In *Proceedings of the ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2005)* (2005), H. Beyer et al., Eds., ACM Press.

[147] Sastry, K., and Goldberg, D. E. Analysis of mixing in genetic algorithms: A survey. IlliGAL Report No. 2002012, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2002.

[148] Sastry, K., and Goldberg, D. E. Scalability of selectorecombinative genetic algorithms for problems with tight linkage. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003), Part II, LNCS 2724* (2003), E. Cantu-Paz et al., Eds., Springer, pp. 1332–1344.

[149] Sastry, K., and Goldberg, D. E. Designing competent mutation operators via probabilistic model building of neighborhoods. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Part II, LNCS 3103* (2004), K. Deb and et al., Eds., Springer, pp. 114–125.

[150] Sastry, K., and Goldberg, D. E. Let's get ready to rumble: Crossover versus mutation head to head. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Part II, LNCS 3103* (2004), K. Deb and et al., Eds., Springer, pp. 126–137.

[151] Sastry, K., and Goldberg, D. E. Let's get ready to rumble redux: Crossover versus mutation head to head on exponentially scaled problems. In *Proceedings of the ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2007)* (New York, NY, USA, 2007), ACM, pp. 1380–1387.

[152] Sastry, K., Goldberg, D. E., and Pelikan, M. Don't evaluate, inherit. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* (San Francisco, CA, 2001), L. Spector et al., Eds., Morgan Kaufmann, pp. 551–558.

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

[154] Sastry, K., Pelikan, M., and Goldberg, D. E. Efficiency enhancement of genetic algorithms via building-block-wise fitness estimation. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (2004), pp. 720–727.

[155] Sastry, K., Pelikan, M., and Goldberg, D. E. Efficiency enhancement of estimation of distribution algorithms. In *Scalable optimization via Probabilistic Modeling: From Algorithms to Applications*, M. Pelikan, K. Sastry, and E. Cant-Paz, Eds. Springer, 2006, pp. 161–186.

[156] Schwarz, G. Estimating the dimension of a model. *The Annals of Statistics 6* (1978), 461–464.

[157] Schwefel, H.-P. Numerische optimierung von computer-modellen mittels der evolutionsstrategie. In *Interdisziplinäre Systemforschung*. Birkhäuser Verlag, Basel and Stuttgart, 1977, ch. 1, pp. 5–8.

[158] Shan, Y., McKay, R. I., Essam, D., and Abbass, H. A. A survey of probabilistic model building genetic programming. In *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, M. Pelikan, K. Sastry, and E. Cantu-Paz, Eds. Springer, 2006.

[159] Sinha, A. A survey of hybrid genetic and evolutionary algorithms. IlliGAL Report No. 2003004, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2003.

[160] Smith, R. E., Dike, B. A., and Stegmann, S. A. Fitness inheritance in genetic algorithms. In *SAC '95: Proceedings of the 1995 ACM symposium on Applied computing* (New York, NY, USA, 1995), ACM, pp. 345–350.

[161] Soto, M. R. *A Single Connected Factorized Distribution Algorithm and Its Cost of Evaluation.* PhD thesis, University of Havana, Havana, Cuba, 2003.

[162] Srivastava, R. Time continuation in genetic algorithms. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 2002.

[163] Thierens, D. Scalability problems of simple genetic algorithms. *Evolutionary Computation 7*, 1 (1999), 45–68.

[164] Thierens, D., and Goldberg, D. Convergence models of genetic algorithm selection schemes. In *Parallel Problem Solving from Nature, PPSN III* (Berlin, 1994), Y. Davidor, H.-P. Schwefel, and R. Manner, Eds., Springer-Verlag, pp. 116–121.

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

[166] Thierens, D., Goldberg, D. E., and Pereira, A. G. Domino convergence, drift, and the temporal-salience structure of problems. In *Proceedings of 1998 IEEE International Conference on Evolutionary Computation* (Picataway, NJ, 1998), IEEE Press, pp. 535–540.

[167] Van Laarhoven, P. J. M., and Aarts, E. H. L. *Simulated Annealing: Theory and Applications.* D. Reidel Publishing Company, Dordrecht, Holland, 1987.

[168] Watson, J.-P. *Empirical modeling and analysis of local search algorithms for the job-shop scheduling problem.* PhD thesis, Colorado State University, Fort Collins, CO, 2003.

[169] Watson, R. A., Hornby, G., and Pollack, J. B. Modeling building-block interdependency. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature* (1998), Springer-Verlag, pp. 97–108.

[170] Weiss, G. M., and Provost, F. Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research 19* (2003).

[171] Wu, H., and Shapiro, J. L. Does overfitting affect performance in estimation of distribution algorithms. In *Proceedings of the ACM SIGEVO Genetic*

and Evolutionary Computation Conference (GECCO-2006) (2006), M. Keijzer et al., Eds., ACM Press, pp. 433–434.

[172] Yang, S. Constructing dynamic test environments for genetic algorithms based on problem difficulty. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (2004), IEEE Press, pp. 1262–1269.

[173] Yu, T.-L. *A Matrix Approach for Finding Extrema: Problems with Modularity, Hierarchy, and Overlap*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 2007.

[174] Yu, T.-L., and Goldberg, D. E. Dependency structure matrix analysis: Offline utility of the dependency structure matrix genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Part II, LNCS 3103* (2004), K. Deb et al., Eds., Springer, pp. 355–366.

[175] Yu, T.-L., Goldberg, D. E., Yassine, A., and Chen, Y.-P. A genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm. In *Proceedings of the Artificial Neural Networks in Engineering 2003 (ANNIE 2003)* (2003), pp. 327–332.

[176] Yu, T.-L., Sastry, K., and Goldberg, D. E. Population size to go: Online adaptation using noise and substructural measurements. In *Parameter Setting in Evolutionary Algorithms*, F. G. Lobo et al., Eds. Springer, 2007, pp. 205–224.

[177] Yu, T.-L., Sastry, K., Goldberg, D. E., and Pelikan, M. Population sizing for entropy-based model building in genetic algorithms. In *Proceedings of the ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2007)* (2007), D. Thierens et al., Eds., ACM Press, pp. 601–608.

[178] Zlochin, M., Birattari, M., Meuleau, N., and Dorigo, M. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research 131* (2004), 373–395.