BRNO UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

Parallel Estimation of Distribution Algorithms

Doctoral Thesis

Author: Ing. Jiří Očenášek

Supervisor: Ing. Josef Schwarz, CSc.
Submited: November 20, 2002
State Doctoral Exam: June 15, 2001

The thesis is available in the library of the Faculty of Information Technology of the Brno University of Technology.

Abstract

The thesis deals with the new evolutionary paradigm based on the concept of Estimation of Distribution Algorithms (EDAs) that use probabilistic model of promising solutions found so far to obtain new candidate solutions of optimized problem.

There are six primary goals of this thesis:

1. Suggestion of a new formal description of EDA algorithm. This high level concept can be used to compare the generality of various probabilistic models by comparing the properties of underlying mappings. Also, some convergence issues are discussed and theoretical ways for further improvements are proposed.

2. Development of new probabilistic model and methods capable of dealing with continuous parameters. The resulting Mixed Bayesian Optimization Algorithm (MBOA) uses a set of decision trees to express the probability model. Its main advantage against the mostly used IDEA and EGNA approach is its backward compatibility with discrete domains, so it is uniquely capable of learning linkage between mixed continuous-discrete genes. MBOA handles the discretization of continuous parameters as an integral part of the learning process, which outperforms the histogram-based approach. The original metrics for MBOA is derived as the incremental version of Bayesian Dirichlet metrics (BDe). Its usefulness for modelling of Boolean functions is also investigated and confirmed.

3. Parallelization of EDA algorithms. Different versions of parallel EDA algorithms are proposed for fine-grained, coarse-grained and multithreaded environment. All these algorithms use the original principle of restricted ordering of nodes in Bayesian network to minimize the communication between parallel processes:

- *The pipelined Parallel Bayesian Optimization algorithm (PBOA) is proposed and simulated for fine-grained type of parallelism. The target platform for its implementation can be Transputers, or one-purpose MIMD processor.*
- *Distributed Bayesian optimization algorithm (DBOA) is proposed and implemented for coarse-grained type of parallelism. The experiments are running on the uniform cluster of Sun Ultra 5 workstations connected by fast Ethernet switch. Both PBOA and DBOA are based on the parallelization of classical Pelikan's BOA code.*
- *The multithreaded MBOA algorithm, which uses original MBOA code with mixed decision trees model, was simulated in Transim tool.*

In all parallel algorithms the additional progress towards realtime performance can be achieved by choosing the KBOA version of BOA algorithm. The concept of KBOA is based on the partial (local) information about the linkage. This information is used to adjust the prior of probabilistic model and injection of building blocks is used to improve the quality of initial population.

4. Extension of single-objective EDA to multi-objective EDA. The experiments show that the realized Pareto-strength BOA algorithm, which utilizes a promising Pareto-strength niching technique, outperforms the classical constraint- or weighting-based approach and is comparable to best recombination-based multiobjective-algorithms of its time - SPEA and NSGA. A completely new optimization technique for MBOA core was designed in collaboration with the author of SPEA2. This new selection operator based on epsilon-archives leads to implementation of Bayesian Multi-objective Optimization Algorithm (BMOA).

5. Design of the modular development system DEBOA for rapid prototyping of optimization applications on the basis of BOA algorithm. The general requirements on the development system are identified including modularity and visualization of results. The design of the development system itself is realized on the platform of Java language, which ensures great portability. DEBOA system can be easily extended to support new types of fitness functions as well as new types of visualizations.

6. Testing of the developed algorithms on well-known benchmarks as well as several real-world problems, mainly from the area of circuit design.

Key Words: *Bayesian network, Gaussian network, Estimation of Distribution Algorithm, Factorized Distribution Algorithm, Bayesian Optimization Algorithm, Univariate Marginal Distribution Algorithm, Bivariate Marginal Distribution Algorithm, Bayesian-Dirichlet metric, scoring metric, dependency structure, machine learning, linkage learning, decision graph, kernel distribution, mixture model, formal description, building block corruption, convergence, hypergraph bisectioning, knapsack problem, additively decomposable function, multiobjective optimization, Pareto set, niching, Pareto-strength fitness, epsilon dominance, pipelined processing, distributed processing, multithreaded processing, scalability, rapid prototyping of BOA application.*

Acknowledgement

First and foremost I thank to my supervisor Josef Schwarz who has been a permanent source of stimulation and encouragement throughout my research. His advices and constructive criticism help me in successful completion of my research work. I am also grateful to Martin Pelikan from Illinois Genetic Algorithms Laboratory for inspiration and productive discussions, his pioneering work in the EDA area influenced me a lot. I also highly appreciate a cheerful collaboration in multicriterial optimization and personal friendship with Marco Laumanns from ETH Zurich. A special thanks to my wife Zdenka, my parents and the rest of my closest family for their understanding and support. Last but not least I would like to thank to my colleagues from the Faculty of Information Technology (FIT VUT Brno) I had the pleasure of working with. I would also like to thank my dissertation committee members for agreeing to serve on my committee and giving me valuable comments.

My research has been carried out under the financial support of the Czech Ministry of Education (research grant FRVS 0171/2001 “The parallelization of Bayesian Optimization Algorithm” and research intention CEZ: J22/98: 262200012 - "Research in information and control systems") and the Grant Agency of Czech Republic (research grant GA 102/02/0503 "Parallel system performance prediction and tuning").

Contents

List of Abbreviations

AI	Artificial Intelligence.
BB	Building block.
BDe	Bayesian-Dirichlet metrics.
BGe	Bayesian metrics for Gaussian networks having score equivalence.
BIC	Bayesian Information Criteria.
BMDA	Bivariate Marginal Distribution Algorithm.
BMOA	Bayesian Multiobjective Optimization Algorithm.
BOA	Bayesian Optimization Algorithm.
CART	Classification And Regression Trees.
cGA	Compact Genetic Algorithm.
CPD	Conditional probability distribution.
EBNA	Estimation of Bayesian Networks Algorithm.
ECGA	Extended Compact Genetic Algorithm.
EDA	Estimation of Distribution Algorithm.
EGNA	Estimation of Gaussian Networks Algorithm.
EP	Evolutionary Programming.
ES	Evolution Strategies.
FDA	Factorized Distribution Algorithm.
GA	Genetic Algorithm.
HC	Hill climber.
IDEA	Iterated Density Estimation Algorithm.
LFDA	Learning Factorized Distribution Algorithm.
MBOA	Mixed Bayesian Optimization Algorithm
mBOA	Multiobjective Bayesian Optimization Algorithm
MIMIC	Mutual-Information-Maximizing Input Clustering.
PBIL	Population-Based Incremental Learning.
PBOA	Parallel Bayesian Optimization Algorithm.
PDF	Probability density function.
PLS	Probabilistic Logic Sampling.
PMBGA	Probabilistic Model-Building Genetic Algorithm.
UMDA	Univariate Marginal Distribution Algorithm.

1 Introduction

In natural evolution, populations of individuals compete to survive and reproduce. Relatively fit individuals survive longer and thus reproduce more. The fitter examples of random variations accumulate and average fitness tends to increase with time. According to the Darwinian theory, this process of natural selection is responsible for the development of all life forms on Earth.

A common way of pursuing this approach for computational purposes involves use of the crossover-based genetic algorithms [41]. In this approach, candidate solutions are represented as strings of characters and reproduction involves the production of new individuals through the pairwise recombination operators. This produces two offspring strings which then replace relatively unfit individuals from the population. According to the building-block hypothesis and schema theorem of Holland [50] GA is an efficient search method. However, empirical work has shown that in some cases the efficiency is hard to guarantee because there is a deep contradiction between the building-block hypothesis and the schema theorem.

To avoid the disruption of building blocks a new Estimation of Distribution Algorithms (EDAs) have been recently proposed. These evolutionary optimization algorithms incorporate methods for automated learning of linkage between variables of the encoded solutions. The process of sampling new individuals from a probabilistic model respects these mutual dependencies among the string variables such that disruption of important building blocks is avoided.

My research focused on development of new types of EDA algorithms with various probabilistic models. The following three initial chapters are dedicated to the brief introduction to investigated areas – genetic algorithms (Chapter 2), probabilistic models (Chapter 3) and optimization via probabilistic models (Chapter 4). Different disadvantages of classical genetic algorithms are highlighted and the utilization of probabilistic models in evolutionary computation is justified. In Chapter 5 the state-of-the-art is analyzed and the main open problems that remain to be solved are identified. The first open problem is the lack of formal basis in present EDA papers. This formal approach together with the other theoretical foundations was proposed in Chapter 6. Chapter 7 defines the benchmark instances used in the experimental parts of chapters 8-10. It contains both the adopted well-established benchmarks as well as my new mixed continuous-discrete benchmark. The second open problem lies in the existence of application areas where present EDA algorithms are unapplicable or ineffective:

- continuous optimization
- realtime optimization
- multi-objective optimization

Consequently, the Chapter 8 proposes new type of EDA based on the idea of CART model (Classification And Regression Trees), which is applicable also for problems with continuous or even mixed parameters. In Chapter 9 the optimization of large problems is enabled by parallel processing – different versions of EDA algorithms are proposed for fine-grained, coarse-grained and multithreaded environment. Also the usage of heuristics for software acceleration is discussed. Chapter 10 proposes new multiobjective-capable EDA algorithms: first version which combines the classical EDA together with the Pareto-strength approach and second one which incorporated the reliable epsilon-dominance concept into new CART-based EDA.

Finally, Chapter 11 deals with the design and implementation of fast prototyping tool, which enables fast development of EDA applications using Java language.

Chapters 2, 3, 4 contain a review of existing sources, whereas chapters 5, 6, 8, 9 and 10 present an original outcome of my research (the way in which their sub-chapters are composed is further explained in 5.2). Chapter 11 is an extension of MSc. thesis I supervised.

2 Classical genetic algorithms

Genetic algorithms (GA), first proposed by Holland in 1975 [50], are a class of computational techniques that mimic natural evolution to solve problems in a wide variety of domains. Generally, the optimization techniques can be divided into three broad classes:

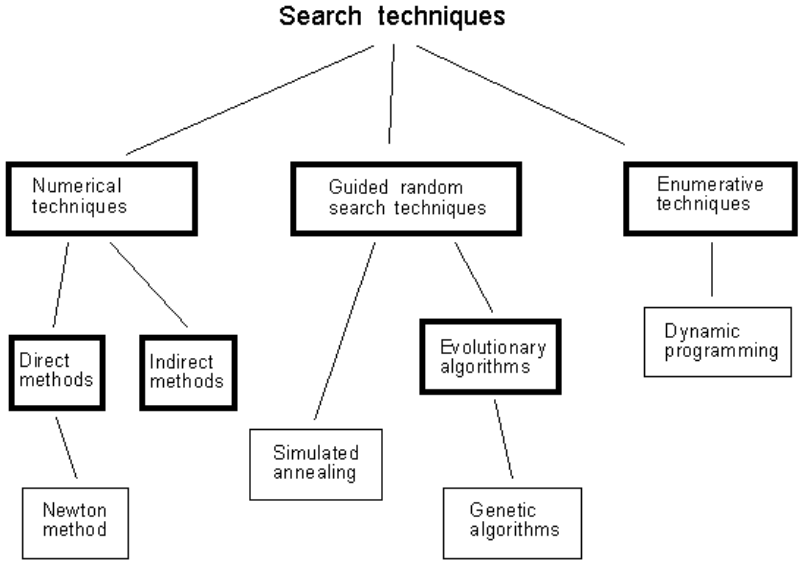


Fig. 2.1*: Simplified classification of search techniques*

Calculus based techniques are only suitable for a restricted set of well-behaved systems. The set of possible solutions can be enormous, which rules out the enumerative techniques too. Only the guided random search techniques, for example genetic algorithms, provide robust and well scalable approach applicable even to black-box optimization. They are able to use additional information to guide the search, which reduces the search space to manageable sizes.

2.1 Principles

Genetic algorithms (GA) are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome like data structure and apply recombination operators so as to preserve critical information. Genetic algorithms are often viewed as function optimizer, although the range of problems is quite broad. GA is any population based model that uses selection and recombination operators to generate new sample point in a search space.

The reproduction process is performed in such a way that only the chromosomes which represent a better solution are given more chances to reproduce than those chromosomes which are poorer solutions. The goodness/fitness is defined in the frame of the population.

The terms *string / individual* are also used as synonyms for *chromozome*.

2.1.1 Evolutionary cycle

The genetic algorithm manipulates the most promising strings in the search space to obtain an improved solution. It operates through a simple cycle:

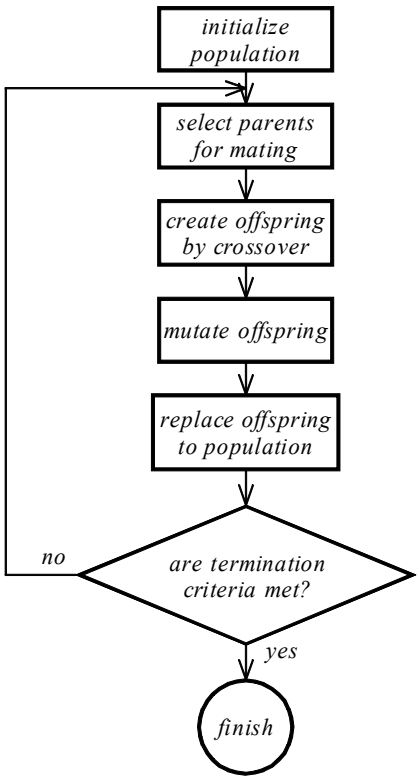


Fig. 2.2: Flowchart of classical genetic algorithm

Each cycle produces a new generation of possible solutions (individuals) for a given problem. At the beginning, a population of possible solutions is created. Each individual in this population is represented as a string (the chromozome) to be manipulated by the genetic operators. Traditionally, binary encodings have being used because they are easy to implement. But many optimization problems use continuous parameters. A common technique for encoding them in the binary form uses a fixed-point integer encoding, each parameter being coded using a fixed number of bits. The binary code of all the parameters can then be concatenated in the strings of the population. A drawback of encoding parameters as binary strings is the presence of Hamming barrier (large Hamming distances between the codes of some adjacent integers), which can be solved by usage of Gray code.

In the next stage, the individuals are decoded and evaluated according to their performance in relation to the target response. This determines how fit this individual is in relation to the others in the population. Based on each individual's fitness, a selection mechanism chooses the best pairs for the genetic manipulation process. The selection policy is responsible to assure the survival of the fittest individuals. Mainly the truncation selection method will be used in this thesis. It selects the best $1/s$ part of the current population, where s determines selection pressure. To select a population of the same size as the population before selection is, s copies of selected solutions can be created.

Crossover is one of the genetic operators used to recombine the population genetic material. It takes two chromosomes and swaps part of their genetic information to produce new chromosomes. This operation is similar to sexual reproduction in nature. In the usual approach, parent genotypes are split at a certain point, forming a left part and a right part. The right part from one parent is then joined to left part from other, and vice versa.
This is the most frequently used one-point crossover. However, a number of other operators have been introduced since the basic model was proposed. They are usually better adapted to constraints of a particular problem.

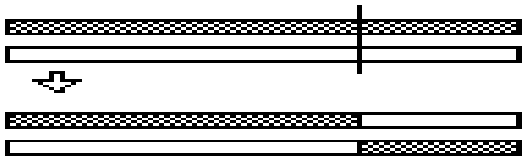


Fig. 2.3*: Example of one-point crossover*

The recombination process alone cannot explore search space sections not represented in the population's genetic structures. This could make the search get stuck around local minima. Here mutation goes into action. The mutation operator introduces new genetic structures in the population by randomly changing some of its genes, helping the algorithm escape local minima traps. Since the modification is totally random and thus not related to any previous genetic structures present in the population, it creates different structures related to other sections of the search space.

After applying crossover and mutation to the set of promising solutions, the population of new candidate solutions replaces the original one and the next iteration is executed, unless the termination criteria are met. For example, the run can be terminated when the population converges to a single solution, or the population contains a good enough solution, or a maximum number of iterations has been reached.

2.1.2 Problem dependent issues

When designed properly, the recombination operator assures that the most fit genetic structures, called building blocks, are retained for future generations. For critical review of this assumption see chapter 2.3.

Also, the key issue to the algorithm performance is the choice of underlying encoding for the solution of the optimization problem (the individual in the population). Namely the ordering of parameters can affect the mutation and cross-over operations significantly.

The lack of methodologies dealing with proper choice of encoding and recombination operator for given problem is one of the strongest disadvantages of classical GAs. My thesis research focused on Estimation of Distribution Algorithms, where the recombination operators are replaced by more advanced and parameter-less statistical techniques, that are even robust to ordering of parameters/genes.

2.2 Formal description

Theoretical computer science operates with formally defined objects. Several formalisms have been introduced in the field of GAs (see [2], [114]). This chapter in an extraction of Back's formal definition of evolutionary algorithms presented in [2]. It serves as a basis for my own formal description of Estimation of Distribution Algorithms introduced in 6.1.

Definition 2.1 (General Evolutionary Algorithm) :
A general evolutionary algorithm is defined as an 8-tuple

$$\mathrm{EA} = (I, \Phi, \Omega, \Psi, s, \iota, \mu, \lambda), \tag{2.1}$$

where:
(i) I is the space of individuals (a set of chromosomes) ,
(ii) $\Phi : I \to \mathbb{R}$ denotes a fitness function assigning real values to individuals,
(iii) μ is the number of parent individuals, while λ denotes the number of offspring individuals,
(iv) $\Omega = \{\omega_{\Theta_1}, ..., \omega_{\Theta_z} \mid \omega_{\Theta_i} : I^\lambda \to I^\lambda\} \cup \{\omega_{\Theta_0} : I^\mu \to I^\lambda\}$ is a set of probabilistic genetic operators ω_{Θ_i} each of which is controlled by specific parameters summarized in the sets $\Theta_i \subset \mathbb{R}$,
(v) $s_{\Theta_s} : (I^\lambda \cup I^{\mu+\lambda}) \to I^\mu$ denotes the selection operator, which may change the number of individuals from λ or $\lambda+\mu$ to μ, where $\mu, \lambda \in \mathbb{N}$ and $\mu = \lambda$ is permitted. An additional set Θ_s of parameters may be used by the selection operator.
(vi) $\Psi : I^\mu \to I^\mu$ is the generation transition function which describes the complete process of transformation of current population P into subsequent one by applying genetic operators and selection:

$$\Psi = s \circ \omega_{\Theta_{i_1}} \circ ... \circ \omega_{\Theta_{i_j}} \circ \omega_{\Theta_0}$$

$$\Psi(P) = s_{\Theta_s}(Q \cup \omega_{\Theta_{i_1}}(...(\omega_{\Theta_{i_j}}(\omega_{\Theta_0}(P))...)),$$

where $\{i_1, ..., i_j\} \subseteq \{1, ..., z\}$ and $Q \in \{\varnothing, P\}$
(vii) $\iota : I^\mu \to \{true, false\}$ is the termination criterion.

Definition 2.1 is based on high-level description where the population of individuals is manipulated by genetic operators and undergoes a fitness-based selection process. This is captured in the generation transition function Ψ, iterated application of which generates a population sequence and leads to definitions of the running time and the result of EA (see appropriate definitions in [2]).

Once the definition of the general EA and its computation is available, we can immediately establish formal definitions of genetic algorithm:

Definition 2.2 (Genetic Algorithm) :

$\mathrm{EA} = (I, \Phi, \Omega, \Psi, s, \iota, \mu, \lambda)$

is called Genetic Algorithm iff

(i) $I = \{0,1\}^n$,

(ii) $\forall \boldsymbol{x} \in I : \Phi(\boldsymbol{x}) = \delta(F(\Upsilon(\boldsymbol{x})), \Theta_\delta)$, where $\delta : \mathbb{R} \times \Theta_\delta \to \mathbb{R}^+$ denotes a scaling function, F denotes the optimization problem and Υ is a decoding function

(iii) $\Omega = \{m_{\{p_m\}} : I^N \to I^N, r_{\{p_c,z\}} : I^N \to I^N, r_{\{p_c\}} : I^N \to I^N\}$, where $m_{\{p_m\}}$ is a mutation with a probability p_m, $r_{\{p_c,z\}}$ is a z-point crossover with probability p_c and $r_{\{p_c\}}$ is an uniform crossover with probability p_c.

(iv) $\Psi(P) = s(m_{\{p_m\}}(r_{\{p_c,z\}}(P)))$,

(v) $s : I^N \to I^N$ is the proportional selection operator, which selects individuals from population $P(t) = \{\boldsymbol{x}^0(t),...,\boldsymbol{x}^{N-1}(t)\} = \{(x_0^0(t),...,x_{n-1}^0(t)),...,(x_0^{N-1}(t),...,x_{n-1}^{N-1}(t))\}$ according to the selection probability density function p_s:

$$p_s(\boldsymbol{x}^i(t)) = \frac{\Phi(\boldsymbol{x}^i(t))}{\sum_{j=0}^{N-1}\Phi(\boldsymbol{x}^j(t))} \tag{2.2}$$

(vi) ι is the termination criterion defined as

$$\iota(P(t)) = \begin{cases} true, & if\ t > t_{\max} \\ false, & otherwise \end{cases} \tag{2.3}$$

or as

$$\iota(P(t)) = \begin{cases} true, & if\ b(P(t)) > b_{\max} \\ false, & otherwise \end{cases}, \tag{2.4}$$

where the bias $b(P(t))$ of population is calculated as

$$b(P(t)) = \frac{1}{n.N}\sum_{j=0}^{n-1}\max\left\{\sum_{i=0}^{N-1}(1-x_j^i(t)), \sum_{i=0}^{N-1}x_j^i(t)\right\}. \tag{2.5}$$

(vii) $\lambda = \mu = N$

2.3 Disadvantages of classical recombination

Holland's schema analysis [50] provides an explanation how the classical GA explore the space of possible genotypes. In this analysis we assume that the GA is a way of processing genotype features rather then genotypes themselves - a feature being simply a set of values in specific positions.

2.3.1 Schema growth under pure reproduction

A particular feature is defined in terms of a schema. This is a genotype-like string with specific values in some positions and `don't care' values (asterisks) in others. An example is:

*10**0****

This schema has ten characters in all, including seven "don't care" values. It will match any 10-character genotype with a 1 in the second position, a 0 in the third position and a 0 in the sixth position. The genotypes, which match the schema, are referred to as its instances.

Note that we can construct a schema, which will match some specific genotype by replacing any one of its characters with asterisks. There are thus 2^n schemas matching to any genotype of length n. There are potentially $N*2^n$ schemas for a population of N genotypes of length n, although in practice there will usually be fewer due to duplication. (There are always at least 2^n.) The *defining length* of a schema δ is the number of positions between its first and last specific position. Its *order* o is the number of specified bits. Thus the defining length of "*10**0****" is 5 and its order is 3.

Fitness value of a schema is defined as the average fitness of its instances. We can think of GA as carrying out "schema processing". The goal of the GA is the multiplication of highly fit schemas in the population. As Holland [50] has shown, this goal is achieved by ordinary reproduction (that is, genotype-copying) without the need for crossover. Under this scenario, the evolutionary process involves repeatedly selecting an individual and making its copy. Provided that genotypes are selected with a probability proportional to their fitness, the growth formula for schemas in a pure-reproduction scenario is [41]

$$m(H,t+1) = m(H,t) \cdot \frac{F(H)}{\overline{F}}, \tag{2.6}$$

where H is a schema, $m(H,t)$ is the number of instances of H in the population at time t, $F(H)$ is the fitness of H and $\overline{F}$ is the average fitness.

If we assume $F(H) = (1+c).\overline{F}$, we can write $m(H,t)$ as

$$m(H,t) = m(H,0).(1+c)^t. \tag{2.7}$$

We see that under pure reproduction, schemas can be expected to grow exponentially if the average fitness of the schema is higher than the average fitness of the population. Unfit schemas gradually get squeezed out of the population because their instances tend not to be reproduced.

2.3.2 Schema growth under reproduction with crossover

If the original population does not contain a good solution then the pure reproduction method obviously cannot succeed in finding it. Some source of novelty is required. The reproduction regime is typically modified to use crossover. This ensures that novel genotypes can be generated. Unfortunately, the crossover method involves cutting parental genotypes into two parts and thus runs the risk of slicing up and destroying useful schemas.

We therefore need to determine the extent to which the destructive properties of the crossover operator undermine schema processing. In the schema analysis, this is dealt with by introducing a schema conservation probability to the growth formula. The probability of schema destruction depends on the defining length of the schema and the probability p_c of crossover itself. Longer schemas have a higher probability of being destroyed under crossover. The probability of destruction is thus the ratio of defining length to genotype length.(In Goldberg's book - it is the ratio of defining length to genotype length - 1) and the probability of conservation is the complement of this:

$$m(H,t+1) \geq m(H,t) \cdot \frac{F(H)}{\overline{F}} \cdot \left[1 - p_c \cdot \frac{\delta(H)}{n-1} \right] \tag{2.8}$$

Accounting for the destructive effects of a mutation operation require further change to the formula (see [41]).

The destructive properties of the crossover operation are assumed to be negligible with schemas of short defining length since, in this case, the bracketed part of the growth formula evaluates to a value close to 1. This leads directly to the so-called schema theorem, which states that "short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations".

2.3.3 The shortness and low-epistasis assumptions

According to the schema theorem, schemas will only be correctly preserved if they are of short defining length. In fact a schema will only be preserved correctly if its fitness advantage (the excess of its fitness over the average fitness) is greater than its vulnerability - the ratio of its defining length to the genotype length.

The schema theorem also covertly introduces an assumption concerning schema fitness. It explicitly refers to "above-average schemata" and is thus implicitly assuming that schemas have an independent fitness value. However, in the crossover scenario, instances do change from generation to generation. Thus the fitness value of a particular schema may vary. A schema which is assumed to have a constant fitness value under crossover has an independent impact on fitness since its fitness value is not affected by its genetic context. Conversely, a schema fitness value which does change under crossover is affected by context. The technical term for the situation where schema fitness is affected by context is *epistasis*. The schema theorem assumes that schema fitness is not affected by context. It thus introduces the assumption of low epistasis.

It can be shown that for typical fitness assignment and for typical genotype length both the shortness and low-epistasis assumptions are easily violated.

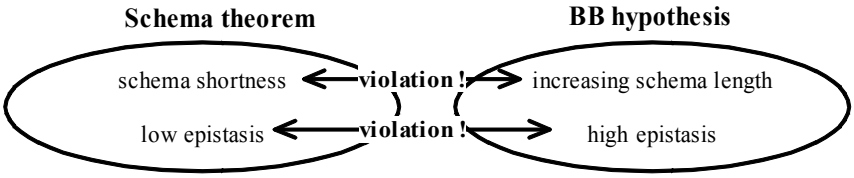


Fig. 2.4*: Schema theorem assumptions violate building block hypothesis assumptions*

2.3.4 The building blocks hypothesis

The credibility of the crossover GA does not rest solely on the schema theorem. It also rests on the so-called *building-block hypothesis*. This states that the crossover GA works well when short, low-order, highly fit schemas recombine to form even more highly fit, higher-order schemas. The ability to produce fitter and fitter partial solutions by combining blocks is believed to be the primary source of the GA's search power. Unfortunately, when we examine the assumptions introduced by the building-block hypothesis, we find that they contradict those introduced by the schema theorem.

The building-block hypothesis assumes that the fitness of any block is typically affected by the other blocks on the genotype. If this were not the case it would be meaningless to talk about a building-block process. Thus the building-block hypothesis implicitly assumes only a positive effect of epistasis on fitness and thus contradicts the low-epistasis assumption introduced by the schema theorem.

Considering the length implications of the building-block hypothesis we uncover a further contradiction. During the building-block process, the schemas that require processing at any given stage are actually the blocks that have been put together by the prior building-block process. Except at the initial stage, the defining length of these schemas is related to the defining lengths of the components of the blocks. If we make the conservative assumption that, on average, the defining lengths of blocks will be at least the sum of the defining lengths of the components, then we see that real schema length will increase at least exponentially during the building-block process. We see that the building-block hypothesis (which depends on negligible schema length) is very likely to be violated in all but the initial stage of processing. It demands increasing schema length and thus violates the shortness assumption introduced by the schema theorem.

The scenario, in which fitness is affected by epistatic interactions between parts of the genotype has, of course, been intensively investigated by the GA community. The construction of the so-called *deceptive problem* is typically a matter of deliberately introducing epistasis ("nonlinearity") in an encoding (see 7.4).

2.3.5 Alternatives for schema processing

There are two major approaches to resolve the problem of building-block disruption. The first approach is based on manipulating the representation of solutions in the algorithm in order to make the interacting components of partial solutions less likely to be broken by recombination operators. Various reordering and mapping operators were used. However, reordering operators are often too slow and lose the race against selection, resulting in premature convergence to low-quality solutions. Reordering is not sufficiently powerful in order to ensure a proper mixing of partial solutions before these are lost. This line of research has resulted in algorithms which evolve the representation of a problem among individual solutions, e.g. the Messy Genetic Algorithm (mGA, [42]), the Gene Expression Messy Genetic Algorithm (GEMGA, [55]), the Linkage Learning Genetic Algorithm (LLGA, [44]), or the Linkage Identification by Non-linearity Checking Genetic Algorithm (LINC-GA, [84]).

A different way to cope with the disruption of partial solutions is to change the basic principle of recombination. In the second approach, instead of implicit reproduction of important building blocks and their mixing by selection and two-parent recombination operators, new solutions are generated by using the information extracted from the entire set of promising solutions. These Estimation of Distribution Algorithms (for introduction see chapter 4) are subject of my research. They are insensitive to ordering of genes, like messy GAs, but they allow also for solving functions with higher degree of epistasis.

3 Probabilistic models

Probabilistic inference has become a core technology in AI, largely due to developments in graph-theoretic methods for the representation and manipulation of complex probability distributions. Even more importantly, the graph-theoretic framework has allowed for the development of Estimation of Distribution Algorithms, which in many cases provide orders of magnitude speedups over brute-force search methods and better robustness than classical evolutionary methods.

3.1 Notation

Consider a set $\boldsymbol{X}=\{X_0,\ldots, X_{n-1}\}$ of n discrete random variables where each variable X_i may take on values from a finite domain $\mathcal{A}$. Capital letters such as X_1, X_2, X_3 denote variable names, and lowercase letters such as x_1, x_2, x_3 denote specific values taken by those variables. If not all the variables X_i are defined over the same domain $\mathcal{A}$, then the set of values X_i can attain is denoted as $Dom(X_i)$ and the cardinality of this set is denoted as $||X_i|| = |Dom(X_i)|$. Sets of variables are denoted by boldface capital letters such as $\boldsymbol{X}$ and assignments of values to the variables in these sets are denoted by boldface lowercase letters such as $\mathbf{x}$ (I use $Dom(\boldsymbol{X})$ and $||\boldsymbol{X}||$ in the obvious way). The term $p(X_i,X_j,X_k)$ denotes the joint probability of discrete random variables X_i,X_j,X_k whereas $p(X_i|X_j,X_k)$ denotes the conditional probability of X_i given X_j,X_k. Note that the terms $p(\boldsymbol{X})$, $p(X_i,X_j,X_k)$ and $p(X_i|X_j,X_k)$ denote probability distributions, but $p(\boldsymbol{x})$, $p(x_i,x_j,x_k)$ and $p(x_i|x_j,x_k)$ denote one probability value. An alternative notation for $p(x_i)$ is also $p(X_i=x_i)$.

Boldface symbol $\boldsymbol{\Pi}_i$ denotes the set of random variables which completely determine the probability distribution of variable X_i, such that X_i is independent of the other variables from $\boldsymbol{X} \setminus \boldsymbol{\Pi}_i$ given the variables from $\boldsymbol{\Pi}_i$. Lowercase boldface symbol $\boldsymbol{\pi}_i$ denotes one concrete configuration from $Dom(\boldsymbol{\Pi}_i)$ assigned to variables in $\boldsymbol{\Pi}_i$.

Now consider a data set $D=\{\boldsymbol{x}^0,\boldsymbol{x}^1,\ldots,\boldsymbol{x}^{N-1}\}$ containing N samples from $p(\boldsymbol{X})$ distribution. I use superscript numbers to denote the index of concrete sample vector within the data set. The counts of some patterns in the data set are denoted m. For example $m(x_i,x_j,x_k)$ denotes the number of sample vectors in the data set matching with the concrete values of i-th, j-th and k-th coordinate, whereas $m(X_i,X_j,X_k)$ denotes the count table of size $||X_i||.||X_j||.||X_k||$.

Throughout this thesis some chapters deal with probabilistic models over continuous or mixed domains. For these purposes I use analogical notation as in the discrete case, but I prefer the letter $\boldsymbol{Y}$ to denote set of continuous random variables $\{Y_0,\ldots, Y_{n-1}\}$. Thus, for example $f(\boldsymbol{Y})$ denotes the joint probability density function whereas $p(\boldsymbol{X})$ denotes joint probability distribution. Analogically $f(Y_i,Y_j,Y_k)$ denotes the joint probability density function and $f(Y_i|Y_j,Y_k)$ denotes the conditional probability density function. Lowercase letters such as y_1, y_2, y_3 denote specific values taken by variables, so for example $f(y_i|y_j,y_k)$ denotes one concrete value of the local conditional probability density.

Note that during this thesis I use 2 as the implicit base for logarithms, if not specified otherwise.

Note that the probability density functions have following properties:

$$\iiint f(y_i, y_j, y_k)\, dy_i\, dy_j\, dy_k = 1 \tag{3.1}$$

$$f(y_j, y_k) = \int f(y_i, y_j, y_k)\, dy_i \tag{3.2}$$

$$f(y_i \mid y_j, y_k) = \frac{f(y_i, y_j, y_k)}{f(y_j, y_k)} = \frac{f(y_i, y_j, y_k)}{\int f(y_i, y_j, y_k)\, dy_i} \tag{3.3}$$

$$\forall (y_j, y_k) : \int f(y_i \mid y_j, y_k)\, dy_i = 1 \tag{3.4}$$

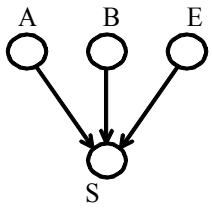
D is used in the superscript in $p_D(\boldsymbol{X})$ resp. $f_D(\boldsymbol{y})$ to indicate that the probability distribution resp. probability density function refers to the distribution of the samples within the dataset D.

3.2 Introduction

3.2.1 Bayesian network (BN)

Bayesian networks are graphical representations of probability distributions. These networks provide a compact and natural representation, effective inference, and efficient learning. They have been successfully applied in expert systems, diagnostic engines, and optimal decision making systems.

A *Bayesian network* consists of two components. The first is a directed acyclic graph (DAG) in which each vertex corresponds to a random variable. This graph describes *conditional independence* properties of the represented distribution. It captures the structure of the probability distribution, and is exploited for efficient inference and decision making. Thus, while Bayesian networks can represent arbitrary probability distributions, they provide computational advantage to those distributions that can be represented with a sparse DAG. The second component is a collection of *conditional probability distributions* (CPDs) that describe the conditional probability of each variable given its parents (ancestors) in the graph. Together, these two components represent a unique probability distribution.



A B E	p(S|A,B,E)
1 1 1	0.95
1 1 0	0.95
1 0 1	0.20
1 0 0	0.05
0 1 1	0.00
0 1 0	0.00
0 0 1	0.00
0 0 0	0.00

Fig. 3.1: A simple Bayesian network structure and the associated CPD for variable S (showing the probability values for S=1)

Definition 3.1 (Conditional independence) :
Let $p(\boldsymbol{X})$ be a joint probability distribution over the variables in $\boldsymbol{X}$ and let $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$ be the subsets of $\boldsymbol{X}$. $\boldsymbol{U}$ and $\boldsymbol{V}$ are *conditionally independent*, given $\boldsymbol{W}$, if for all $\boldsymbol{u} \in Dom(\boldsymbol{U})$, $\boldsymbol{v} \in Dom(\boldsymbol{V})$ and $\boldsymbol{w} \in Dom(\boldsymbol{W})$, we have that $p(\boldsymbol{u}|\boldsymbol{v},\boldsymbol{w}) = p(\boldsymbol{u}|\boldsymbol{w})$ whenever $p(\boldsymbol{v},\boldsymbol{w}) > 0$.

Definition 3.2 (Bayesian network) :
More formally, a Bayesian network (BN) for $\boldsymbol{X}$ is the pair $B = (\mathcal{G}, \mathcal{L})$. $\mathcal{G}$ is a DAG whose nodes correspond to the random variables $X_0, \ldots, X_{n-1}$, and whose edges encode the following set of independence statements: each variable X_i is independent of its nondescendants $\boldsymbol{X} \setminus \boldsymbol{\Pi}_i$, given its parents $\boldsymbol{\Pi}_i$ in $\mathcal{G}$. The set composed of a variable and its parents is usually referred to as a *family*. The second component $\mathcal{L}$ specifies the conditional probability $p(X_i|\boldsymbol{\Pi}_i)$ for all variables X_i.

Any distribution $p(\boldsymbol{X})$ that satisfies the independence statements encoded in the graph $\mathcal{G}$ can be *factorized* as

$$p(X_0,\ldots,X_{n-1}) = \prod_{i=0}^{n-1} p(X_i \mid \boldsymbol{\Pi}_i)\,. \tag{3.5}$$

3.2.2 Bayesian network with local structure

In its most naive form, a CPD is encoded by means of a tabular representation that is locally exponential in the number of parents of a variable X_i: for each possible assignment of values to the parents of X_i, we need to specify a distribution over the values X can take. For example, consider the simple network in Fig. 3.1, where the variables *A, B, E* and *S* correspond to the events "alarm armed", "burglary", "earthquake", and "loud alarm sound" respectively. Assuming that all variables are binary, a tabular representation of the CPD for *S* requires eight parameters, one for each possible state of the parents. One possible quantification of this CPD is given in Fig. 3.1. Note, however, that when the alarm is not armed (i.e., when $A = 0$) the probability of $S = 1$ is zero, regardless of the values *B* and *E*. Thus, the interaction between *S* and its parents is simpler than the eight-way situation that is assumed in the tabular representation of the CPD.

The locally exponential size of the tabular representation of the CPDs is a major problem in learning Bayesian networks. It can be simplified using *context-specific* independence (CSI, see [34]). These independence statements imply that in some contexts, the conditional probability of variable *X* is independent of some of its parents. For example, in the network of Fig 3.1, when the the alarm is not set (i.e., the context defined by $A = 0$), the conditional probability does not depend on the value of *B* and *E*, *i.e.* $p(S \mid A=0, B=b, E=e)$ is the same for all values *b* and *e* of *B* and *E*. As we can see, CSI properties induce equality constraints among the conditional probabilities in the CPDs.

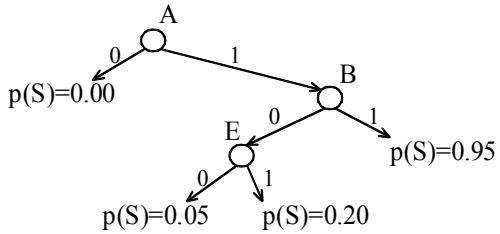


Fig. 3.2: Decision tree as an alternative representation of a local CPD structure

In Fig. 3.2 efficient CPD representation based on decision trees is described. Each leaf in the decision tree describes a probability for *S,* and the internal nodes and arcs encode the necessary information to decide how to choose among leaves, based on the values of S's parents. For example, in the tree the probability of $S = 1$ is 0 when $A = 0$, regardless of the state of *B* and *E;* and the probability of $S = 1$ is 0.95 when $A = 1$ and $B = 1$, regardless of the state of *E*. In this example, the decision tree requires four parameters instead of eight.

Incorporating local structure representations into the learning procedure leads to two important improvements in the quality of the induced models. First, the induced parameters are more reliable. Since these representations usually require less parameters, the frequency estimation for each parameter takes, on average, a larger number of samples into account and thus is more robust. Second, the global structure of the induced network is a better approximation to the real (in)dependencies in the underlying distribution. The use of local structure enables the learning procedure to explore networks that would have incurred an exponential penalty (in terms of the number of parameters required) and thus would have not been taken into consideration.

3.2.3 Gaussian network

This section deals with a domain composed of a set of continuous random variables. Its probabilistic model is a joint *probability density function* (PDF).

As a continuous variant of (3.5) it can be shown that each joint PDF can be factored into local terms as

$$f(\boldsymbol{Y}) = \prod_{i=0}^{n-1} f(\boldsymbol{Y}_i \mid \boldsymbol{\Pi_i}), \tag{3.6}$$

In the case of Gaussian network we assume that the value of each continuous random variable Y_i fulfills the normal (Gaussian) probability density function $N(\mu, \sigma)$, where the mean value μ_i of Y_i is affected by linear combination of "parent" values Y_j from $\boldsymbol{\Pi}_i$. The strength of influence between Y_i and Y_j is expressed by regression coefficient b_{ji}, the conditional deviation of Y_i from the predicted value is denoted σ_i.

Definition 3.3 (Gaussian network) :
Gaussian network (GN) for $\boldsymbol{Y}$ is the pair $G = (\mathcal{G},\mathcal{L})$. $\mathcal{G}$ is a DAG whose nodes correspond to the random continuous variables $Y_0, \ldots, Y_{n-1}$, and whose edges encode the independence assertions such that Y_i and $(\boldsymbol{Y} \setminus \boldsymbol{\Pi}_i)$ are independent given $\boldsymbol{\Pi}_i$. The second component $\mathcal{L}$ specifies the set of local probability density functions having the form of linear-regression model:

$$f(Y_i \mid \boldsymbol{\Pi_i}, \boldsymbol{\theta_i}) = N\left(Y_i; \mu_i + \sum_{Y_j \in \boldsymbol{\Pi}_i} b_{ji}(Y_j - \mu_j), \sigma_i \right), \tag{3.7}$$

where $N(Y_i;\mu,\sigma)$ is an univariate normal (Gaussian) distribution with mean μ and standard deviation $\sigma>0$. The local parameters of GN are given by $\boldsymbol{\theta_i} = (\mu_i, \boldsymbol{b_i}, \sigma_i)$, where $\boldsymbol{b_i} = (b_{0i}, b_{1i}, \ldots, b_{i-1\,i})^t$ is a column vector containing the linear regression coefficients. A missing arc from Y_j to Y_i implies that $b_{ji}=0$.

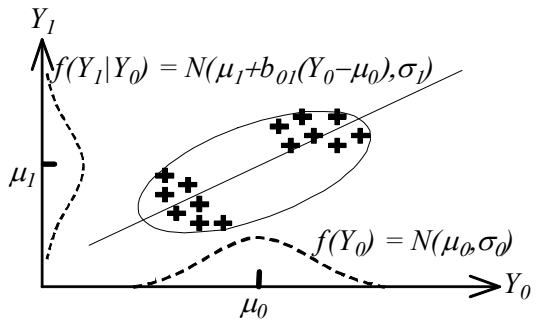


Fig. 3.3: Gaussian network for 2 variables. The variable Y_0 is independent and is expressed by unidimensional normal density with parameters μ_0, σ_0. Variable Y_1 is expressed by conditional normal density where the mean value μ_1 linearly changes with Y_0. The confidence region of resulting joint probability density function has the elliptical shape with the slope b_{01}.

It should be noted that each n-dimensional normal distribution for domain $\boldsymbol{Y}$ with mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ can be equivalently expressed in the conditional factored form as a Gaussian network and vice versa. The principle of transformation from unconditional normal distribution to conditional normal distribution follows from (3.3). However, instead of the linear regression, more general formula can be used for each local conditional PDF. This allows us to capture the factorization of a distribution which is more general than n-dimensional normal distribution.

3.2.4 Mixture distributions

Usually the sample points do not exhibit the multivariate normal distribution perfectly. For example in Fig. 3.3 you see that the density of samples (shown as bold crosses) is segmented and can be better approximated as a mixture of 2 normal distributions.

The key issue is the notion of *clusters*, which are possibly overlapping subsets of the original data set D. The nonoverlapping case of clustering is called *partitioning*. The use of clusters allows us to efficiently break up non-linear interactions so that we can use simpler models to get an adequate representation of data set.

Definition 3.4 (Mixture of multivariate normal distributions) :
More formally, consider the clustering of original data set D into k clusters $D=D_0 \cup D_1 \cup \ldots \cup D_{k-1}$. The mixture of multivariate normal distributions $f(\mathbf{Y})$ can be written as the sum of particular multivariate normal distributions

$$f(\boldsymbol{Y}) = \sum_{i=0}^{k-1} \beta_i \cdot f_{D_i}(\boldsymbol{Y}), \qquad (3.8)$$

The reason why we require to have an D_i in the superscript in f_{D_i} is to indicate that the i-th multivariate normal distribution is based upon the i-th cluster of samples. Each probability distribution is weighted by coefficient $\beta_i \in [0,1]$. In case the sum of all these weights equals to 1, the result is again a probability distribution.

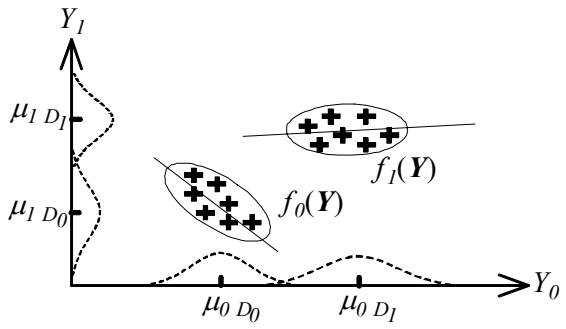


Fig. 3.4: The sample points from Fig. 3.3 can be partitioned into clusters D_0,D_1 and better approximated using mixture of two multivariate normal distributions.

The mixture approach is also used for discrete domains where the set of samples is sometimes clustered using Hamming distance. The joint probability distribution can then be written as

$$p(\boldsymbol{X}) = \sum_{i=0}^{k-1} \beta_i . p_{D_i}(\boldsymbol{X}). \tag{3.9}$$

3.2.5 Normal kernels

A joint normal kernels distribution is a mixture of joint normal distributions, where each point $\boldsymbol{y}^j$ in the modeled data serves as a center of one mixture component:

$$f(\mathbf{Y}) = \frac{1}{N} \sum_{j=0}^{N-1} f_{\mathbf{y}^j}(\mathbf{Y}), \tag{3.10}$$

such that all dimensions of mixture component are unbiased and independent:

$$f_{\mathbf{y}^j}(\mathbf{Y}) = \prod_{i=0}^{n-1} N(Y_i; y_i^j, \delta), \tag{3.11}$$

Using kernel distributions corresponds to using a fixed zero-mean normally distributed mutation for each promising solution as is often done in evolution strategies.

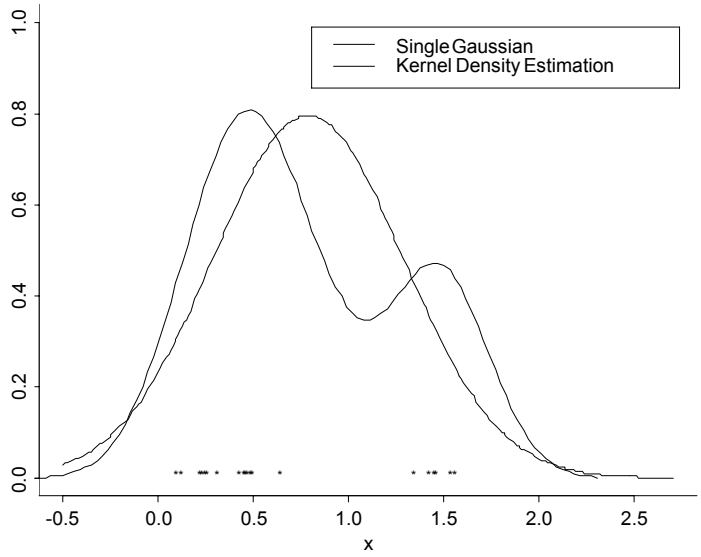


Fig. 3.4: Single Gaussian versus kernel density estimation for one variable.

3.3 Construction of probabilistic models

As in many machine learning problems, the problem of learning a graphical model from data can be divided into the problem of parameter learning (parameter fitting) and the problem of structure learning. Much progress has been made on the former problem, while the problem of learning the structure of a graph from data is significantly harder. In practice, most structure learning methods are heuristic methods that perform local search.

3.3.1 Fitting CPD parameters

Suppose we have a data set D containing N samples from the unknown joint probability distribution $p(\boldsymbol{X})$. The goal is to estimate the original distribution from the sample set D. Remember that the Bayesian network is composed of two components $B=(\mathcal{G},\mathcal{L})$. Sometimes the independence assertions encoded in the acyclic graph $\mathcal{G}$ are predefined. This is the simplest case, because only the parameters of CPD need to be estimated. For example assume that $\mathcal{G}$ is empty and all variables are independent.

$$p(\boldsymbol{X}) = \prod_{i=0}^{n-1} p(X_i) \tag{3.12}$$

In this trivial case each local CPD can be represented by univariate probabilities estimated as

$$p(x_i) \approx \frac{m(x_i)}{N}. \tag{3.13}$$

Note that this CPD representation needs only $|Dom(X_i)|$-1 parameters, because the last parameter is supplement to 1. In the case of general $\mathcal{G}$ the CPDs are estimated as

$$p(x_i \mid \boldsymbol{\pi_i}) \approx \frac{m(x_i, \boldsymbol{\pi_i})}{m(\boldsymbol{\pi_i})}, \tag{3.14}$$

where $m(x_i, \boldsymbol{\pi_i})$ denotes the number of simultaneous occurrence of x_i and $\boldsymbol{\pi_i}$ in the dataset, whereas $m(\boldsymbol{\pi_i})$ denotes the number of occurrences of $\boldsymbol{\pi_i}$. Note that for small data sets it is better to use the Bayesian estimate instead (see 3.3.3).

3.3.2 Fitting PDF parameters

For continuous domains the situation is similar. For example assume that the dependency graph $\mathcal{G}$ is empty and all variables are independent.

.

$$f(\boldsymbol{Y}) = \prod_{i=0}^{n-1} f(Y_i) \tag{3.15}$$

In the case of Gaussian network the joint probability density function among independent variables can be factored as a product of one-dimensional independent normal densities

$$f(\boldsymbol{Y} \mid \boldsymbol{\mu}, \boldsymbol{\sigma}) = \prod_{i=0}^{n-1} N(Y_i; \mu_i, \sigma_i) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi}\sigma_i} e^{\frac{1}{2}\left(\frac{Y_i - \mu_i}{\sigma_i}\right)^2}, \tag{3.16}$$

The vector of mean values $\boldsymbol{\mu}=\{\mu_0,\ldots,\mu_{n-1}\}$ can be estimated from data set D using the following equation for each component:

$$\mu_i \approx \frac{1}{N}\sum_{j=0}^{N-1} y_i^j \tag{3.17}$$

Since all the variables are independent, each component of vector $\boldsymbol{\sigma}$ can be estimated from D as:

$$\sigma_i \approx \sqrt{\frac{1}{N-1}\sum_{j=0}^{N-1}(y_i^j-\mu_i)^2} \tag{3.18}$$

Now let us consider more complicated case – a Gaussian network with predefined arbitrary dependence graph $\mathcal{G}$. For each variable Y_i the mean value μ_i can be estimated using (3.17) again. The regression coefficients $\boldsymbol{b_i}=(b_{0i},b_{1i},\ldots,b_{i-1\,i})^t$ can be estimated for example by min-square method and σ_i can be estimated as the standard deviation of the true value Y_i from the value predicted by regression model

$$\sigma_i^2 \approx \frac{1}{N-1}\sum_{j=0}^{N-1}(y_i^j-(\mu_i+\sum_{k\in\Pi_i} b_{ki}(y_k^j-\mu_k)))^2 . \tag{3.19}$$

Another method for obtaining parameters of Gaussian network follows from the equality with the n-dimensional unconditional joint PDF with mean vector $\boldsymbol{\mu}$ (estimated as (3.17)) and the covariance matrix $\boldsymbol{\Sigma}$ (computed as

$$\boldsymbol{\Sigma} \approx 1/N.\sum_{j=0}^{N-1}(\boldsymbol{y}^j-\boldsymbol{\mu}).(\boldsymbol{y}^j-\boldsymbol{\mu})^T) \tag{3.20}$$

or the precision matrix W (which is equal to $\boldsymbol{\Sigma}^{-1}$).

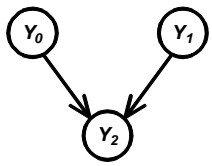


Fig. 3.4: An example of Gaussian network structure for variables Y_0, Y_1 and Y_2

For example, if the data satisfies the independence constraint shown in previous figure, then the relationship between covariance matrix and the desired regression coefficients holds

$$\boldsymbol{\Sigma}=\begin{pmatrix} \sigma_0 & 0 & b_{02}\sigma_0 \\ 0 & \sigma_1 & b_{12}\sigma_1 \\ b_{02}\sigma_0 & b_{12}\sigma_1 & \sigma_0 b_{02}^2+\sigma_1 b_{12}^2+\sigma_2 \end{pmatrix}$$

Generaly, the multidimensional normal PDF f_{NM} can be written as

$$f_{N_M}(\boldsymbol{Y}\mid\boldsymbol{\mu},\boldsymbol{\Sigma})=(2\pi)^{-n/2}\left|\boldsymbol{\Sigma}\right|^{-1/2}e^{-\frac{1}{2}(\boldsymbol{Y}-\boldsymbol{\mu})^T(\boldsymbol{\Sigma})^{-1}(\boldsymbol{Y}-\boldsymbol{\mu})}, \tag{3.21}$$

where $|\boldsymbol{\Sigma}|$ denotes the determinant of covariance matrix. Given (3.21) the Gaussian network can be obtained using the chain rule for factorization as

$$f(Y_i\mid\boldsymbol{\Pi_i},\boldsymbol{\mu},\boldsymbol{\Sigma})=\frac{f_{N_M}(Y_i,\boldsymbol{\Pi_i}\mid\boldsymbol{\mu}_{(Y_i,\boldsymbol{\Pi}_i)},\boldsymbol{\Sigma}_{(Y_i,\boldsymbol{\Pi}_i)})}{f_{N_M}(\boldsymbol{\Pi_i}\mid\boldsymbol{\mu}_{(\boldsymbol{\Pi}_i)},\boldsymbol{\Sigma}_{(\boldsymbol{\Pi}_i)})}, \tag{3.22}$$

where the subscript terms (Y_i,Π_i) resp. (Π_i) select from the mean vector and from the covariance matrix only those components corresponding to variables from the set $\{Y_i\}+\Pi_i$ resp. Π_i. Now the expression (3.22) can be transformed into the Gaussian network form, such that the regression coefficients and conditional deviation parameter can be identified. For further details see [19].

3.3.3 Bayesian versus classical estimation

The Bayesian estimation relates to Bayes theorem to estimate the probabilistic model. There is a slight controversy between two statistical schools: the classical and the Bayesian school. The classical school usually considers probabilities as an objective property of the nature, which can be measured with sufficient accuracy (and sufficient certainty) by repeating an experiment sufficiently many times. This is the frequentist interpretation of probabilities. The Bayesians on the other hand claim that probabilities merely signify our (lack of) knowledge of the world, and that they thus depend mainly on how much information we have about the problem. This makes probabilities in one sense subjective, in that people with different knowledge assign different probabilities to the same events (see also [54],[51]).

As an example, consider a scientist whose assistant flips a coin, notes the result, and puts a handkerchief over it, without letting the scientist see the result. Clearly the scientist and the assistant have different opinions of the probability of the coin being heads up.

The typical way to reason in a Bayesian manner is to accept more or less vague prior conception of the world. Then you make some observations or experiments, and given these you modify the conception to include the newly acquired knowledge. This can be repeated, so that further experiments may make your knowledge more relevant. In formulas this process is described by Bayes theorem

$$p(M \mid D, I) = \frac{p(D \mid M, I).p(M, I)}{p(D, I)}, \tag{3.23}$$

where $p(D|M,I)$ is the probability of getting the data D given the current model (or hypothesis) M and the background information (or context of the whole experiment) I, $p(M,I)$ is the *prior probability* of the model, *i.e.* what it is known about the world before the observations, and $p(M|D,I)$ is the *posterior probability*, or what you believe after observation of the data.

The term $p(D,I)$ expresses the whole probability of observation of data D, independently on the model M. It can thus be considered as a normalizing constant. For the model-estimation purposes the background information I can be omitted too. The simplified equation can be written as

$$p(M \mid D) \propto p(D \mid M).p(M) \tag{3.24}$$

where operator $\propto$ denotes the proportionality.

Classically there is a difference between a probability distribution and its parameters. For example there may be a random variable Y with a normal distribution with mean value μ, and standard deviation δ. Then it is possible to talk about the probability of Y having an outcome in some small interval, but it is normally not possible to talk about the probability of μ having some value.

In Bayesian statistics the parameters are just another set of random variables with some distributions, corresponding to the uncertainty about their real values. If a number of samples are taken from the variable *Y,* the parameter μ can (classically) be estimated as the mean value of these observations. But if this experiment is repeated, and the same number of samples are taken for each of several different estimates of μ, these estimates will differ slightly from each other. This motivates considering μ itself (at a fixed sample size) as a random variable. The interpretation is that after having seen a certain number of samples, each value of the parameter has a certain probability. Of course the standard deviation of μ gets smaller as the sample size increases, and the estimate gets closer to the "true" value.

The classical approach uses the concept of *maximum likelihood estimation (MLE).* This means that the estimated value of the parameter is the one that maximizes *p(D|M).* The Bayesian way of thinking is rather that what ought to be maximized is not the probability of getting the data that we already got, but the probability of the parameter given the data, *p(M|D).* As can be seen from (3.24), the relation between these two is just the prior probability distribution for the parameter *p(M)*, *i.e.* what we expect about the parameter (or about the whole model including its parameters) before observing the data.

As an example, consider estimating the probability of heads on a (possibly unbalanced) coin by tossing it several times and counting the number of heads and tails. Let us denote the parameter, the probability of heads, for q. The probability of getting c_H heads out of $c = c_H + c_T$ tosses is

$$p(D \mid M) = p(c_H \mid q) = \binom{c}{c_H} q^{c_H} (1-q)^{c_T} \tag{3.25}$$

To find the maximum likelihood estimate, differentiate with respect to q:

$$\begin{aligned} \frac{d}{dq} p(c_H \mid q) &= \binom{c}{c_H} c_H . q^{c_H - 1} (1-q)^{c_T} - \binom{c}{c_H} c_T . q^{c_H} (1-q)^{c_T - 1} \\ &= (c_H (1-q) - c_T q) \binom{c}{c_H} q^{c_H - 1} (1-q)^{c_T - 1} \end{aligned} \tag{3.26}$$

Equating with zero (and ignoring the trivial solutions $q = 0$ and $1 - q = 0$) gives

$$c_H (1-q) = c_T . q \quad \Rightarrow \quad \bar{q} = \frac{c_H}{c_H + c_T} = \frac{c_H}{c} \tag{3.27}$$

This is the classical estimate of the parameter q of a Bernoulli random variable. As expected, $\bar{q}$ can be shown to converge to the "true" value of q when the number of tosses increases. However, for small sample sizes (3.27) may give unsatisfactory results. For example, if after three tosses there has been two heads and one tail, it gives that the most likely q is two thirds. Even worse, if after two tosses there have been two tails, (3.27) gives the estimate that $q = 0$, *i. e.* that it is impossible to get a head. This is especially serious if the estimates are to be used as a basis for further probability calculations. Just because some event never occurred in the training data it does not necessarily mean that it is impossible. (Of course, we could test the estimate at some confidence level and reject it if it is too uncertain, but the point is that we do not want to reject any of the scare data, but use it for exactly what it is worth.)

Let us compare this outcome with the Bayesian estimate. First, a prior distribution for q *is needed*. Since the value of q may be anything, suppose first that we assume all probabilities q to be equally likely. Then the prior distribution of the model is

$$p(M) = p(q) = 1 \tag{3.28}$$

(where *p(q)* is now actually a probability density function for *q)* and the posterior distribution becomes

$$p(M \mid D) = p(q \mid c_H, c_T) \propto \binom{c}{c_H} q^{c_H} (1-q)^{c_T} \tag{3.29}$$

Now, this is not a binomial distribution, since it is not to be considered as a distribution for c_H any more, but as a distribution for the parameter q. Note that (3.29) is not an equality, but we have to normalize it to find the distribution. The calculation is slightly involved, but the result of integrating the right hand side (skipping the binomial coefficient which does not depend on *q)* is

$$\int_0^1 q^{c_H} (1-q)^{c_T} \, dq = \frac{c_H! c_T!}{(c_H + c_T + 1)!} \tag{3.30}$$

This gives the distribution for q as

$$p(q \mid c_H, c_T) = \frac{(c_H + c_T + 1)!}{c_H! c_T!} q^{c_H} (1-q)^{c_T} \tag{3.31}$$

which is a Beta distribution with parameters $c_T + 1$ and $c_H + 1$. Now, we are not really interested in the value of q which maximizes this term (which would be the same q as in the classical case), but more in the expectation of q. To calculate $E(q)$ we first use (3.30) again to evaluate the integral

$$\int_0^1 q.q^{c_H} (1-q)^{c_T} \, dq = \int_0^1 q^{c_H+1} (1-q)^{c_T} \, dq = \frac{(c_H + 1)! c_T!}{(c_H + c_T + 2)!} \tag{3.32}$$

and then use this to calculate the expectation

$$\begin{aligned} \bar{q} = E(q) &= \int q.p(q \mid c_H, c_T) dq = \frac{(c_H + c_T + 1)!}{c_H! c_T!} \int_0^1 q.q^{c_H} (1-q)^{c_T} \, dq \\ &= \frac{(c_H + c_T + 1)!}{c_H! c_T!} . \frac{(c_H + 1)! c_T!}{(c_H + c_T + 2)!} = \frac{c_H + 1}{c_H + c_T + 2} = \frac{c_H + 1}{c + 2} \end{aligned} \tag{3.33}$$

Note that for large samples this converges to the same result as the classical estimate, eq. (3.27), but for small samples it will tend more towards the value 1/2. For example, for two tails in two tosses it still estimates the probability of a head in the next toss to 1/4. Also note that since we have arrived at a whole distribution for q we need not stop at calculating the expectation, but can also get the variance which can be used as a measure of how certain the estimate is.

Usually the above prior for q is not used, but rather the more general form

$$p(q) \propto q^{\alpha-1}(1-q)^{\alpha-1} \tag{3.34}$$

for which a similar calculation gives the estimate of q as

$$\bar{q} = \frac{c_H + \alpha}{c + 2\alpha} \tag{3.35}$$

Note that this has as special cases the classical estimate when α = 0 and Laplace's formula for successions when α = 1. The parameter α expresses how much importance we give the prior and how much we trust the data directly.

Equation (3.35) holds for an estimation of probability of coin heads. The estimation of CPDs from the set of samples D can be done in the same way. Let us consider the independent discrete variable X_i with $||X_i||$ possible values – there is an analogy with a die having $||X_i||$ sides. We deal with the vector of parameters $\boldsymbol{q} = (q_0, q_1, ..., q_{\|X_i\|-1})$, each parameter q_{x_i} denotes probability estimate for one concrete side x_i of the die. The prior distribution for the whole $\boldsymbol{q}$ vector is selected as

$$p(\boldsymbol{q}) = \prod_{x_i} q_{x_i}{}^{\alpha-1} \tag{3.36}$$

This makes the posterior distribution of the parameters $\boldsymbol{q}$ a multi-Beta distribution (for more details see [47]):

$$p(\boldsymbol{q} \mid D) = \frac{\Gamma(N + \|X_i\|\alpha)}{\prod_{x_i} \Gamma(m(x_i) + \alpha)} \prod_{x_i} q_{x_i}{}^{m(x_i)+\alpha-1} \tag{3.37}$$

Here $\Gamma(x)$ is the gamma function which is defined for all positive real numbers and coincides with the factorial of x-1 for all positive integers x. The normalizing constant is here determined only by identification with a multi-Beta distribution, and is not easy to calculate directly.

Now we can use a very similar derivation as in (3.33), identifying $q_{x_i} . p(\boldsymbol{q} \mid D)$ with a new multi-Beta distribution, and take the quotient between the old and new normalizing constants. The resulting estimate of probability of one concrete outcome x_i is:

$$p(x_i) \approx \frac{m(x_i) + \alpha}{N + \|X_i\|\alpha} \tag{3.38}$$

The main critique from the classical statisticians is that a Bayesian needs to assert a "highly subjective" prior distribution before being able to calculate anything. However, in practice the Bayesian and classical results will converge very quickly to the same values, as the number of observations increases, as long as the prior is "reasonable" (nonzero everywhere for example). Also, when so called *non-informative priors* are used, the results are in many cases identical to the classical results. The problem with the classical method of estimation is that it may not work well for rare samples of data.

3.3.4 Learning BN by MDL metrics

In previous chapters we discussed the estimation of parameters $\mathcal{L}$. Now let us consider a more general case when both components of probabilistic model B=($\mathcal{G}$,$\mathcal{L}$) are unknown. First of all the independence assertions $\mathcal{G}$ are determined and then the set of local parameters $\mathcal{L}$ for given $\mathcal{G}$ is estimated.

One of the most frequent approaches is the MDL principle motivated by *universal coding*. The MDL framework dictates that the best model for a given data set is the one that minimizes the coding length/cost of both the data and the model. If a model fits the data exactly, the data need not be encoded and the cost is that of coding the model. If the data is not represented by a model, the cost is that of encoding the data. One can show that minimizing the MDL is equivalent to selecting the model that minimizes the Bayes risk assuming cost of errors is uniform, i.e. for a data set $\boldsymbol{D}$, the MDL prefers the model $\boldsymbol{M}$ for which $p(\boldsymbol{M}|\boldsymbol{D})$ is maximized. This can be shown by a simple application of Bayes rule which, after taking the logarithm of each side reduces this to

$$-\log(p(M \mid D)) = -\log(p(D \mid M)) - \log(p(M)) + \log(p(D)) \qquad (3.39)$$

Noting that p(D) is a constant for all models being compared, and that the minimal cost of encoding an object requires at least logarithm of its probability in bits, we see that MDL is proper for choosing the model with the maximum likelihood given the data. The lesson here is that a penalty must be paid if more complex models are used. The formula above gives the appropriate tradeoff.

Now the representation length required for the storage of both the network and the coded data will be expressed. To store a network $B = (\mathcal{G}, \mathcal{L})$ we need to describe $\boldsymbol{X}$, $\mathcal{G}$ and $\mathcal{L}$ to describe $\boldsymbol{X}$, we store the number of variables, n, and the cardinality of each variable X_i. Since $\boldsymbol{X}$ is the same for all candidate networks, we can ignore the description length of $\boldsymbol{X}$ in the comparisons between networks.

To describe the DAG $\mathcal{G}$, it is sufficient to store for each variable X_i a description of $\boldsymbol{\Pi_i}$ (namely, its parents in $\mathcal{G}$). This description consists of the number of parents, k, followed by the index of the set $\boldsymbol{\Pi_i}$ in some (agreed upon) enumeration of all $\binom{n}{k}$ sets of this cardinality. Since we can encode the number k using $\log(n)$ bits, and we can encode the index using $\log\binom{n}{k}$ bits, the description length of the graph structure is

$$DL_{graph}(\mathcal{G}) = \sum_{i=0}^{n-1}\left(\log n + \log\binom{n}{|\boldsymbol{\Pi}_i|}\right) \qquad (3.40)$$

To describe the CPDs in $\mathcal{L}$, we must store the parameters in each conditional probability table. For the table associated with X_i, we need to store $||\boldsymbol{\Pi_i}||(||X_i|| - 1)$ parameters. The representation length of these parameters depends on the number of bits we use for each numeric parameter. The usual choice in the literature is $\frac{1}{2}\log(N)$.

Thus, the encoding length of X_i's CPD is

$$DL_{tab}(X_i, \boldsymbol{\Pi_i}) = \frac{1}{2} \| \boldsymbol{\Pi_i} \| (\| X_i \| - 1) \log N \tag{3.41}$$

To encode the training data, we use the probability measure defined by the network B to construct a Huffman code for the instances in D. In this code, the exact length of each codeword depends on the probability assigned to that particular instance. We can approximate the optimal encoding length using $-\log p_B(\boldsymbol{x})$ as the encoding length of each instance $\boldsymbol{x}$ in the network B. Thus, the description length of the data is approximated by

$$DL_{data}(D \mid B) = -\sum_{j=0}^{N-1} \log p_B(\boldsymbol{x}^j) \tag{3.42}$$

We can rewrite this expression in a more convenient form, as a sum of terms that are "local" to each CPD:

$$DL_{data}(D \mid B) = -\sum_{i=0}^{n-1} \sum_{x_i, \boldsymbol{\pi}_i} m(x_i, \boldsymbol{\pi_i}) \log p_B(x_i \mid \boldsymbol{\pi_i}) = N \sum_{i=0}^{n-1} H(X_i \mid \boldsymbol{\Pi_i}), \tag{3.43}$$

where $H(X_i \mid \boldsymbol{\Pi_i}) = -\sum_{x_i, \boldsymbol{\pi}_i} p_B(x_i, \boldsymbol{\pi_i}) . \log p_B(x_i \mid \boldsymbol{\pi_i})$ is the conditional entropy of X_i, given $\boldsymbol{\Pi_i}$.

Finally, the MDL score of a candidate network structure $\mathcal{G}$, assuming that we choose parameters $\mathcal{L}$ as prescribed in equation (3.14), is defined as the total description length

$$DL(\mathcal{G}, D) = DL_{graph}(\mathcal{G}) + \sum_{i=0}^{n-1} DL_{tab}(X_i, \boldsymbol{\Pi_i}) + N \sum_{i=0}^{n-1} H(X_i \mid \boldsymbol{\Pi_i}) \tag{3.44}$$

The MDL metrics coincides with the so called Bayesian information criterion (BIC) which contains only DL_{data} and DL_{tab}. In the binary case, BIC assigns the network structure a score

$$BIC(\mathcal{G}) = \sum_{i=0}^{n-1} \left(-H(X_i \mid \boldsymbol{\Pi_i}) N - 2^{|\boldsymbol{\Pi}_i|} \frac{\log_2(N)}{2} \right) \tag{3.45}$$

Thus the BIC score is the negative of the MDL score, when we ignore the description of $\mathcal{G}$.

According to the MDL principle, we should strive to find the network structure that minimizes this description length. In practice, this is usually done by searching over the space of possible networks.

3.3.5 Learning BN by Bayesian-Dirichlet metrics (BDe)

Scores for learning Bayesian networks can also be derived from methods of Bayesian statistics. A prime example of such scores is the Bayesian-Dirichlet metrics (BDe), proposed by Heckerman et al. [47]. This score is based on earlier work of Cooper and Herskovits [27] and Buntine [21]. The BDe score is (proportional to) the posterior probability of each network structure, given the data. Learning amounts to searching for the network(s) that maximize this probability.

Let B_s^h denote the hypothesis that the underlying distribution satisfies the independencies encoded in $\mathcal{G}$ (see [47] for a more elaborate discussion of this hypothesis). The posterior probability we are interested in is $p(B_s^h \mid D)$.

Using Bayes' rule we write this term as

$$p\left(B_S^h \mid D\right) \propto p\left(D \mid B_S^h\right) p\left(B_S^h\right). \tag{3.46}$$

The term $p(B_S^h)$ is the prior probability of the network structure, and the term $p(D \mid B_S^h)$ is the probability of the data, given that the network structure is $\mathcal{G}$.

There are several ways of choosing a prior over network structures. Heckerman et al. suggest choosing a prior

$$p\left(B_S^h\right) = c\kappa^{\delta(\mathcal{G},\mathcal{G}')}, \tag{3.47}$$

where c is a normalization constant, $\delta(\mathcal{G},\mathcal{G}')$ is the difference in edges between $\mathcal{G}$ and a prior network structure $\mathcal{G}'$, and $0 < \kappa \leq 1$ is penalty for each such edge.

For a given hypothesis B_S^h about the independence structure $\mathcal{G}$, let Θ_{BS} represent the vector of parameters for the CPDs quantifying $\mathcal{G}$ (a random vector which represents estimate of $\mathcal{L}$). To evaluate the $p(D \mid B_S^h)$ we must consider all possible parameter assignments to $\mathcal{G}$. Thus,

$$p\left(D \mid B_S^h\right) = \int p\left(D \mid \Theta_{BS}, B_S^h\right) p\left(\Theta_{BS} \mid B_S^h\right) d\Theta_{BS} \tag{3.48}$$

where $p\left(D \mid \Theta_{BS}, B_S^h\right)$ is defined by (3.5), and $p\left(\Theta_{BS} \mid B_S^h\right)$ is the prior density over parameter assignments to $\mathcal{G}$. Heckerman et al. (following Cooper and Herskovits [27]) identify a set of assumptions that justify decomposing this integral:

1. Local distribution functions belong to the exponential family
2. The parameters of local CPDs are mutually independent
3. The parameters of local CPDs have conjugate priors
4. Database is complete

Roughly speaking, they assume that each distribution p($X_i|\boldsymbol{\pi_i}$) can be learned independently of all other distributions. Using this assumption, they rewrite $p(D \mid B_S^h)$ as

$$p\left(D \mid B_S^h\right) = \prod_{i=0}^{n-1} \prod_{\boldsymbol{\pi}_i} \int \left(\prod_{x_i} \theta_{x_i|\boldsymbol{\pi}_i}{}^{m(x_i,\boldsymbol{\pi}_i)} \right) p\left(\Theta_{X_i|\boldsymbol{\pi}_i} \mid B_S^h\right) d\Theta_{X_i|\boldsymbol{\pi}_i}, \tag{3.49}$$

where $\Theta_{X_i|\boldsymbol{\pi}_i}$ denotes a vector of parameters of size $||X_i||$ which serves for learning p($X_i|\boldsymbol{\pi_i}$). The coordinates of this vector are denoted $\theta_{x_i|\boldsymbol{\pi}_i}$. This decomposition is analogous to the transformation of equation (3.42) into (3.43). When the prior on each multinomial distribution $\Theta_{X_i|\boldsymbol{\pi}_i}$ is a Dirichlet prior, the integrals in (3.49) have a closed-form solution (see [47]).

I briefly review the properties of Dirichlet priors. For more detailed description, I refer the reader to DeGroot [31]. A Dirichlet prior for a multinomial distribution Θ_X of a variable X is specified by a set of hyperparameters $\{m'(x) \mid x \in \mathrm{Dom}(X)\}$.

We say that it has a Dirichlet distribution if

$$p(\Theta_X) \propto \prod_x \theta_x^{m'(x)}, \tag{3.50}$$

If the prior is a Dirichlet prior, the probability of observing a sequence of values of X with counts $m(x)$ is

$$\int \left(\prod_x \theta_x^{m(x)} \right) p\left(\Theta_X \mid B_S^h\right) d\Theta_X = \frac{\Gamma\left(\sum_x m'(x)\right)}{\Gamma\left(\sum_x (m'(x)+m(x))\right)} \prod_x \frac{\Gamma\left(m'(x)+m(x)\right)}{\Gamma\left(m'(x)\right)}. \tag{3.51}$$

Note that the parameters $m'(x)$ denote the priors for observed $m(x)$. Returning to the BDe score, if we assign to each $\Theta_{X_i|\boldsymbol{\pi}_i}$ a Dirichlet prior with hyperparameters $m'(x_i,\boldsymbol{\pi}_i)$, then

$$p(D \mid B_S^h) = \prod_{i=0}^{n-1} \prod_{\boldsymbol{\pi}_i} \frac{\Gamma(\sum_{x_i} m'(x_i,\boldsymbol{\pi}_i))}{\Gamma(\sum_{x_i} m'(x_i,\boldsymbol{\pi}_i)+m(\boldsymbol{\pi}_i))} \prod_{x_i} \frac{\Gamma(m(x_i,\boldsymbol{\pi}_i)+m'(x_i,\boldsymbol{\pi}_i))}{\Gamma(m'(x_i,\boldsymbol{\pi}_i))} \tag{3.52}$$

If we replace $\sum_{x_i} m'(x_i,\boldsymbol{\pi}_i)$ with $m'(\boldsymbol{\pi}_i)$ and introduce symbol ξ which denotes the background context of the Bayesian inference, then we obtain the final BDe metrics

$$\begin{aligned} p_{BDe}(D,B_S^h \mid \xi) &= p(B_S^h \mid \xi).p(D \mid B_S^h,\xi) = \\ &= p(B_S^h \mid \xi) \prod_{i=0}^{n-1} \prod_{\boldsymbol{\pi}_i} \frac{\Gamma(m'(\boldsymbol{\pi}_i))}{\Gamma(m(\boldsymbol{\pi}_i)+m'(\boldsymbol{\pi}_i))} \prod_{x_i} \frac{\Gamma(m(x_i,\boldsymbol{\pi}_i)+m'(x_i,\boldsymbol{\pi}_i))}{\Gamma(m'(x_i,\boldsymbol{\pi}_i))} \end{aligned} \tag{3.53}$$

It is necessary to establish the prior values $m'(\boldsymbol{\pi}_i)$ and $m'(x_i,\boldsymbol{\pi}_i)$ which denote the expected values of counts $m(\boldsymbol{\pi}_i)$ and $m(x_i,\boldsymbol{\pi}_i)$. The simplest approach was used in the K2 algorithm proposed in Cooper and Herskovits in [27], they used uniform assignment for priors $m'(x_i,\boldsymbol{\pi}_i)=1$; $m'(\boldsymbol{\pi}_i)=\sum_{x_i} m'(x_i,\boldsymbol{\pi}_i) = \| X_i \|$, which yields the K2 metric:

$$\begin{aligned} p_{K2}(D,B_S^h \mid \xi) &= p(B_S^h \mid \xi) \prod_{i=0}^{n-1} \prod_{\boldsymbol{\pi}_i} \frac{\Gamma(\| X_i \|)}{\Gamma(m(\boldsymbol{\pi}_i)+\| X_i \|)} \prod_{x_i} \Gamma(m(x_i,\boldsymbol{\pi}_i)+1) = \\ &= p(B_S^h \mid \xi) \prod_{i=0}^{n-1} \prod_{\boldsymbol{\pi}_i} \frac{(\| X_i \|-1)!}{(m(\boldsymbol{\pi}_i)+\| X_i \|-1)!} \prod_{x_i} m(x_i,\boldsymbol{\pi}_i)! \end{aligned} \tag{3.54}$$

It can be shown that asymptotically for $N \to \infty$ both BDe and MDL metrics prefer the same networks.

The problem of finding the best network given a scoring metric has been proven to be NP-complete for most Bayesian and non-Bayesian metrices (see [23]). However, since most of the commonly used metrics (such as the BDe and MDL metrics) can be decomposed into independent terms each of which corresponds to one variable (node), to construct a network, a greedy algorithm, local hill-climbing, or simulated annealing can be used very efficiently.

In my experiments I have used a simple greedy search algorithm. Each iteration of the algorithm, the graph operation that improves the network score the most is performed. The simple operations that can be performed on a network include edge additions, edge reversals, and edge removals. Only operations that keep the network acyclic are allowed and the number of parents of each node is bound by a constant in order to avoid superfluously complex models. The construction finishes when no operations are allowed or no applicable graph operation improves the score.

3.3.6 BDe metrics for Bayesian networks with local structure

It has been shown (see [24] or [96]) that the Bayesian score can be computed for Bayesian networks where the independence constraints are encoded by a decision graph for each of the variables in a very similar way.

Conditional probabilities for a variable X_i are stored in a decision graph G_i, i.e. for each variable there is one decision graph. The outer product from BDe remains the same. The middle product runs over all leaves of the decision graph G_i corresponding to the variable X_i. The inner-most product runs over all possible instances of the variable X_i. Thus,

$$p(D,B_s^h \mid \xi) = p(B_s^h \mid \xi).\prod_{i=0}^{n-1}\prod_{l\in L_i}\frac{\Gamma(m'(i,l))}{\Gamma(m(i,l)+m'(i,l))}\prod_{x_i}\frac{\Gamma(m(x_i,i,l)+m'(x_i,i,l))}{\Gamma(m'(x_i,i,l))}, \quad (3.55)$$

where B_s^h is now the hypothesis that the independence constraints can be expressed by the set of decision graphs $G_0,\ldots,G_{n-1}$; L_i is the set of leaves in the decision graph G_i for X_i, $m(i,l)$ is the number of instances in D which end up the traversal through the graph G_i in the leaf i, $m(x_i,i,l)$ is the number of instances that have $X_i = x_i$ and end up the traversal of the graph G_i in the leaf l. The values $m'(i,l)$ and $m'(x_i,i,l)$ represent our prior knowledge about the values of $m(i,l)$ and $m(x_i,i,l)$.

Again, by setting $m'(x_i,i,l)=1$; $m'(i,l)=\sum_{x_i} m'(x_i,i,l) = \| X_i \|$, the K2 metrics for decision trees can be written as

$$p_{K2}(D,B_s^h \mid \xi) = p(B_s^h \mid \xi).\prod_{i=0}^{n-1}\prod_{l\in L_i}\frac{\Gamma(\| X_i \|)}{\Gamma(m(i,l)+\| X_i \|)}\prod_{x_i}\Gamma(m(x_i,i,l)+1), \quad (3.56)$$

To favor simpler networks to the more complex ones we can set the prior probability of a network $p(B_s^h \mid \xi)$ to decrease exponentially with the description length of the set of parameters they require (see the MDL chapter for DL_{tab}). Thus,

$$p(B_s^h \mid \xi) = c.2^{(-0.5\log_2 N)\sum_i |L_i|}, \quad (3.57)$$

where c is a normalization constant which does not affect the relative comparisons.

Each iteration, the greedy construction of decision graphs examines all possible operators and selects the one that improves the score most. The operators include (1) splitting a leaf of some decision graph on a variable that was not encountered on the path from the root to the leaf and (2) merging two leaves into a single leaf.

3.3.7 Learning Gaussian networks structure

For Gaussian networks we can also use Bayesian approach to search for proper network structure. The BGe metrics (Bayesian metrics for Gaussian networks having score equivalence) was proposed by Geiger & Heckerman in [38]:

$$f(D, B_S^e \mid \xi) = p(B_S^e \mid \xi) \prod_{i=0}^{n-1} \frac{f(D^{Y_i, \boldsymbol{\Pi}_i} \mid B_{S_c}^e, \xi)}{f(D^{\boldsymbol{\Pi}_i} \mid B_{S_c}^e, \xi)}, \tag{3.58}$$

where B_S^e denotes the event (hypothesis) that the data D were sampled from minimal Gaussian network with structure $\mathcal{G}$, ξ denotes the background information, $\boldsymbol{\Pi_i}$ denotes the set of parent variables of Y_i in $\mathcal{G}$, $D^{Y_i, \boldsymbol{\Pi}_i}$ resp. $D^{\boldsymbol{\Pi}_i}$ denotes the data D restricted only to $\{Y_i\} \cup \boldsymbol{\Pi}_i$ resp. $\boldsymbol{\Pi}_i$ columns and $B_{S_c}^e$ denotes the Gaussian network with complete structure (with no missing edges). This equation holds under similar assumptions (parameter independence, parameter modularity, etc. - see 3.3.5) as the BDe metrics.

Note that this direct search for $\mathcal{G}$ is quite time-consuming, because the computation of $f(D^{Y_i, \boldsymbol{\Pi}_i} \mid B_{S_c}^e, \xi)$ resp. $f(D^{\boldsymbol{\Pi}_i} \mid B_{S_c}^e, \xi)$ involves the estimation of parameters for $B_{S_c}^e$ itself (see 3.3.2).

However, it is possible to adopt additional assumptions and heuristics that make the search for structure faster.

3.3.8 Learning normal mixtures

Finding an estimate of the maximum likelihood parameters of a mixture model, for a given data set, is a non-linear constrained optimization problem which is difficult to solve. The EM (Expectation-Maximisation) algorithm provides a numerical method for estimating these maximum likelihood parameters. In the context of finding an optimal normal mixture model, the data is implicitly modelled as consisting of k populations, or groups, with each assumed to be normally distributed. The incompleteness arises from the fact that the population of origin of a given data point is unknown. If the group membership were known for each data point, finding an estimate of the maximum likelihood parameters would simply be a matter of finding the sample mean and covariance matrix for each group of data, then determining the weightings of the components by the fraction of points in the sample belonging to each group. The EM algorithm alternately estimates the group membership of the data points using a previous estimate of the parameters of the model, and then updates this estimate of the parameters using the estimate of the group membership of the data points:

Begin by initializing the parameters of the model. For example let each covariance matrix be of the form

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_0 & 0 & \dots & 0 \\ 0 & \sigma_1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \sigma_{k-1} \end{bmatrix},$$

where each δ_i is equal to roughly 20% of the length of the range of Y_i, and assign each mean vector $\boldsymbol{\mu_i}$ to a randomly chosen location in the feature space. Let p_i be the fraction

$$p_i(\boldsymbol{y}) = \frac{\beta_i f_i(\boldsymbol{y})}{\sum_{j=0}^{k-1} \beta_j f_j(\boldsymbol{y})}, \tag{3.59}$$

The EM algorithm effectively splits each data point $\boldsymbol{y}$ among the k-mixture Gaussian according to the fraction $p_i(\boldsymbol{y})$. (That is, the probability that a point belongs to population i is estimated by p_i). The new estimate for the weight of each component Gaussian is given by

$$\beta_i' = \frac{\sum_{\boldsymbol{y} \in D} p_i(\boldsymbol{y})}{|D|}, \tag{3.60}$$

The re-estimates for the parameters $\boldsymbol{\mu_i}$ and $\boldsymbol{\Sigma_i}$ of the i-th component Gaussian are calculated by

$$\boldsymbol{\mu_i}' = \frac{\sum_{\boldsymbol{y} \in D} \boldsymbol{y}.p_i(\boldsymbol{y})}{\sum_{\boldsymbol{y} \in D} p_i(\boldsymbol{y})}, \tag{3.61}$$

$$\boldsymbol{\Sigma_i}' = \frac{1}{\sum_{\boldsymbol{y} \in D} p_i(\boldsymbol{y}) - 1} \sum_{\boldsymbol{y} \in D} p_i(\boldsymbol{y}) \left[(\boldsymbol{y} - \boldsymbol{\mu_i}').(\boldsymbol{y} - \boldsymbol{\mu_i}')^T \right]. \tag{3.62}$$

Now recalculate the fraction p_i and repeat the process until the likelihood no longer increases, or until the number of iterations exceeds some predetermined number. The final parameters of each component $\boldsymbol{\mu_i}$ and $\boldsymbol{\Sigma_i}$ are transformed into factorized conditional form (see [19]) and the coefficients β_i are used for proportional selection of components during sampling.

To obtain better results with EM algorithm, it is also possible to use some explicit clustering technique for preprocessing. See paper [19] for detailed description of IDEA algorithms with k-means clustering or leader-k clustering, where Euclidean or Mahalanobis metrics is used for distance of data samples.

3.3.9 Learning normal kernels

The construction of Gaussian normal kernels (see 3.2.5) is straightforward and does not require any search for structure. Simply, the mean of each kernel is equal to coordinate of one sample. The variance of kernels δ is the only parameter that is not easy to determine. It can be either fixed to a relatively small value, or it is possible to directly take up methods for adapting the variance of each kernel from evolution strategies.

3.4 Model sampling

Once the desired probability distribution is found, new samples can be generated using Gibbs sampling ([39]) or Probabilistic Logic Sampling (PLS, [48]).

3.4.1 Gibbs sampling

Gibbs sampling, named by its authors Geman &; Geman [39] after the physicist J.W. Gibbs, can be the easiest to implement. In its basic form, it changes just one component of $\boldsymbol{X}$, say the i-th component X_i, at a time. All other components preserve their values, but the i-th component is chosen by re-sampling from the marginal distribution $p(X_i|X_0,\ldots,X_{i-1},X_{i+1},\ldots,X_{n-1})$. With this algorithm, the acceptance probability is always 1, so transitions are never rejected. The choice of component i can be deterministic or random.

Gibbs sampling yields a particularly slow algorithm if there is strong coupling between components of $\boldsymbol{X}$, i.e. whenever the "mountain" $p(\boldsymbol{X})$ is high, and has sharp ridges that are not conveniently aligned with the coordinate axes.

3.4.2 PLS algorithm

The Probabilistic Logic Sampling (PLS, [48]) takes advantage on how a Bayesian network defines a probability distribution. It generates the values for the variables following their ancestral ordering which guarantees that $p(\boldsymbol{X})$ will be instantiated every time, whereas Gibbs sampling converges to the true $p(\boldsymbol{X})$ distribution only after large number of re-sampling. The pseudo-code for PLS can be written as:

```
Find an ancestral ordering o=(o0,o1,…,on-1) of nodes in Bayesian network;
for i:=0 to n-1 do
begin
   X_oi := new sample from p(X_oi | Π_oi);
end
```

3.5 Cross validation of model construction

The accuracy of model estimating techniques presented in previous chapters can be determined by cross-validation. Some reference probabilistic model is sampled to obtain the test data which are then used to estimate new model. The agreement between those two models determines the accuracy.

A popular measure for estimating the approximation error between distributions is the entropy distance proposed by Kullback and Leibler.

Definition 3.5 (KL divergence) :
The Kullback Leibler divergence between two distributions p and $\bar{p}$ is defined as

$$KL(p,\bar{p}) = \sum_{\boldsymbol{x}} \bar{p}(\boldsymbol{x}).\log\frac{\bar{p}(\boldsymbol{x})}{p(\boldsymbol{x})} \tag{3.63}$$

The KL measure has the following properties:

- $KL(p,\bar{p}) \geq 0$
- $KL(p,\bar{p}) = 0$ iff $p = \bar{p}$

Note that the equality $p = \bar{p}$ means that both models return the same value given the same argument, but both models might be of different structure. For example if the problem is additively decomposable into independent subproblems, then the product of such local probability distributions gives the same results as the n-dimensional joint probability distribution over the whole Dom($\boldsymbol{X}$).

KL divergence between constructed model and complete joint probability distribution can also serve as a metrics for model construction.

4 EDA algorithms

In previous chapter I have introduced useful probabilistic models and algorithms to find such models. This chapter elaborates on how these probabilistic models can be used for optimization tasks in Estimation of Distribution Algorithms (EDAs, also called PMBGA - Probabilistic Model-Building Genetic Algorithms). See excellent reviews [92] and [65].

4.1 Notation

For the description of Estimation of Distribution Algorithms I extend the notation for probabilistic models introduced in chapter 3.1. The genotype of length n is interpreted as a vector of random variables $\boldsymbol{X}=(X_0,\ldots,\ X_{n-1})$. The terms *gene, random variable* and *dependency graph node* are equivalent in the EDA framework. Upper case symbol X_i means the i-th gene variable, lower case symbol x_i means one concrete allele. Note that in 3.1 the boldface symbols such as $\boldsymbol{X}$ denote sets of random variables whereas in EDA framework I use the same notation for random vectors, nevertheless both definitions result in the joint probability distribution $p(\boldsymbol{X})$. The parent population of chromozomes D can be written as a data set of N concrete vector samples $D=\{\boldsymbol{x}^0,\boldsymbol{x}^1,\ldots,\boldsymbol{x}^{N-1}\}=\{(x^0_0,x^0_1,\ldots,x^0_{n-1}),(x^1_0,x^1_1,\ldots,x^1_{n-1}),\ldots,(x^{N-1}_0,x^{N-1}_1,\ \ldots,x^{N-1}_{n-1})\}$. I use superscript numbers to denote the index of concrete individual within the population, such that $\boldsymbol{x}^j$ denotes the concrete instance of j-th chromozome, $F(\boldsymbol{x}^j)$ denotes its fitness and x^j_i denotes the concrete value of i-th gene of j-th chromozome. I use symbol $D(t)$ to emphasize the instantaneous content of population in time t, then $\boldsymbol{x}^j(t)$ denotes its j-th chromozome and $p_{D(t)}(\boldsymbol{X})$ denotes the joint probability distribution of chromozomes in $D(t)$. Symbol $P(t)$ denotes the whole population from which $D(t)$ was selected and L denotes its size. Usually 50% selection presure is used, so $L=2*N$.

4.2 Basic principles

The general procedure of EDA algorithm is similar to that of GA, but the classical recombination operators are replaced by probability estimation followed by probability sampling. First, the initial population of EDA is generated randomly. In each iteration, promising solutions are selected from the current population of candidate solutions and the true probability distribution of these selected solutions is estimated. New candidate solutions are then generated by sampling the estimated probability distribution. The new solutions are then incorporated into the original population, replacing some of the old ones or all of them. The process is repeated until the termination criteria are met.

The pseudo-code of general EDA algorithm can be written as follows:

```
Set t := 0;
Randomly generate initial population P(0);
while termination criteria are not satisfied do
begin
  Select a set of promising/parent solutions D(t) from P(t);
  Estimate the probability distribution of the selected set D(t);
  Generate a set of new strings O(t) according to the estimate;
  Create a new population P(t+1) by replacing part of P(t) by O(t);
  Set t := t+1;
end
```

To better understand the basic principles of EDAs, I apply the simplest version of this approach to well known OneMax function (see chapter 7.4). Suppose we try to obtain the maximum of function $F_{OneMax}(x_0,...,x_{n-1}) = \sum_{i=0}^{n-1} x_i$. For example let the chromozome length be n=5, the whole population size is 10 and the parent population size is 5. See the following figure.

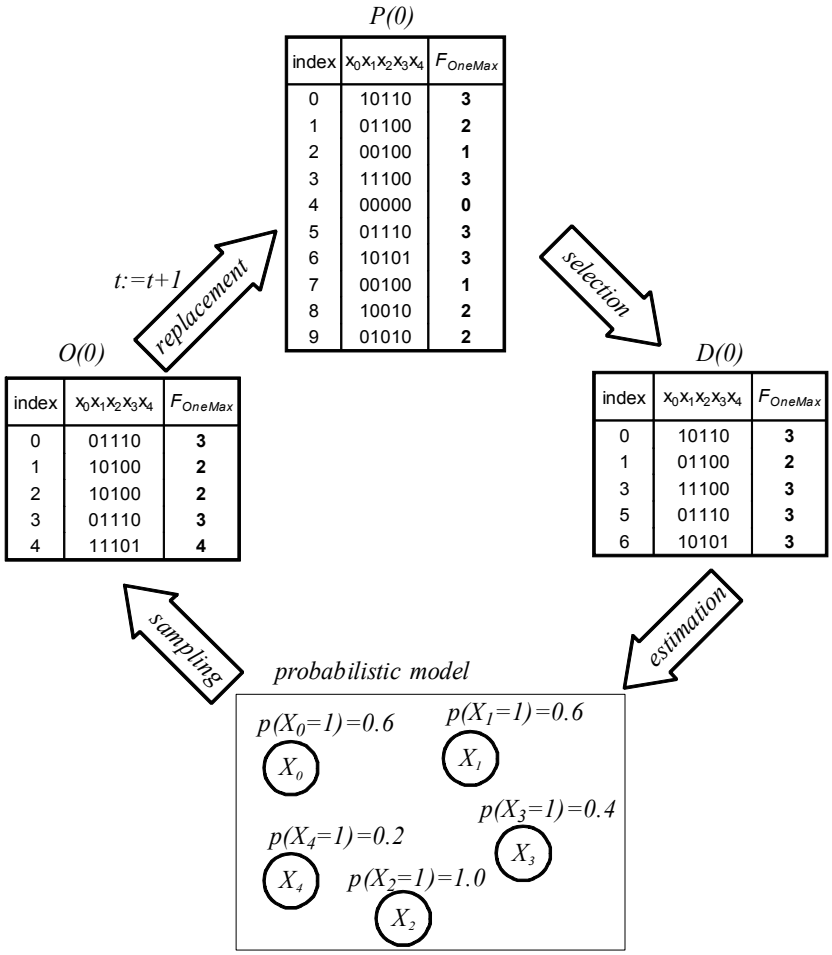


Fig. 4.1*: Example of EDA cycle for OneMax problem of size n=5 with independent genes/variables.*

The initial population is obtained by random sampling of probability distribution $p(\boldsymbol{X}) = \prod_{i=0}^{n-1} p(X_i)$. The initial distribution is uniform, so $\forall i \in \{0,...,n\text{-}1\}$: $p(X_i=1) = p(X_i=0) = 0.5$. An example of concrete population $P(0)$ shown in the figure. The better half of $P(0)$ is selected into $D(0)$ with respect to F_{OneMax}. You see that individuals no. 0,1,3,5,6 are selected. Then the probability model is estimated from $D(0)$. Since the genes in OneMax problem are independent, there are no edges between nodes in the graph. The estimated frequency of 1-alleles is shown above the nodes. Then $O(0)$ is generated using these frequencies. When we evaluate $O(0)$, we see that the last individual in $O(0)$ is better than any previous solution. During replacement stage, $P(1)$ is created by replacing the worst part of $P(0)$ (individuals 2,4,7,8,9) by $O(0)$. And the whole cycle is repeated until termination criteria are met or maximum number of generations is reached.

4.3 Initial discrete probabilistic models

4.3.1 Without dependencies – PBIL, UMDA, cGA

In all the algorithms belonging to this category it is assumed that all parameters of a given problem are independent. That is, the *n*-dimensional joint probability distribution of solutions can be factored as a product of independent univariate probability distributions.

PBIL

The Population Based Incremental Learning (PBIL) was introduced by Baluja in [3]. The solutions are represented by binary strings of fixed length, but the population of solutions is replaced by the so-called probability vector (p_0, p_1,…, p_{n-1}), where p_i refers to the probability of obtaining a value of 1 in the *i*-th gene. This vector is initially set to assign each value at the same position with the same probability $p_i = 0.5$. At each generation a number of solutions is generated using this vector. Then the few best solutions are selected and the probability vector is shifted towards the selected solutions by using Hebbian learning rule:

$$p_i = (1-\alpha)p_i + \alpha\frac{1}{N}m(X_i = 1), \tag{4.1}$$

where N is the size of selected population, $\alpha \in (0,1]$ is a parameter of the algorithm and $m(X_i = 1)$ denotes numer of individuals in the selected part of population having X_i=1. Note that PBIL with $\alpha \neq 1$ introduces conservativeness, which is not typical for EDAs.

PBIL has received much attention from EDA researchers. For comparison of PBIL with heuristic approaches see [4], [75], for proposed PBIL extensions see [5], [63] and theoretical aspects of PBIL includes the paper [49].

UMDA

The Univariate Marginal Distribution Algorithm (UMDA) was introduced by Műehlenbein in [76]. In contrast to PBIL, the population of solutions is kept and processed. In each iteration the frequency functions on each position for the selected set of promising solutions are computed and these are then used to generate new solutions. The new solutions replace the old ones and the process is repeated until the termination criteria are met. From the implementation point of view the UMDA matches the pseudo-code of typical EDA algorithm (see 4.2), except that the dependence graph of constructed probabilistic model contains no edges.

The probabilistic model is as simple as possible:

$$p(\boldsymbol{X}) = \prod_{i=0}^{n-1} p(X_i), \tag{4.2}$$

where each univariate marginal distribution $p(X_i)$ is estimated from marginal frequencies:

$$p(X_i = 1) = \frac{m(X_i = 1)}{N}, \tag{4.3}$$

where N is the size of selected population and $m(X_i = 1)$ denotes number of individuals in the selected part of population having X_i=1.

For a mathematical analysis of UMDA see papers [76], [74] and [80]. A modification of offspring generation proposed in [106] enables UMDA to solve problems with constraints.

For its simple implementation UMDA is still very popular, namely in combination with the clustering approach.

cGA

In the compact Genetic Algorithm (cGA) proposed by Harik et. al. in [46] the population is replaced by a single probability vector like in PBIL. However, unlike the PBIL, it modifies the probability vector so that there is direct correspondence between the population that is represented by the probability vector and the probability vector itself. Two individuals are generated from this vector of probabilities, the competition between them is carried out and the winner is used for vector update. But instead of shifting the vector proportionally to the distance from either 0 or 1, each component of the vector is updated by shifting its value by the contribution of a single individual to the total frequency assuming a particular population size. By using this update rule, theory of simple genetic algorithms can be directly used in order to estimate the parameters and behavior of the cGA.

4.3.2 Bivariate dependencies – BMDA, MIMIC, Dependency trees

All algorithms described in previous chapter perform similarly. They work very well for linear problems where they achieve linear or sub-quadratic performance, depending on the type of a problem, but they fail on problems with strong interactions among variables. Obviously the independence assumption is not valid for most optimization problems, because interdependencies between the variables usually exist. In some cases it is sufficient to consider second-order statistics and use the following algorithms based on pairwise dependencies. In contrast to algorithms from previous chapter, the problem with determination of dependencies occurs.

MIMIC

The Mutual Information Maximization for Input Clustering (MIMIC) algorithm was developed by De Bonet et. al. in [12]. Each generation MIMIC selects promising solutions and searches for the best permutation between the variables in order to maximize the so-called mutual information of neighboring variables. The chain probability distribution can be written as

$$p(\boldsymbol{X}) = p(X_{o_{n-1}}) \prod_{i=0}^{n-2} p(X_{o_i} \mid X_{o_{i+1}}), \tag{4.4}$$

where $\boldsymbol{o}=(o_0,o_1,\ldots,o_{n-1})$ is a permutation of numbers from $\{0,1,\ldots,n\text{-}1\}$. The Kullback-Leibler divergence (see 3.5) between the chain and the complete joint distribution is minimized. The greedy algorithm starts in a variable with the lowest unconditional entropy. The chain is expanded by adding a new variable that minimizes the conditional entropy of the new variable given the last variable in the chain. Once the full chain is constructed for the selected population of promising solutions, new solutions are generated by sampling the distribution.

COMIT

The COMIT algorithm (Combining Optimizers with Mutual Information Trees) was proposed by Baluja and Davies in [8] and [9]. A dependency tree encodes the probability distribution where the variable in the root of the tree is independent, and each other variable is conditioned on its parent in the tree. COMIT uses minimum spanning tree technique - one variable is randomly chosen to form the root of the tree, and then the remaining variables are "hanged" to the existing tree so that the mutual information between the parent of the new variable and the variable itself is maximized.

Once a whole tree is constructed, new solutions are generated according to the distribution encoded by the constructed dependency tree and the conditional probabilities estimated from the population. Moreover, MIMIC hybridizes the EDA approach with local optimizers – the newly generated solutions can be improved by local search.

BMDA

The Bivariate Marginal Distribution Algorithm (BMDA) was proposed by Pelikan and Műehlenbein in [89]. The pairwise dependencies in BMDA are discovered by Pearson's chi-square statistics. In my master thesis [xvi] I implemented BMDA with the following form of Pearson's chi-square statistics:

$$\chi^2_{i,j} = N\left(\sum_{\forall x_i \in Dom(X_i)} \sum_{\forall x_j \in Dom(X_j)} \frac{m^2(x_i, x_j)}{m(x_i)m(x_j)} - 1 \right), \tag{4.5}$$

where N is the size of parent population and $m(x_i,x_j)$, $m(x_i)$ resp. $m(x_j)$ denote the number of individuals in the parent population with concrete values of x_i and/or x_j. From the theoretical point of view this metric can be seen as statistical testing of hypothesis – for example binary genes X_i and X_j are considered to be independent at 95% confidence level if $\chi^2_{i,j} < 3.84$. Like COMIT, BMDA also uses a variant of minimum spanning tree technique to learn a model. However, during construction of the tree, if none of the remaining variables can be "hanged" to existing tree, then BMDA starts to form another tree from remaining variables. The final probability distribution is thus a forest distribution (a set of mutually independent dependency trees):

$$p(\boldsymbol{X}) = \prod_{X_r \in R} p(X_r) \prod_{X_i \in V \setminus R} p(X_i \mid X_{j(i)}), \tag{4.6}$$

where V is the set of vertices of dependency tree, R is the set of root nodes and $X_{j(i)}$ denotes the parent node of X_i. Given the tree dependence structure, the univariate marginal probability distributions $p(X_r)$ are estimated from the promising/parent population as (4.3) and the bivariate conditional probability distributions $p(X_i \mid X_{j(i)})$ are estimated as

$$p(x_i \mid x_{j(i)}) = \frac{m(x_i, x_{j(i)})}{m(x_{j(i)})}. \tag{4.7}$$

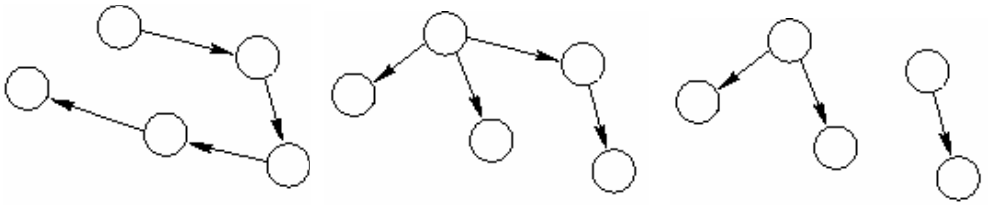


Fig. 4.2*: Example of dependency graph for: a) MIMIC, b) COMIT, c)BMDA*

4.4 Advanced discrete probabilistic models

The algorithms presented in previous section can identify, propagate, and mix building blocks of order two. Nonetheless, capturing only some pair-wise interactions has still shown to be insufficient for solving problems with multivariate or highly overlapping building blocks. That is why following algorithms have been proposed.

4.4.1 Explicit factorization – FDA

The Factorized Distribution Algorithm (FDA, see [81]) uses a fixed dependency graph throughout the whole computation. The model is allowed to contain multivariate marginal and conditional probabilities, but FDA learns only the CPT parameters, not the dependency graph. The problem must be decomposed and factorized in advance by expert. For additively decomposed functions this decomposition is theoretically justified, but for real world problems the factorization is not straightforward. However, the theoretical results for FDA are very important for understanding the behavior of EDAs, see papers [77],[78],[79],[80],[122] and [74].

4.4.2 Clustering – EcGA

Extended compact Genetic Algorithm (EcGA) was proposed by Harik in [45]. The basic idea is to use a marginal product model to estimate the joint probability distribution. The variables are grouped together and each group of variables is assumed to be independent of the rest. If we consider each group as one independent multinomial variable, the model is an analogy to UMDA model:

$$p(\boldsymbol{X}) = \prod_{i=0}^{k-1} p(\boldsymbol{X}_i), \tag{4.8}$$

where $\boldsymbol{X}_{0,\ldots,}\boldsymbol{X}_{k-1}$ are disjoint subsets of variables from $\boldsymbol{X}$. Each generation the algorithm that carries out the grouping begins with n groups, one variable in each group. Then the algorithm iteratively merges two groups that maximize the improvement of the model with respect to Bayesian Information Criterion (BIC, see 3.3.4). If no more improvement is possible, the current model is used. The advantage of using MDL-based metrics is that they penalize complex models when they are not supported by significant statistical evidence. If the constructed model reflects a proper decomposition of the problem, EcGA is a very powerful algorithm [108], [107]. However, many real-world problems contain overlapping dependencies, which cannot be accurately modeled by dividing the variables into disjoint partitions. This can result in poor performance of EcGA on those problems.

4.4.3 Bayesian network – BOA, EBNA, LFDA

The Bayesian Optimization Algorithm (BOA) was developed by Martin Pelikan in [87],[88]. It uses Bayesian network (BN) to encode the structure of a problem (see 3.2.1):

$$p(X_0,\ldots,X_{n-1}) = \prod_{i=0}^{n-1} p(X_i \mid \boldsymbol{\Pi}_i). \tag{4.9}$$

The methods and metric for BN building were adopted from the area of data mining, namely from Heckerman, Geiger & Chickering [47]. The Bayessian Dirichlet metric (BDe) is used to measure the quality of the network. A special case of BD metric, so-called K2 metric, is used when no prior information about the problem is available. For detailed explanation of methods for learning BN structure using BD metric see chapter 3.3.5.

Many algorithms can be used to build up the network from parent population. The optimal search is NP-hard, so in the implementation [91] a simple greedy algorithm was used with only one edge addition in each step. It is a variant of Algorithm B from Buntine [21]. The Algorithm B starts with an arc-less structure $\mathcal{G}$ and for each edge that can be added it computes the K2 metrics of the structure $\mathcal{G}'$ that can be constructed from $\mathcal{G}$ by adding this edge. The edge giving the highest metrics improvement is then added to the network $\mathcal{G}$. This process is repeated until no more improvement is possible. By the term 'edge can be added' we mean the test whether the edge keeps the network acyclic, does not exceed the limit of incoming edges and does not belong to the network yet. To reduce the space of networks, number of incoming edges into each node in $\mathcal{G}$ is limited to k. After the construction of DAG $\mathcal{G}$ the parameters $\mathcal{L}$ of BN are estimated (see also 3.3.1) as:

$$p(x_i \mid \boldsymbol{\pi_i}) \approx \frac{m(x_i, \boldsymbol{\pi_i})}{m(\boldsymbol{\pi_i})}, \tag{4.10}$$

where $m(x_i, \boldsymbol{\pi_i})$ denotes the number of simultaneous occurrence of x_i and $\boldsymbol{\pi_i}$ in the parent population, whereas $m(\boldsymbol{\pi_i})$ denotes the number of occurrences of $\boldsymbol{\pi_i}$.

The new individuals are generated using PLS algorithm (see 3.4.2). First, the variables (genes) are ordered in the topological order according to BN and each iteration, the variables whose parents are already determined are generated using the conditional probabilities. This is repeated until all the variables are generated.

The complete BOA evolutionary cycle is shown in the following figure.

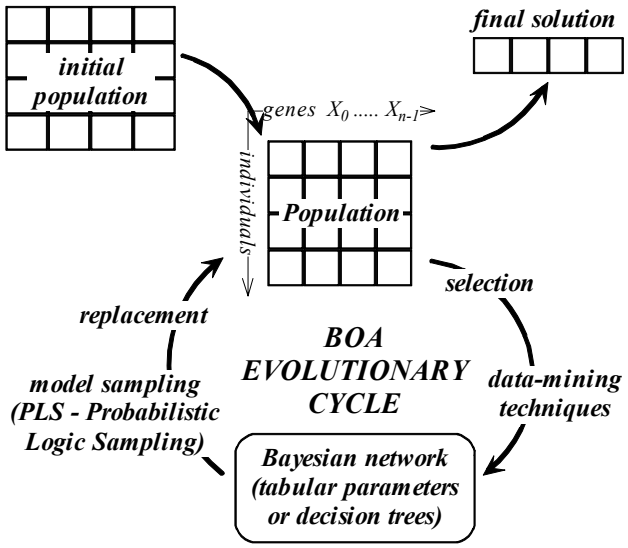


Fig. 4.3: BOA evolutionary cycle

The BOA algorithm attracted much attention during few last years. An empirical comparison between BMDA and BOA can be found in [xiv]. A study on incorporating prior problem-specific information to improve the performance of BOA in graph partitioning can be found in [xi]. For theoretical analysis see [93], [99] or Pelikan's disertation [100].

Also many improvements have been proposed. See [94] and [97] for a modification of the BOA approach in order to model hierarchical problems using so called Huffman network. In [95] a combination of BOA , BMDA and UMDA with clustering techniques is used to optimize symmetrical problems. Very significant advancement is proposed in [96], where BOA is adapted to include binary decision diagrams (BDD) that ensure compact representation of Bayesian networks (for more detailed explanation of BDD advantages see 3.2.2 and for BD metrics with local structure see [24] and chapter 3.3.6).

Other work that uses Bayesian approaches to optimization with EDAs, but in this case the Bayesian network paradigm is not used, is that of Zhang (see [120] and [121]).

A discussion of the use of Bayesian networks as an extension to tree models can also be found in [9]. Another algorithm that uses Bayesian networks to model promising solutions was independently developed by Etxeberria and Larrañaga in [32], who called it the Estimation of Bayesian Network Algorithm (EBNA). Several criteria for guiding the search for good model structures based on different scoring – BIC, K2+penalty as well as on testing conditional (in)dependencies between variables (so called algorithm PC) have been implemented in [68]. For application of the EBNA approach see [10] and [11].

Műhlenbein and Mahnig [79] later improved the original FDA by using Bayesian networks together with the greedy algorithm for learning the networks described above. The modification of FDA was named the Learning Factorized Distribution Algorithm (LFDA). The complexity of learnt model is controlled by the BIC criterion in conjunction with the restriction of the maximum number of incoming edges in BN. Another two variants FDA-BC and FDA-SC have been proposed by Ochoa et. al. in [85] and [86].

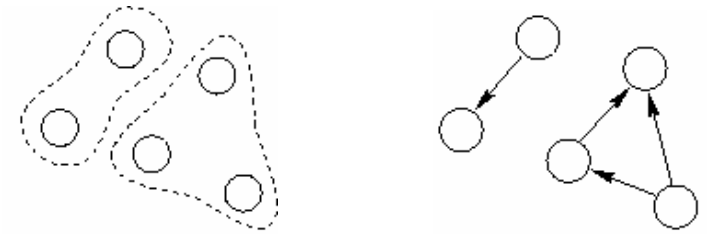


Fig. 4.4*: Example of dependency graph for: a) EcGA, b)BOA/EBNA/LFDA*

4.5 Continuous probabilistic models

There are two basic approaches to extending EDAs for problems with real parameters:

1. Map the problem to the domain of discrete strings, solve the discrete problem, and map the solution back to the problem's original domain.
2. Let the parameters be continuous (like in evolution strategies) and use a proper model for this continuous domain.

The first approach has been used in genetic and evolutionary algorithms for decades, but is not scalable in the EDA framework, since the complexity of required discrete probabilistic model exponentially increases with the required precision of solution. See [98] for example of BOA with fixed-width and fixed-height histogram discretization. This section reviews EDAs based on the second approach.

4.5.1 Without dependencies – $UMDA_c$, SHCLVND, $PBIL_c$

All algorithms discussed in this section perform similarly, since they factorize the joint probability density function as a product of one-dimensional and independent normal densities (see also 3.3.2):

$$f(\boldsymbol{Y} \mid \boldsymbol{\mu}, \boldsymbol{\sigma}) = \prod_{i=0}^{n-1} N(Y_i; \mu_i, \sigma_i) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi}\sigma_i} e^{\frac{1}{2}\left(\frac{Y_i - \mu_i}{\sigma_i}\right)^2}, \tag{4.11}$$

The Univariate Marginal Distribution Algorithm for Continuous domains ($UMDA_c$) was introduced by Larrañaga et. al. in [67] and [69]. For each variable two parameters are estimated from the parent population – the mean value and the standard deviation – using equations (3.17) and (3.18).

Rudlof and Köppen in their SHCLVND (Stochastic Hill Climbing with Learning by Vectors of Normal Distributions, [103]) replace the population of solutions by a vector of mean values of Gaussian normal distribution μ_i for each optimized variable. The standard deviation σ is stored globally and it is the same for all variables. After generating a number of new solutions, the Hebbian learning rule is used to shift the mean values μ_i towards the best of the generated solutions and the standard deviation σ is reduced to make future exploration of the search space narrower.

In [111] Sebag and Ducoulombier implemented $PBIL_C$ – the extension of PBIL to continuous spaces. The adaptation of the vector of means $\boldsymbol{\mu}=\{\mu_0,\ldots,\mu_{n-1}\}$ is done as:

$$\mu_i(t+1) = (1-\alpha)\mu_i(t) + \alpha\left(y_i^{best1} + y_i^{best2} - y_i^{worst}\right), \tag{4.12}$$

where α is the learning parameter and y_i^{best1}, y_i^{best2} and y_i^{worst} denote the i-th coordinate of best, second best and worst individual in the population. Also various heuristics for modifying the σ parameter have been exploited.

Another implementation of real-coded EDA based on interval adaptation can be found in Servet et. al. [112]. This algorithm stores for each variable left and right boundary of its interval and the frequency of occurrence of alleles from the right half of the interval. Each time new solutions are generated using the corresponding intervals, the best solutions are selected and the frequencies are shifted towards them. If these frequencies get close to either 0 or 1, the interval is reduced to the corresponding half of it.

4.5.2 Bivariate dependencies - $MIMIC_c^G$

$MIMIC_c^G$ was introduced by Larrañaga et. al. in [67] and [69]. The underlying probability model for neighbouring pair of variables in the chain is assumed to be a bivariate Gaussian. In the first step of structure learning, the variable with the smallest sample variance is chosen. In following steps, the variables Y_i with the smallest value of

$$\frac{\sigma_{Y_i}^2 \sigma_{Y_j}^2 - \sigma_{Y_i Y_j}^2}{\sigma_{Y_j}^2} \tag{4.13}$$

are linked to the chain (here Y_j denotes the previously linked variable).

4.5.3 Multivariate normal distribution – EMNA

The EMNA approach [70] is based on the direct estimation of a multivariate normal density function at each generation. The vector of means and the variance-covariance matrix are estimated using their straightforward mathematical definition (3.17) and (3.20). There is no need to search for the dependency graph in EMNA, because the resulting probability model is equivalent to Gaussian network with complete dependency graph. However, the straightforward mathematical form of multivariate normal density function is hard to sample and needs to be converted into factorized form.

4.5.4 Gaussian network – EGNA

The Gaussian network is the basic model used by Larrañaga et al. in their EGNA algorithm [67]. The value of continuous parameter X_i is defined by the normal Gaussian probability density function, where the mean value μ_i of X_i is affected by linear combination of "parent" values (see 3.2.3). In the $EGNA_{ee}$ the Gaussian network is induced at each generation by means of edge exclusion tests, whereas in the $EGNA_{BGe}$ the Bayesian score ([38]) is used and in $EGNA_{BIC}$ the BIC score (3.3.4) is used. For more details see [66],[68] and [69]. Application of EGNA can be found in [10].

This simple-peak Gaussian model is not able to effectively describe and process non-linearity and symmetry (the shape of reliable sampling region is elliptical), as it is shown in 3.2.4. Moreover, in EGNA there have been problems with using higher order factorizations, because the multivariate statistical analysis for getting the exact coefficients of Gaussian network is computationally prohibitive.

4.5.5 Gaussian mixtures and kernels – IDEA

The IDEA (Iterated Density Estimation Evolutionary Algorithm) framework was proposed by Bosman and Thierens in [14].

In addition to using one probability density function (like in EGNA) it is also possible to use each solution as a source of elementary normal PDF - the Gaussian kernel (see [19] or 3.2.5 for details). Another models within the IDEA framework are based on the mixture distributions. First, the clustering strategy is applied to divide the samples into clusters and for each cluster one Gaussian network is used. We assume that the space of optimized parameters can be divided into some few regions where the fitness function has only single local extreme. This is more general than usage of the single linear regression for the entire space of parameters. Moreover, efficient EM algorithm can be utilized to search for the mixture effectively. For more details on learning mixture distributions see the explanation in chapter 3.3.8. Some experiments in continuous optimization using IDEA can be found in [16], [17] and [18].

However, the drawback of all clustering techniques (including IDEA) is that different clusters do not share any mutual information. The sampling of solutions in all regions is performed independently, even if solutions from different regions might have some common features and might have the same important building blocks. This becomes more evident in the case of Gaussian kernels, where the linkage information is completely missing and the performance of optimization depends mainly on the density of search space sampling and kernel variance.

4.6 Continuous EDA vs. evolution strategies

Generally, we can state that classical Evolution Strategies (ES) also use normal PDFs. One might therefore say that there already exist algorithms that operate in a similar fashion as the continuous EDAs. In this section, I describe how ES differs from EDAs and point out that there are fundamental differences that distinguish the approaches from one another. My goal is not to go give a description of ES, but to point out the similarities and differences. This section is a summary of more detailed comparison which can be found in [19].

In ES, the parents are recombined and subsequently mutated as is the case for the simple GAs. However, the mutation operator has always been the most important one for ES. This operator samples from normal PDFs and makes the ES appear similar to the EDAs with normal PDFs. With each variable, a normal PDF is associated. Each normal PDF can either be allowed to have identical standard deviations in each dimension, allowed to be aligned with the axes of the search space without necessarily identical standard deviations or allowed to be any arbitrary n-dimensional normal PDF. This comes down to a covariance matrix with respectively either similar entries on the diagonal and zero entries off the diagonal, non-similar entries on the diagonal and zero entries off the diagonal or an arbitrary symmetric covariance matrix.

However, the way in which the normal PDF is found in the next generation is different. In the EDA approach the PDF is estimated from the promising part of population. For the ES approach however, the variables that code the matrix are incorporated into the genome. The direction as well as the length of the step in which we move the normal PDF is subject to evolution itself. This means that we can regard the ES approach as moving the normal PDF in some direction. In other words, the ES in this case can be seen as evolving to find local gradient information on how to best traverse the search space.

Concluding, the two approaches are fundamentally different. This becomes clear for instance on problems in which local information is important, such as Rosenbrock's function, which has a narrow valley. The bottom of this valley has a unique minimum. However, the gradient along the bottom of the valley is very small. EDA based approach will quite easily find the valley, but will not be able to traverse the valley to find the global minimum. The reason for this is that density estimation converges on a certain part of the valley since samples are only available in that part of the search space. On the other hand, once the ES is inside the valley, it can adapt its mutation direction and stepsize to follow the valley to its minimum in a gradient descent fashion. Even though this is a time consuming process, the ES is not as likely to prematurely converge on such a problem as is the EDA approach.

5 Research objectives

5.1 Open problems

There is currently a considerable amount of research surrounding Estimation of Distribution Algorithms. However, the following open problems can be identified:

1. There does not appear to be a complete formal framework for EDAs.
2. The research on general models for continuous domains should be refined – the present models are computationally expensive or too simplified.
3. Computational complexity of model construction significantly restricts the size of solvable problems.
4. Existing EDAs are not applicable to problems with multiple objectives.
5. Modular development system for rapid prototyping of EDA applications is missing.
6. The efficiency of EDAs for real world problems was not investigated – most empirical studies deal only with synthetic benchmarks with bounded degree of epistasis.

5.2 Research outline

This dissertation solves all the open problems stated in previous section. The main goals include :

1. Identification of the common features of existing EDAs and creation of their formal description (chapter 6).
2. Development of new probabilistic model which is effectively applicable to real-valued or even mixed optimization problems (chapter 8).
3. Time analysis of main EDA components for the purpose of algorithmic simplification and parallel implementation. Development of new parallel methods and heuristics providing for realtime performance (chapter 9).
4. Extension of EDA framework for multiobjective optimization (chapter 10).
5. Design and implementation of modular development system for rapid prototyping of EDA applications (chapter 11).
6. Investigation of EDA performance on real world problems, namely from the area of circuit design (throughout all experiments, for example see chapter 7 for definition of hypergraph partitioning benchmark and its multiobjective version).

Chronologically my research can be divided into two epochs. In the first epoch I used for experiments the original Pelikan's implementation of simple BOA algorithm with Bayesian network model (see [91]). A number of modifications to this baseline code have been suggested and its capabilities have been improved according to the established goals. In the second epoch I implemented my own MBOA algorithm with mixed binary decision trees and used it as a basis for further improvements. This subdivision is crucial to clear up the organization of following chapters. Each of the chapters 8-10 is devoted to different goal. Usually in its first subsection the problem is introduced and analyzed, in the second subsection the solution for BOA is proposed and in the last subsection the solution for MBOA is proposed. The detailed guidelines are shown in the following table.

Tab. 5.1. The organization of chapters 8-10

Probability model / **Goal**	**Bayesian network (original Pelikan's BOA core [91])**	**Mixed Binary Decision Trees (my own code)**
Solving continuous & mixed problems	incompetent model for continuous domain	chapter 8.2 (implemented MBOA)
Parallel processing	chapter 9.3 (simulated PBOA) & 9.4 (implemented DBOA)	chapter 9.5 (simulated in Transim)
Multiobjective optimization	chapter 10.2 (implemented Pareto BOA)	chapter 10.3 (implemented BMOA)

6 Theoretical background

6.1 Formal approach to EDAs

The following paragraphs describe my original formal definition of Estimation of Distribution Algorithm. The reasons for formal approaches are straightforward since: (i) problem specification is rigorous, (ii) a mathematical apparatus can be applied to investigate their properties, and (iii) tools for automatic analysis, verification and design can be developed. As the baseline I use Back's formal definition of evolutionary algorithms presented in [2] and shortly summarized in chapter 2.2. I use also some symbols from EDA notation defined in chapter 4.1.

Definition 6.1 (Estimation of Distribution Algorithm) :
An Estimation of Distribution Algorithm is defined as an 11-tuple

$$\text{EDA} = (I, \Phi, \mathcal{M}, \Psi, \rho, \omega, s, r, \iota, \mu, \lambda), \tag{6.1}$$

where:
(i) $I=A^n$ is the space of individuals of length n over the domain A,
(ii) $\Phi: I \to \mathbb{R}$ denotes a fitness function assigning real values to individuals (in EDA framework often denoted as $F(\boldsymbol{x})$),
(iii) $\mathcal{M} = \{\mathrm{M} \mid \mathrm{M} : I \to [0,1]\}$ is the space of admissible probabilistic models,
(iv) $\Psi : I^{\mu+\lambda} \to I^{\mu+\lambda}$ is the generation transition function,
(v) $\rho : I^{\mu} \times \mathcal{M} \to \mathbb{R}$ denotes the metrics for evaluating probabilistic models,
(vi) $\omega : \mathcal{M} \to I^{\lambda}$ denotes the model sampling operator (non-deterministic),
(vii) $s : I^{\mu+\lambda} \to I^{\mu}$ is the (μ+λ)-selection operator, which selects the parent population $D(t)$ from population $P(t)$. The most frequent truncation selection operator is defined as $s_{(\mu+\lambda)}(P(t))=D(t)$ such that: $\exists \boldsymbol{x} \in P(t) \setminus D(t) : (\exists \boldsymbol{x'} \in D(t) : \Phi(\boldsymbol{x}) > \Phi(\boldsymbol{x'}))$,
(viii) $r : I^{\mu+\lambda+\lambda} \to I^{\mu+\lambda}$ is the replacement operator, which creates new population $P(t+1)$ from the old population $P(t)$ and offspring population $O(t)$. The most frequent replace-worst operator is defined as $P(t+1) = r_{(\mu+\lambda+\lambda)}(P(t),O(t)) = s_{(\mu+\lambda)}(P(t)) \cup O(t)$,
(ix) $\iota : I^{\mu+\lambda} \to \{true, false\}$ is the termination criterion,
(x) μ is the number of parent individuals (often denoted as N), while λ denotes the number of offspring individuals (usually equal to N, so $L = \lambda+\mu = 2*N$).

Definition 6.1 is based on high-level concept, where each probabilistic model M is a mapping from the space of strings to the space of probability values. Since M specifies some probability distribution, it must hold

$$\sum_{\boldsymbol{x} \in I} M(\boldsymbol{x}) = 1 \tag{6.2}$$

This concept is also extendable for continuous domains $I \subseteq \mathbb{R}^n$, where M specifies the probability density function:

$$\int_{\boldsymbol{y} \in I} M(\boldsymbol{y}) d\boldsymbol{y} = 1 \tag{6.3}$$

Definition 6.2 (Optimal choice of model) :
A choice of model $M^* \in \mathcal{M}$ is optimal with respect to the parent population $D(t) \in I^\mu$ and metrics $\rho : I^\mu \times \mathcal{M} \to \mathbb{R}$, iff

$$M^* = \arg\max_{M \in \mathcal{M}} \rho(D(t), M) \tag{6.4}$$

Note that without loss of generality I considered a metrics which should be maximized.

Definition 6.3 (Competent sampling of model) :
Assume that there exists some arbitrary ordering among the genes from I (binary relation "<") and that $\mathcal{F}_M(\boldsymbol{x})$ denotes cumulative probability distribution with respect to the model M:

$$\mathcal{F}_M(\boldsymbol{x}) = \sum_{\substack{\forall \boldsymbol{x}' \in I \\ \boldsymbol{x}' < \boldsymbol{x}}} M(\boldsymbol{x}') \tag{6.5}$$

or cumulative probability density function

$$\mathcal{F}_M(\boldsymbol{y}) = \int_{\text{"0"}}^{\boldsymbol{y}} M(\boldsymbol{y}') d\boldsymbol{y}', \tag{6.6}$$

where "0" denotes infimum of all genes from I with respect to the relation "<". The competent model sampling operator is a non-deterministic function $\omega : \mathcal{M} \to I^\lambda$ defined as

$$\omega(M) = \mathcal{F}_M{}^{-1}(\chi), \tag{6.7}$$

where $\chi \in [0,1]$ denotes an uniform random variable sampled anew for each call of sampling operator. Also the random generation of initial population can be described as the competent sampling of uniform model, which has the following property

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in I : M_{uniform}(\boldsymbol{x}) = M_{uniform}(\boldsymbol{x}') = \frac{1}{|I|}. \tag{6.8}$$

Thus, competent sampling operator $\omega(M)$ generates individuals with frequency asymptotically proportional to their probability in M. For example Probabilistic Logic Sampling (PLS, see 3.4.2) is competent sampling operator, but Gibbs sampling (see 3.4.1) is not – it only approximates the PLS.

Definition 6.4 (Offspring population) :
A population $O(t) \in I^\lambda$ obtained by repetitive sampling of the optimal model of parent population $D(t) \in I^\mu$ is called offspring population:

$$O(t) = \left\{ \boldsymbol{x}^0, \ldots, \boldsymbol{x}^{\lambda-1} \mid \boldsymbol{x}^i = \omega\left(\arg\max_{M \in \mathcal{M}} \rho(D(t), M) \right) \right\} \tag{6.9}$$

Definition 6.5 (Generation transition function of EDA algorithm) :
Given an EDA algorithm with replacement operator $r : I^{\mu+\lambda+\lambda} \to I^{\mu+\lambda}$, the function $\Psi : I^{\mu+\lambda} \to I^{\mu+\lambda}$ is called generation transition function of EDA algorithm iff

$$\Psi(P(t)) = r(P(t), O(t)), \tag{6.10}$$

where $P(t) \in I^{\mu+\lambda}$ is the population of individuals in time t, and $O(t) \in I^\lambda$ is its corresponding offspring population from definition 6.4. Using all the EDA components, Ψ can be written as:

$$\Psi(P(t)) = r\left(P(t), \omega^\lambda\left(\arg\max_{M \in \mathcal{M}} \rho(s(P(t)), M) \right) \right). \tag{6.11}$$

Definition 6.6 (Population sequence of EDA algorithm) :
Given an EDA algorithm with generation transition function $\Psi : I^{\mu+\lambda} \to I^{\mu+\lambda}$ and an initial population $P(0) \in I^{\mu+\lambda}$, the sequence $P(0)$, $P(1)$, $P(2)$,… is called a population sequence or evolution of $P(0)$ iff

$$\forall t \geq 0 : P(t+1) = \Psi(P(t)) \tag{6.12}$$

Definition 6.7 (Running time of EDA algorithm) :
Given an EDA algorithm with generation transition function $\Psi : I^{\mu+\lambda} \to I^{\mu+\lambda}$ and an initial population $P(0) \in I^{\mu+\lambda}$, the time T is called running time of EDA algorithm iff

$$T = \min\left\{ t \in \mathrm{N} \mid t\left(\Psi^t(P(0))\right) = true \right\}. \tag{6.13}$$

Like in the case of classical GAs, the termination is often based on bias measurement (see equations 2.4 and 2.5).

Definition 6.8 (Result of EDA algorithm) :
Given an EDA algorithm with generation transition function $\Psi : I^{\mu+\lambda} \to I^{\mu+\lambda}$ and an initial population $P(0) \in I^{\mu+\lambda}$, the chromozome $\boldsymbol{x}^* \in I$ is called a result of EDA algorithm iff

$$\boldsymbol{x}^* = \arg\min \Phi(\boldsymbol{x}), \quad \text{for every } \boldsymbol{x} \in \bigcup_{t=0}^{T} \Psi^t(P(0)). \tag{6.14}$$

Definition 6.9 (Bayesian network) :
Bayesian network is a pair $B=(\mathcal{G},\mathcal{L})$, where $\mathcal{G}=(V,E)$ is a Directed Acyclic Graph (DAG) with $V=\{0,\ldots,n\text{-}1\}$, $E \subseteq V \times V$; and $\mathcal{L}$ is a set of local CPDs

$$\mathcal{L} = \{ \mathcal{L}_i \mid \mathcal{L}_i : A \times A^{|\boldsymbol{\Pi}_i|} \to [0,1],\ i \in \{0,\ldots,n\text{-}1\} \} \tag{6.15}$$

where $\boldsymbol{\Pi}_i = \{X_j \mid (j,i) \in E\}$.

Since each $\mathcal{L}_i$ specifies local conditional probability distribution $p(X_i|\boldsymbol{\Pi}_i)$, it must hold

$$\forall \boldsymbol{\pi}_i \in Dom(\boldsymbol{\Pi}_i) : \sum_{\forall x_i \in Dom(X_i)} \mathcal{L}_i(x_i, \boldsymbol{\pi}_i) = 1 \tag{6.16}$$

For EDA algorithms with simpler models (UMDA, BMDA, etc.) the formal definition can be easily obtained by modifying the domain of $\mathcal{L}_i$ and by specifying additional constraints for E. For example in the case of BMDA we allow only $|\boldsymbol{\Pi}_i| \leq 1$, such that

$$\mathcal{L}_i : \begin{cases} A \to [0,1] & if\ \boldsymbol{\Pi}_i = \varnothing \\ A \times A \to [0,1] & if\ |\boldsymbol{\Pi}_i| = 1 \end{cases}$$

EDA algorithms with decision graphs (resp. trees) can be formally defined after introducing some reflexive, symmetrical and transitive binary relation $\varphi_i \subseteq A^{|\boldsymbol{\Pi}_i|} \times A^{|\boldsymbol{\Pi}_i|}$ (equivalence) for each decision graph, such that $\forall \boldsymbol{u}, \boldsymbol{v} \in A^{|\boldsymbol{\Pi}_i|} : \boldsymbol{u}\ \varphi_i\ \boldsymbol{v} \Leftrightarrow \boldsymbol{u}, \boldsymbol{v}$ traverse to the same leaf. These equivalence constraints imply from context-specific independence statements (CSI, see 3.2.2). Now $\mathcal{L}_i$ can be defined as

$$\mathcal{L}_i : A \times \left(A^{|\boldsymbol{\Pi}_i|} / \varphi_i \right) \to [0,1], \tag{6.17}$$

where $A^{|\boldsymbol{\Pi}_i|} / \varphi_i$ is a quotient set containing one element for each φ_i-equivalence class from the set $A^{|\boldsymbol{\Pi}_i|}$.

You see that expression (6.17) does not define more general probability distribution than (6.15), but it requires less parameters which can be (on average) estimated more robustly – the reasons for preferring decision graphs are quantitative.

Theorem 6.1 (Model factorization) :
Each probabilistic model M (for discrete domains) can be expressed in the factorized form as Bayesian network B=($\mathcal{G}$,$\mathcal{L}$), such that

$$\forall \boldsymbol{x} \in I : M(\boldsymbol{x}) = \prod_{i=0}^{n-1} \mathcal{L}_i(x_i, \boldsymbol{\pi}_i) \tag{6.18}$$

Proof: Using the probability chain rule each discrete joint probability function can be factorized for example as

$$p(X_0, X_1, ..., X_{n-1}) = \prod_{i=0}^{n-1} p(X_i \mid X_0, X_1, ..., X_{i-1}) = \prod_{i=0}^{n-1} \frac{p(X_0, X_1, ..., X_{i-1}, X_i)}{p(X_0, X_1, ..., X_{i-1})} \tag{6.19}$$

The tabular representation of $\mathcal{L}_i$ can be obtained from the i-th fraction by assigning instances to variables $X_0,...,X_i$. In the case of low-epistatic problems the dependency graph can be further „sparsed“ by conditional independence tests (X_j can be omitted from $\boldsymbol{\Pi}_i$ iff $p(X_i \mid \boldsymbol{\Pi}_i) = p(X_i \mid \boldsymbol{\Pi}_i \setminus X_j)$).

Note that the same theorem does not hold for Gaussian network, because it is not guaranteed that the factored local terms fit the linear regression model.

6.2 Convergence problems and possible EDA improvements

This chapter analyzes the role of probabilistic model in EDA algorithm and proposes future improvements which imply from relation between probabilistic model and other parts of EDA. The goal is to provide qualitative description of EDA convergence under some assumptions, not to provide quantitative analysis, which can be found elsewhere (see papers [93] and [99]).

6.2.1 Model accuracy

The key issue in EDAs is the role of accuracy of probabilistic model construction. To clarify it, I formulated the following theorem:

Theorem 6.2 (Distribution preservation) :
If the EDA algorithm fulfills the following conditions, then the distribution of individuals in the generated offspring population approaches the distribution of individuals in the parent population.

The sufficient (but not necessary) conditions are:

1. All possible probabilistic models are admissible
2. Entropy-based metrics is used for model selection (for example DL_{data} from MDL)
3. No model complexity penalty is used ($DL_{tab} = DL_{graph} = 0$)
4. Competent sampling operator is used
5. Population is sufficiently large

Proof: Let $p_{D(t)}(\boldsymbol{X})$ denote the distribution of individuals in the parent population $D(t)$. The optimal model is selected as

$$M^* = \arg\min_{M \in \mathcal{M}} DL_{data}(D \mid M) = \arg\min_{M \in \mathcal{M}} \left(-\sum_{j=0}^{N-1} \log M(\boldsymbol{x}^j) \right)$$

The population contains N. $p_{D(t)}(\boldsymbol{x})$ copies of individual $\boldsymbol{x}$, thus

$$M^* = \arg\min_{M \in \mathcal{M}} \left(-N. \sum_{\forall \boldsymbol{x} \in Dom(\boldsymbol{X})} p_{D(t)}(\boldsymbol{x}) \log M(\boldsymbol{x}) \right)$$

Assume that $\forall \boldsymbol{x} : M(\boldsymbol{x}) > 0$, such that log(0) error does not occur. Since all probabilistic models are admissible, in $\mathcal{M}$ there must exist a model M^*, for which the sum gives minimum (from the Gibbs theorem it follows that

$$-\sum_{\forall \boldsymbol{x} \in Dom(\boldsymbol{X})} p_{D(t)}(\boldsymbol{x}) \log p_{D(t)}(\boldsymbol{x}) \leq -\sum_{\forall \boldsymbol{x} \in Dom(\boldsymbol{X})} p_{D(t)}(\boldsymbol{x}) \log M(\boldsymbol{x}),$$

where the equality holds iff $\forall \boldsymbol{x} : p_{D(t)}(\boldsymbol{x}) = M(\boldsymbol{x})$). Thus M^* is either identical to $p_{D(t)}(\boldsymbol{X})$ or contains the decomposition of $p_{D(t)}(\boldsymbol{X})$ with the correct independence assertions. By its competent sampling EDA generates offspring population $O(t)$.

$$p_{O(t)}(\boldsymbol{X}) \approx M^*(\boldsymbol{X}) = p_{D(t)}(\boldsymbol{X})$$

Because this sampling is not deterministic, the offspring population size needs to be sufficiently large in order that the equality $p_{O(t)}(\boldsymbol{X}) = p_{D(t)}(\boldsymbol{X})$ holds.

This theorem confirms that EDA algorithm with overfitted probabilistic model is unable to introduce new genetic material. This is one extreme, analogical to pure reproduction scenario for GAs (see 2.3.1). Another extreme is EDA algorithm with very rough probabilistic model, which massively explores the new search space. Obviously, the tradeoff between exploration and exploitation of search space is controlled by accuracy of probabilistic model construction. Simple models are less conservative, allow for escaping from local extremes, but are more sensitive to benchmark deceptiveness.

Concluding, a model M that perfectly fits the data D (i.e. an apparently optimal solution) may be less desirable than one that does not fit the data. This is known as *overfit* or *overspecialization*. The world is inherently simple. Therefore the smallest model that is consistent with the samples is the one that is most likely to deal with unknown solutions correctly. The problem of trading off the simplicity of a model with how well it fits the training data is a well-studied problem. In statistics this is known as the *bias-variance tradeoff*, in Bayesian inference it is known as *penalized likelihood*, and in pattern recognition/machine learning it manifests itself as the *minimum message length* (MML) or *minimum description length* problem (MDL).

6.2.2 Fitness nonlinearity checking

Very significant improvement of EDA algorithm can be proposed after investigating the relationship between the probability which is assigned to each individual by the probabilistic model and the fitness of this individual.

Consider the simple EDA algorithm:

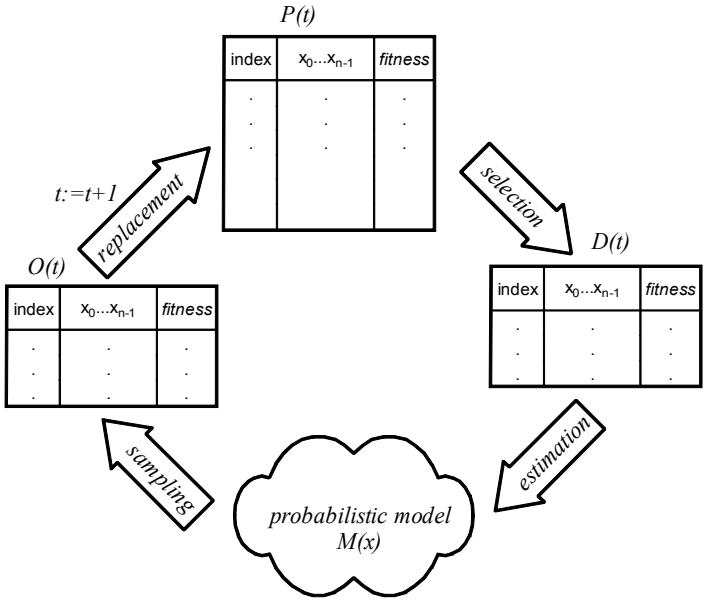


Fig. 6.1*: Illustration of simple EDA cycle.*

Without loss of generality let us assume that the parents are selected by fitness-proportional selection (but it can be shown that similar conclusion will be obtained for truncation selection, tournament selection, etc.). In the first generation the individuals for $D(0)$ are selected proportionally from the initial population $P(0)$:

$$p(\boldsymbol{x} \in D(0) \,|\, \boldsymbol{x} \in P(0)) = \frac{F(\boldsymbol{x})}{\sum_{\forall \boldsymbol{x}' \in P(0)} F(\boldsymbol{x}')}. \tag{6.20}$$

Again, let us focus on the ideal case with overfitted model. From previous chapter we know that under this pure reproduction scenario the probability distribution $M(x)$ is equal to distribution of individuals in $D(t)$:

$$M_{t=0}(\boldsymbol{x}) = p(\boldsymbol{x} \in D(0) \,|\, \boldsymbol{x} \in P(0)) = \frac{F(\boldsymbol{x})}{\sum_{\forall \boldsymbol{x}' \in P(0)} F(\boldsymbol{x}')}. \tag{6.21}$$

It the offspring population is of sufficient size, it also holds

$$p(\boldsymbol{x} \in O(0) \,|\, \boldsymbol{x} \in P(0)) = M(x) \tag{6.22}$$

For simplicity, let us assume that the replacement is defined as $P(t+1) = O(t)$, which is quite common for simple GA without elitism.

Then, in second generation it holds:

$$M_{t=1}(x) = p(\boldsymbol{x} \in D(1) \,|\, \boldsymbol{x} \in P(0)) = \frac{p(\boldsymbol{x} \in D(0) \,|\, \boldsymbol{x} \in P(0)).F(\boldsymbol{x})}{\sum_{\forall \boldsymbol{x}' \in P(0)} p(\boldsymbol{x}' \in D(0) \,|\, \boldsymbol{x}' \in P(0)).F(\boldsymbol{x}')} =$$

$$= \frac{F^2(\boldsymbol{x})}{\sum_{\forall \boldsymbol{x}' \in P(0)} F(\boldsymbol{x}') \sum_{\forall \boldsymbol{x}' \in P(0)} \frac{F(\boldsymbol{x}')}{\sum_{\forall \boldsymbol{x}'' \in P(0)} F(\boldsymbol{x}'')} F(\boldsymbol{x}')} = \frac{F^2(\boldsymbol{x})}{\sum_{\forall \boldsymbol{x}' \in P(0)} F^2(\boldsymbol{x}')} \tag{6.23}$$

Generally, in generation number t the ideal probability distribution should correspond to t iterations of proportional selection operator:

$$M_t(x) = p(\boldsymbol{x} \in D(t) \mid \boldsymbol{x} \in P(0)) = \frac{F^t(\boldsymbol{x})}{\sum_{\forall \boldsymbol{x}' \in P(0)} F^t(\boldsymbol{x}')} = c.F^t(\boldsymbol{x}), \tag{6.24}$$

where c is constant for given t. It can be shown, that under pure reproduction scenario the model of EDA algorithm converges to one individual with the highest fitness value when t approaches infinity.

This equation can be generalized for schema processing - we can view every schema as having a fitness value, which is defined as the average fitness of its instances. Provided that schemas are selected with a probability proportional to their fitness, and reproduced by sampling of overfitted model, the probability of schema H in generation t is again:

$$M_t(H) = c.F^t(H) \tag{6.25}$$

The significance of this equation is remarkable. Every computation with probabilities in EDAs can be converted into computation with fitness values! I strongly believe that future EDAs should use the fitness values for linkage learning. The present EDA algorithms discover linkage by statistical analysis methods, but the required population size is large. The fitness nonlinearity checking is supposed to be more accurate and should work on smaller populations.

6.2.3 Bayesian evolution

From equation (6.21) you see that under pure reproduction scenario the probability distribution of individual $\boldsymbol{x}$ is conditioned on its occurrence in the initial population $P(0)$. This effect can be also described using the Bayesian framework. See the following figure which provides more intuitive view of meaning of probabilistic model.

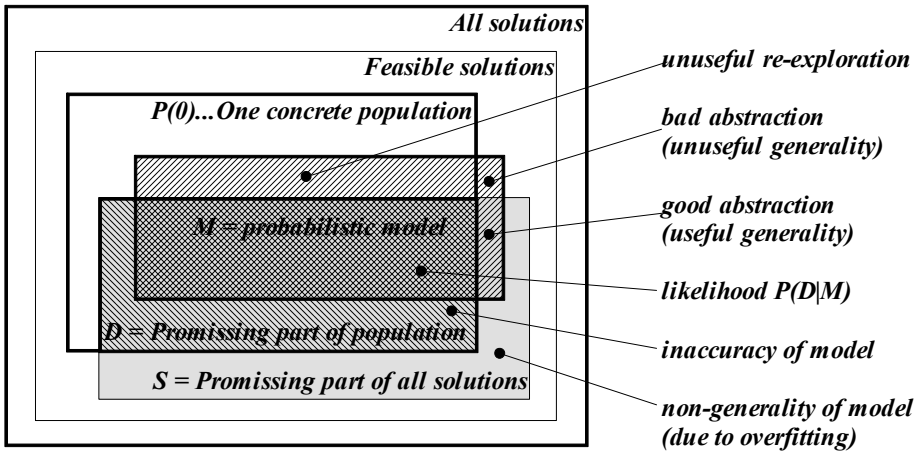


Fig. 6.2: The role of probabilistic model within EDA framework. Without loss of generality the situation in the first generation is analyzed. Note that the sets of individuals are shown with solid boundary, but in real world the boundaries are probabilistic or fuzzy.

First, the random initial population $P(0)$ is generated from the set of feasible solutions. Then, the selection operator takes $P(0)$ and it returns its superior part denoted *D(0)*. The Estimation of Distribution Algorithm then tries to build the probabilistic model *M* for dataset *D(0)*, using well known Bayesian or non-Bayesian methods adopted from the area of data mining and knowledge discovery.

The key issue is the relation between the set of individuals generated by sampling model *M* and the set *S* which denotes the globally superior part of feasible solutions. In the desired case both sets are equal. But in the real case the complete set *S* is not available, only its concrete subset *D(0)* obtained from concrete $P(0)$. We would like EDA to abstract the common properties of solutions from *D(0)* and to include them into estimated model *M*, assuming that these properties are the ones that make the fitness of solutions in *S* superior.

Thus, the key issue in EDA algorithms is to estimate and sample the distribution $p(\boldsymbol{x} \in S)$ as reliably as possible, independently of the optimization environment. For example in the first generation it should be independent of the choice of initial population:

$$p(x \in S) = \sum_{\forall P(0)} p(x \in D(0) \mid x \in P(0)).p(x \in P(0)) \tag{6.26}$$

This brings us into Bayesian character of evolutionary process itself, which was firstly recognized by Gottvald in [43]. He proposed that the result of each optimization step should be seen only within the context of previous optimization steps performed, using the Bayesian theorem to recursively update the knowledge about the problem. I believe that this concept of Bayesian evolution enables future improvements of EDA algorithms.

6.2.4 Solution encoding

It is desirable that the encoding makes the representation as robust as possible. This means that even if a piece of the representation/string is randomly changed, it will still represent a similar individual. This is the key property determining the effectivity of linkage identification. Intuitively, it is unreasonable to look for some dependencies in the genotype domain if similar genotypes are decoded into phenotypes of completely distinct meaning and quality.

7 Selected benchmarks

7.1 Hypergraph partitioning

The decomposition of the system such as communication networks, complex data base system and VLSI layout are usual tasks to be solved. These tasks can be often formalized as a partitioning of hypergraph into k partitions – it is a well known problem of graph theory. Throughout the experiments I have investigated a special case of k-way partitioning for $k=2$, called bisectioning. If necessary the k-way partitioning can be found by recursive 2-way bisectioning (see [xii]).

The particular bisectioning problem is defined as follows: Let us assume a hypergraph $H=(V,E)$, with $n = |V|$ nodes and $m = |E|$ edges. We are supposed to find that bisection (V_1,V_2) of V into equal sized parts ($|V_1|=|V_2|$, $V=V_1 \cup V_2$, $V_1 \cap V_2=\varnothing$) that minimizes the number of hyperedges that simultaneously coincide with both sets V_1, V_2. Such a set of external hyperedges can be denoted as $E_{cut}(V_1,V_2)$. The primary objective function/cost of the hypergraph bisectioning is the number of external hyperedges, C1, shortly called cut size:

$$C1(V_1,V_2) = |E_{cut}(V_1,V_2)| = |\{e \in E \mid e \cap V_1 \neq \varnothing \ \wedge \ e \cap V_2 \neq \varnothing\}| \tag{7.1}$$

Each solution of the bisection is encoded as a binary string $\boldsymbol{x}=(x_0, x_1, \ldots, x_{n-1})$. The value of x_i represents the partition number, the index i specifies the node number. For the case of simple graph bisectioning we have proposed the following cut size calculation:

$$C1 = \sum_{\substack{i=0 \\ j>i}}^{n-1} r_{ij}(x_i + x_j - 2x_i x_j) \tag{7.2}$$

with the balance constraint

$$\sum_{i=0}^{n-1} x_i = \sum_{i=0}^{n-1} (1 - x_i) \quad , \tag{7.3}$$

where the coefficient $r_{ij} \in \mathbb{R}$ equals to one in case the net/edge exists between node i and j, otherwise $r_{ij} = 0$. We are also able to deal with multigraphs, by setting r_{ij} according to the degree of edge between i and j.

In case of k-way partitioning the equation for C1 is not binary, because the string is alphabetical:

$$C1 = \sum_{\substack{i=0 \\ j>i}}^{n-1} r_{ij} F_{equal}(x_i, x_j); \qquad F_{equal}(x_i, x_j) = \begin{cases} 1 & if \ x_i = x_j \\ 0 & otherwise \end{cases} \tag{7.4}$$

In case of hypergraphs the useful term for cost is quite complex, so we use the direct enumeration of external edges instead.

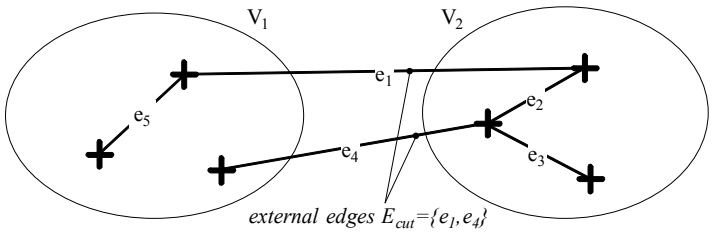


Fig. 7.1*: An illustration of graph bisection with C1=2.*

All the sampling operators (see chapter 3.4) used in EDAs produce individuals with arbitrary balance. To satisfy the balance constraint (7.3), a repair operator has to be used. It normalizes each individual to keep its balance in the considered bound. Naturally, this operation can partly change the cut size, but this effect does not cause an extra genetic drift of the population, because for each individual the genes are selected for normalization randomly. Furthermore, this random normalization does not introduce any statistical bias to the population, so the model estimation algorithm is not confused.

Four sets of graph structures are used as benchmarks:

1. Regular graphs [110] with grid structure (mostly square), denoted by Grid*n*, where *n* specifies the number of nodes. Graphs with the horizontal bottle-neck in the middle are denoted Grid*n.c*, where *c* specifies the bottleneck's minimal cut size. The regular graphs with bottleneck appear to be a proper test benchmarks with many local optima and one known global optimum.

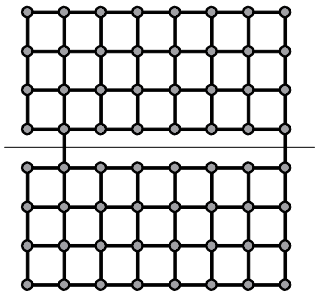


Fig. 7.2*: An illustration of regular graph Grid64.2 with 2-edges horizontal bottleneck in the dashed cut line.*

2. Hypergraphs representing real circuits, denoted by IC*n*. The global optima is not known. The structure of circuits can be characterized as a random logic. The hypergraph IC67 consists of 67 nodes and 134 edges/nets; IC116 consists of 116 nodes and 329 edges/nets; and IC151 consists of 151 nodes and 419 edges/nets. Also a fractal circuit Fract149 belongs to the hypergraphs, it consists of 149 nodes and 147 edges/nets.
3. Geometric graphs, denoted by *Un.d*. These graphs are generated randomly - *n* vertices are placed in the unit square and their coordinates are chosen uniformly. An edge exists between two vertices if their Euclidean distance is *l* or less, where the average expected vertex degree is specified by $d=n\pi l^2$.

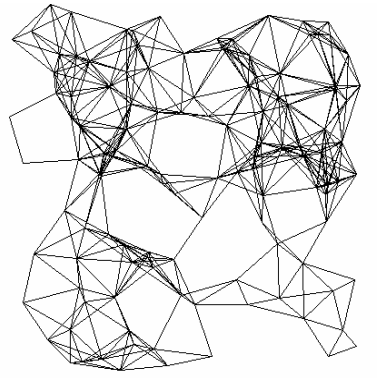


Fig. 7.3: An example of geometric graph - U120.5.

4. Caterpillar graphs *CATk_n*, with *n* nodes, *k* articulations and *(n-k)/k* legs for each articulation.

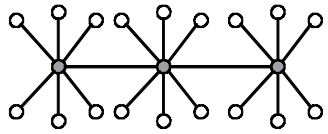


Fig. 7.4: An illustration of caterpillar graph - CAT3_21.

7.2 Multiobjective hypergraph partitioning

For some experiments with multiobjective BOA algorithm a multiobjective hypergraph partitioning benchmark was used. It is an extension of hypergraph partitioning, where the balance constraint is replaced by second cost function C2, shortly called "balance":

$$C2(V_1,V_2) = |V_1| - |V_2| \tag{7.5}$$

For binary encoded solutions $X=(x_0, x_1, \ldots, x_{n-1})$ of hypergraph bisectioning the following form of C2 function was derived:

$$C2 = \left| \sum_{i=0}^{n-1} xi - \sum_{i=0}^{n-1} (1 - xi) \right| \tag{7.6}$$

7.3 Multiobjective knapsack problem

In the field of multi-objective optimization the 0/1 knapsack problem has become a standard benchmark. Results of several comparative case studies are available in the literature, accompanied by test data through the Internet.

Generally, the 0/1 knapsack problem consists of set of items, weight and profit associated with each item, and an upper bound of the capacity of the knapsack. The task is to find a subset of items which maximizes the sum of the profits in the subset, yet all selected items fit into the knapsack so as the total weight does not exceed the given capacity.

This single objective problem can be extended to multiobjective problem by allowing more than one knapsack:

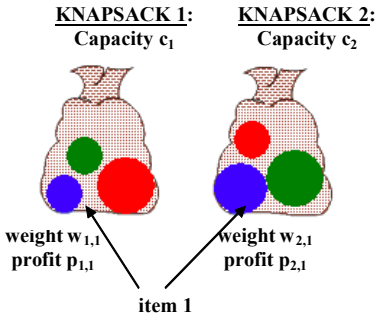


Fig. 7.5: An illustration of multiobjective knapsack problem.

Formally, the multiobjective 0/1 knapsack problem is defined in the following way: Given a set of n items and a set of m knapsacks, with

$p_{i,j}$ profit of item j according to knapsack i

$w_{i,j}$ weight of item j according to knapsack i

c_i capacity of knapsack i

find a vector $\boldsymbol{x} = (x_0,\ldots,x_{n-1}) \in \{0,1\}^n$, such that $x_j = 1$ iff item j is selected and $\boldsymbol{F}(\boldsymbol{x})=(F_1(\boldsymbol{x}),\ldots,F_m(\boldsymbol{x}))$ is maximum, where

$$F_i(\boldsymbol{x}) = \sum_{j=0}^{n-1} p_{i,j} * x_j \tag{7.7}$$

and for which the constraint is fulfilled

$$\forall i \in \{1,2\ldots,m\}: \sum_{j=0}^{n-1} w_{i,j} * x_j \le c_i \tag{7.8}$$

The complexity of the problem solved depends on the values of knapsack capacity. According to [123] I used the knapsack capacities stated by the equation:

$$c_i = 0.5 \sum_{j=0}^{n-1} w_{i,j} \tag{7.9}$$

The encoding of solution into chromosome is realized by binary string of the length n. To satisfy the capacity constraints it is necessary to use repair mechanism on the generated offspring to be the feasible one. The repair algorithm removes items from the knapsack step by step until the capacity constraints are fulfilled. The order in which the items are deleted is determined by the maximum profit/weight ratio per item; the maximum profit/weight ratio q_j is given by the equation

$$q_j = \max_{j=1}^{m} \frac{p_{i,j}}{w_{i,j}} \tag{7.10}$$

The items are considered in increasing order of the q_j, i.e. item with the lowest profit per weight unit is removed first. This mechanism respects the capacity constraints while decreasing the overall profit as little as possible.

7.4 Additively decomposable functions with binary parameters

A OneMax function is a function whose value depends only on the number of ones in an input string. The function values for the strings with the same number of ones are equal:

$$F_{OneMax}(x_0,...,x_{n-1}) = \sum_{i=0}^{n-1} x_i \tag{7.11}$$

The other additively decomposable discrete functions are created by concatenating basis functions of a small order. The contributions of all the functions are added together to determine the overall fitness. Let us have a function F_k defined for strings of length k. Then, the function additively composed of l functions F_k is defined as

$$F(\boldsymbol{X}) = \sum_{i=0}^{l-1} F_k(S_i) \tag{7.12}$$

where $\boldsymbol{X}$ is the set of n variables and S_i for $i \in \{0,\ldots,l\text{-}1\}$ are subsets of k variables from $\boldsymbol{X}$. Sets S_i can be either overlapping or non-overlapping and they can be mapped onto a string (the inner representation of a solution) so that the variables from one set are either mapped close to each other or spread throughout the whole string. Each variable will be required to contribute to the function through some of the subfunction. A function composed in this fashion is clearly additively decomposable of the order of the subfunctions it was composed with.

A deceptive function of order 3, denoted by 3-deceptive, is composed of separable building blocks of order 3, each with a fitness contribution of

$$F_{3deceptive}(u) = \begin{cases} 0.9 & if\ u = 0 \\ 0.8 & if\ u = 1 \\ 0 & if\ u = 2 \\ 1 & otherwise \end{cases}, \tag{7.13}$$

where u is the number of ones in the building block. The overall fitness is computed by summing the contributions of individual BBs.

The trap function of order 5, denoted by trap-5, is defined as the sum of contributions of distinct building blocks of length n=5, each of which is given by

$$F_{trap5}(u) = \begin{cases} 4-u & if\ u < 5 \\ 5 & otherwise \end{cases} \tag{7.14}$$

where u is the number of ones in the building block. Both functions have one global optimum in 111 …1 and a deceptive attractor in 000 …0.

Tab. 7.1. Segments of 3-deceptive function

Triplet	000	001	010	011	100	101	110	111
$F_{3deceptive}$	0,9	0,8	0,8	0	0,8	0	0	1

On the complete table of $F_{3deceptive}$ values we can demonstrate its deceptivity. The average fitness for schema "0**" is (0.9+0.8+0.8+0.4)/4=0.625, but the average fitness for schema "1**" is (0.8+0+0+1)/4=0.45. Thus, the classical GA prefers the schema "0**", even if the schema "1**" leads to global optimum.

7.5 Functions with continuous arguments

In chapter 8 some well-known continuous benchmarks are used, such as Griewank, Michalewicz and Rosenbrock function, see [19]. Their definition is given in Tab. 7.2, all of the test functions should be minimized. For testing I used benchmark instances with n=10 parameters, the optimization stops when a maximum precision of 6 digits is reached.

Tab. 7.2. Continuous benchmarks

Name	Definition	Domain
Griewank	$\frac{1}{4000}\sum_{i=0}^{n-1}(y_i-100)^2-\prod_{i=0}^{n-1}\cos\left(\frac{y_i-100}{\sqrt{i+1}}\right)+1$	**[-5,5]^n**
Michalewicz	$-\sum_{i=0}^{n-1}\sin(y_i)\sin^{20}\left(\frac{(i+1){y_i}^2}{\pi}\right)$	**[0,π]^n**
Rosenbrock	$\sum_{i=0}^{n-2}100(y_{i+1}-y_i^2)^2+(1-y_i)^2$	**[-5.12,5.12]^n**

See also Fig. 7.5 – Fig. 7.7 where a fitness landscape of benchmark instances with n=2 parameters is shown.

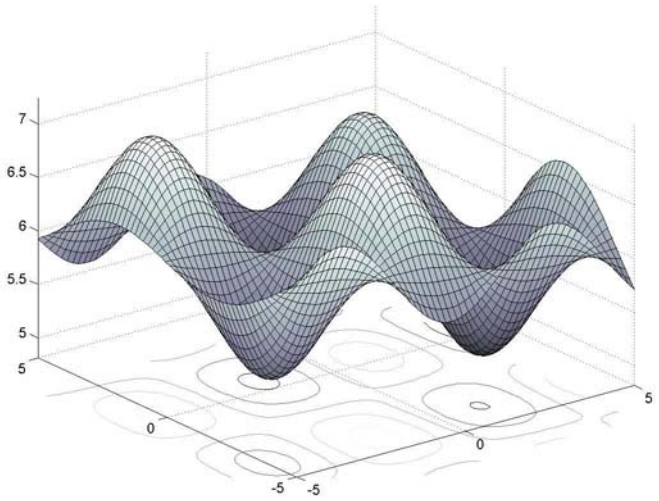


Fig. 7.6: 2-dimensional Griewank function.

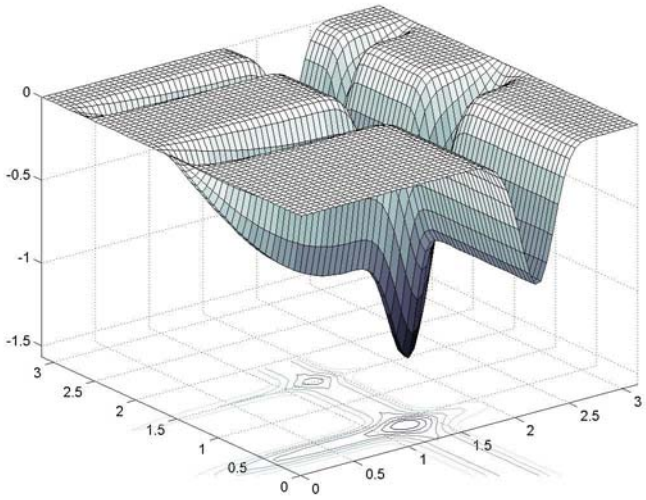


Fig. 7.7: 2-dimensional Michalewicz function.

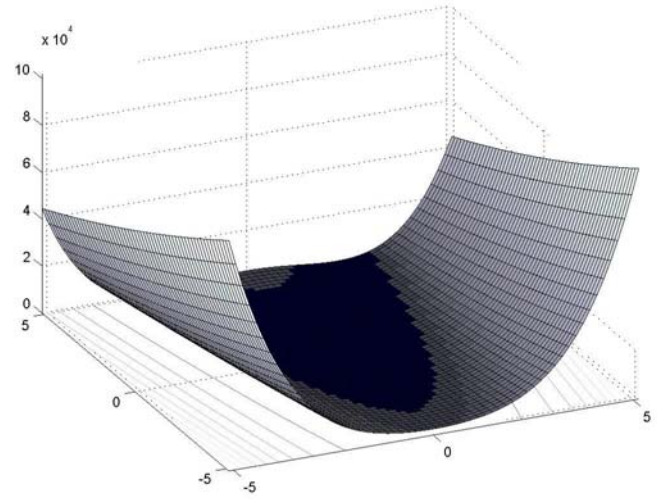


Fig. 7.8: 2-dimensional Rosenbrock function.

An analogy of discrete 2-deceptive function is the continuous 2-deceptive function, defined as

$$F_{2continuous}(y_0, y_1, ..., y_{n-1}) = \sum_{i=0}^{n/2-1} F_{2cont}(y_{2i}, y_{2i+1}) \tag{7.15}$$

Non-overlapping pairs of neighboring variables contribute to the overall fitness by

$$F_{2cont}(u,v) = F_{2c}\left(\sqrt{\frac{u^2+v^2}{2}}\right) \tag{7.16}$$

where F_{2c} is a one-dimensional deceptive function defined on [0,1] as

$$F_{2c}(w) = \begin{cases} 0.8 - w & \text{if } w \le 0.8 \\ 5(w - 0.8) & \text{otherwise} \end{cases} \tag{7.17}$$

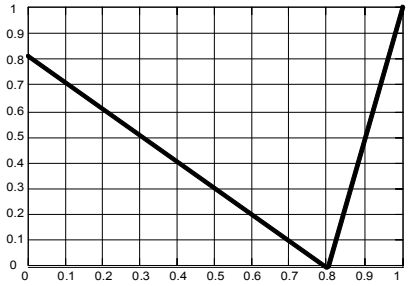


Fig. 7.9: Function F_{2c}.

If both variables are treated independently, the search will progress toward the local optimum. Moreover, the global optimum is very isolated and its attraction is very small. This represents quite difficult benchmark.

7.6 Functions with mixed parameters

For experiments with MBOA algorithm (see 8.5) I prepared my own mixed continuous-discrete benchmark. Each chromozome is composed of binary genes b_i and continuous genes c_i, where each pair (b_i,c_i) is an argument of primitive deceptive-like function F_m:

$$F_{mixed}(b_0, c_0, b_1, c_1, ..., b_{n/2-1}, c_{n/2-1}) = \sum_{i=0}^{n/2-1} F_m(b_i, c_i) \tag{7.18}$$

$$F_m(b_i, c_i) = \begin{cases} 0{,}8 - 0.5 * c_i & b_i = 0 \\ 0.8 - c_i & b_i = 1 \wedge c_i \le 0.8 \\ 5 * (c_i - 0.8) & b_i = 1 \wedge c_i > 0.8 \end{cases} \tag{7.19}$$

8 BOA for continuous and discrete problems

In this chapter I identify some disadvantages of present probabilistic models used in EDAs to solve optimization problems with continuous parameters and propose more general and efficient model based on the decision trees. In my Mixed Bayesian Optimization Algorithm (MBOA) I implemented the decision trees with mixed decision nodes, inspired by the CART model, so MBOA is suitable for both continuous and/or discrete optimization problems.

8.1 Present approaches

8.1.1 BOA with marginal histogram models

The most straightforward way how to solve continuous benchmarks by BOA is to transform each continuous parameter into binary parameter and use the Bayesian network model for such a binary encoding.

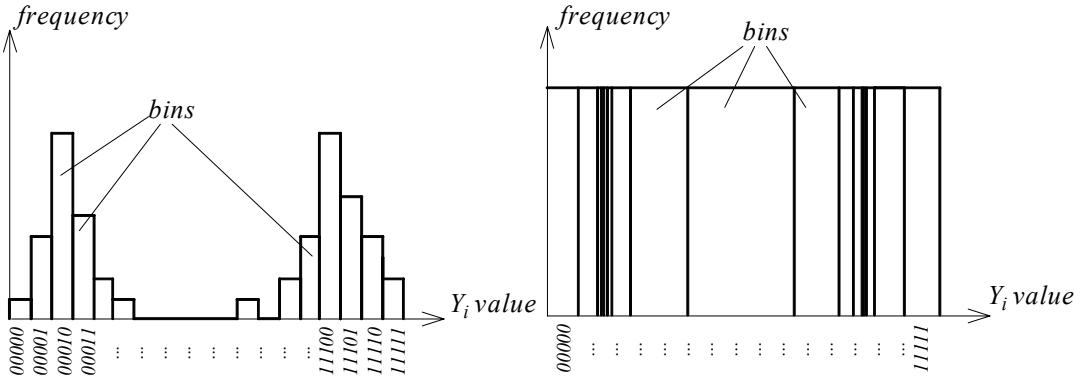


Fig. 8.1*: Converting continuous values of variable Y_i into bins of discrete variable X_i using fixed width histogram (left) and fixed height histogram (right).*

See the papers [118] and [98] for further details on this approach. Unfortunately, this approach is not scalable. With the increasing precision of encoding the BN parameter k becomes unmanageably large and the complexity of BN construction grows exponentially.

8.1.2 IDEA and EGNA

The other way of solving continuous optimization problem by EDA is to let the parameters Y_i be continuous (like in evolutionary strategies) and find a proper model for this continuous domain. For continuous domains there is an IDEA (see [19] or 4.5.4) approach used to discover and encode the parameter dependencies – estimating Gaussian networks model (which has beeen used also in EGNA algorithm, see 4.5.4), Gaussian kernels and mixtures of Gaussians. See chapter 3.3.7 and 3.3.8 for description of methods for learning these models. In chapter 3.2.4 I show that the simple Gaussian network is not accurate and some clustering needs to be preprocessed. The clustering models (like mixture of Gaussians) are accurate, but not compatible with discrete domains. This is the reason for proposing new model.

8.2 Mixed decision trees model for local parameters

Binary decision trees (BDT) have been successfully used in EDAs [96] for discrete optimization problems – in combination with the Bayesian network. In Mixed Bayesian Optimization Algorithm (MBOA) design I have extra focused on non-binary problems too.

8.2.1 Topology of CART model

From the area of data mining I adopted the idea of CART model [20] (Classification and Regression Tree), which is usable also for continuous and categorical splits:

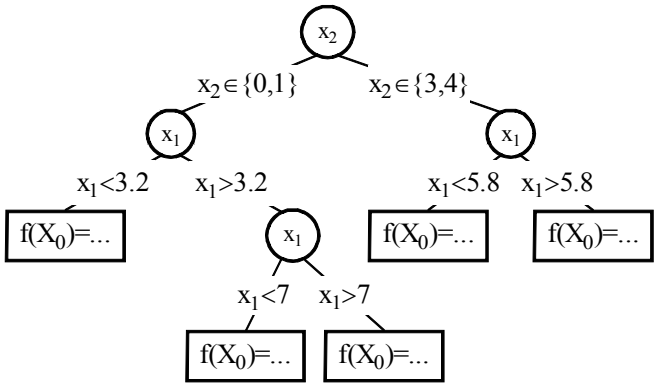


Fig. 8.2*: An example of CART model which comprises of one categorical split on variable X_2 and three continuous splits on variable X_1. The leaf nodes conditionally determine the probability density functions for target variable X_0.*

From the CART model I adopted the idea of mixed splitting, but for time simplicity I omitted the additional CART methods, e.g. the final pruning of constructed tree.

8.2.2 Recursive splitting

For each "target" random variable X_i MBOA builds one decision tree. The parent variables Π_i contained in split nodes of i-th decision tree are used to cut the space of all chromozomes into parts, where the variable X_i is more linear or predictable.

The building of decision trees starts from empty trees and it recursively adds the splitting nodes until no splitting is favourable:

```
Function RecursiveSplitting(Population Pop, TargetVariable Xi,
           ListOfCandidateSplitVariables Pa) : DecisionTreeNode
Begin
  fTemp := EstimateElementaryPDF(Pop,Xi,Pa);
  If "fTemp is sufficiently detailed" then return new LeafNode(fTemp);
  For Each Variable Xj in Pa do
     Ej := Find_optimal_threshold_of_Xj_with_respect_to_Xi;
     ModelGain := Evaluate_the_split_quality("Xj≤Ej", Xi);
     Save the Xj and Ej with the highest ModelGain;
  End for
  Pop1 := SelectIndividuals (Pop,"Xj ≤ Ej");
  Pop2 := SelectIndividuals (Pop,"Xj > Ej")
  return new SplitNode(new SplitCondition("Xj ≤ Ej"),
                       RecursiveSplitting(Pop1,Xi, Pa\{Xj}),
                       RecursiveSplitting(Pop2,Xi, Pa\{Xj}));
End;
```

The step of split condition finding and evaluation is essential. The algorithm uses a greedy search, that is, it picks the best candidate and never looks back to reconsider earlier choices. The choice is based on the equation we derived from the Bayesian-Dirichlet metrics (for details on this metrics see 8.3.1). You see that the continuous-valued decision attributes are incorporated into the learned tree by dynamically defining new binary attributes that partition the continuous attribute value into two intervals ($X_j \leq E_j$ and $X_j > E_j$). The threshold E_j is selected among the candidate thresholds to maximize the metrics.

In the RecursiveSplitting procedure the list of candidate split variables is given in advance, which allows for the future parallelization, see 9.2.2 and 9.5.1.

8.2.3 Elementary probabilistic models

The leaf nodes define the elementary models for obtaining the "target" variable X_i. For continuous variables one-dimensional normal PDF can be estimated and used as the leaf (the estimation of PDF parameters was stated in 3.3.2). But I obtained better results with Gaussian kernels (3.3.8) or with the local linear regression model. The parameters of local regression model can be obtained in the same way as in the Gaussian network case, but I proposed also approximation of coefficients by Widrow-Hoff rule [8] known from iterative training of linear perceptron (using the original notation for those skilled in the NN art):

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \alpha(d - y)\boldsymbol{x}\,, \tag{8.1}$$

where $\boldsymbol{w}$ is the estimated vector of regression coefficients, α is the learning rate, d is the desired output of regression model (the mean value of variable X_i in our case), y is the actual output of regression model (the predicted X_i) and $\boldsymbol{x}$ is the vector of differences of parent variables Π_i from their mean value.

8.3 Building mixed decision trees

8.3.1 Derivation of DT metrics from BDe

To derive an incremental decision tree metrics (DT metrics) for evaluation of one potential split node, let us start from BDe metrics for decision graphs from 3.3.6:

$$p(D, B_S^h \mid \xi) = p(B_S^h \mid \xi).\prod_{i=0}^{n-1}\prod_{l\in L_i}\frac{\Gamma(m'(i,l))}{\Gamma(m(i,l)+m'(i,l))}\prod_{x_i}\frac{\Gamma(m(x_i,i,l)+m'(x_i,i,l))}{\Gamma(m'(x_i,i,l))}.$$

Now let us separate this metrics into local terms, assuming that the problem of finding the proper BN structure B_S^h can be decomposed into independent subproblems of finding local decision graphs G_i :

$$p(D, B_S^h \mid \xi) = p(B_S^h \mid \xi)\prod_{i=0}^{n-1} p(X_i \mid G_i, \xi) \tag{8.2}$$

and let us focus only on the decision graph G_i for variable X_i:

$$p(X_i \mid G_i, \xi) = \prod_{l\in L_i}\frac{\Gamma(m'(i,l))}{\Gamma(m(i,l)+m'(i,l))}\prod_{x_i}\frac{\Gamma(m(x_i,i,l)+m'(x_i,i,l))}{\Gamma(m'(x_i,i,l))} \tag{8.3}$$

Now consider the algorithm for building the decision tree for the target variable X_i. We need to derive the equation for addition of X_j split:

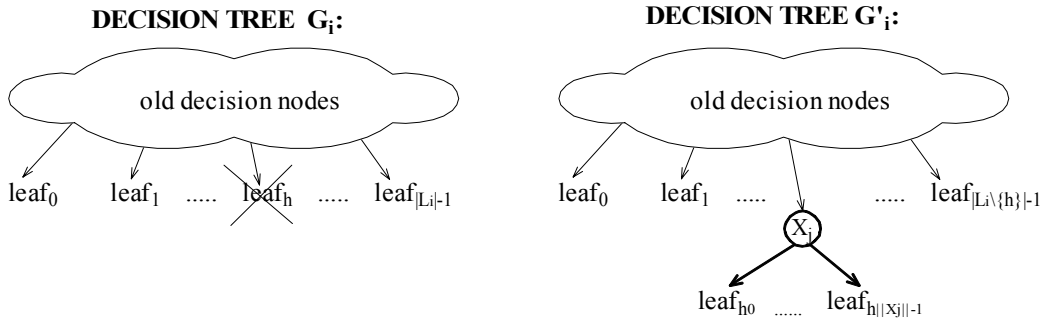


Fig. 8.3*: Adding X_j split.*

This operation replaces the leaf node number h by decision node X_j and introduces new leafs $h_0,\ldots,h_{\|Xj\|-1}$. This results in a new tree G'_i. The relative increase of model quality for X_i is:

$$\frac{p(X_i \mid G'_i,\xi)}{p(X_i \mid G_i,\xi)} = \frac{\prod_{l\in L_i\cup\{h_0,\ldots,h_{\|X_j\|-1}\}\setminus\{h\}} \frac{\Gamma(m'(i,l))}{\Gamma(m(i,l)+m'(i,l))} \prod_{x_i} \frac{\Gamma(m(x_i,i,l)+m'(x_i,i,l))}{\Gamma(m'(x_i,i,l))}}{\prod_{l\in L_i} \frac{\Gamma(m'(i,l))}{\Gamma(m(i,l)+m'(i,l))} \prod_{x_i} \frac{\Gamma(m(x_i,i,l)+m'(x_i,i,l))}{\Gamma(m'(x_i,i,l))}} = \tag{8.4}$$

$$= \frac{\prod_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} \frac{\Gamma(m'(i,l))}{\Gamma(m(i,l)+m'(i,l))} \prod_{x_i} \frac{\Gamma(m(x_i,i,l)+m'(x_i,i,l))}{\Gamma(m'(x_i,i,l))}}{\frac{\Gamma(m'(i,h))}{\Gamma(m(i,h)+m'(i,h))} \prod_{x_i} \frac{\Gamma(m(x_i,i,h)+m'(x_i,i,h))}{\Gamma(m'(x_i,i,h))}}$$

Now the prior values need to be specified. From the K2 approach it follows the uniform assignment $m'(x_i,i,l)=1$ and $m'(i,l)=\sum_{x_i} m'(x_i,i,l)=\| X_i \|$. Analogically, we can set $m'(x_i,i,h)=1$ and $m'(i,h)=\| X_i \|$. But to derive more compact form of equation, I suggest to set the priors for leaf h as the sum of priors for leafs $h_0,\ldots,h_{\|Xj\|-1}$:

$$m'(x_i,i,h)=\sum_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} m'(x_i,i,l)=\| X_j \|. \tag{8.5}$$

$$m'(i,h)=\sum_{l\in\{h_0,\ldots,h_{\|X_j\|}\}} m'(i,l)=\| X_j \|.\| X_i \| \tag{8.6}$$

Thus,

$$\frac{p(X_i \mid G'_i,\xi)}{p(X_i \mid G_i,\xi)} = \frac{\prod_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} \frac{\Gamma(\| X_i \|)}{\Gamma(m(i,l)+\| X_i \|)} \prod_{x_i} \frac{\Gamma(m(x_i,i,l)+1)}{\Gamma(1)}}{\frac{\Gamma(\| X_j \|.\| X_i \|)}{\Gamma(m(i,h)+\| X_j \|.\| X_i \|)} \prod_{x_i} \frac{\Gamma(m(x_i,i,h)+\| X_j \|)}{\Gamma(\| X_j \|)}} =$$

$$= \frac{\prod_{x_i}\Gamma(\| X_j \|)}{\Gamma(\| X_j \|.\| X_i \|)} \cdot \frac{\Gamma(m(i,h)+\| X_j \|.\| X_i \|). \prod_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} \prod_{x_i} \Gamma(m(x_i,i,l)+1)}{\left(\prod_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} \Gamma(m(i,l)+\| X_i \|)\right).\left(\prod_{x_i} \Gamma(m(x_i,i,h)+\| X_j \|)\right)} \tag{8.7}$$

The first fraction is constant if $\|X_i\|$ and $\|X_j\|$ are constant. We are interested only in the relative comparison. Let us call the second fraction $gain(X_i,leaf_h)$. It can be further simplified, because the new leafs $h_0,\ldots,h_{\|Xj\|-1}$ were created by splitting the original leaf h. It holds

$$m(x_i,i,h) = \sum_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} m(x_i,i,l)\,. \tag{8.8}$$

and also

$$m(i,h) = \sum_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} m(i,l) = \sum_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} \sum_{x_i} m(x_i,i,l) \tag{8.9}$$

Thus, the (8.7) can be also written as

$$gain(X_i,leaf_h) = \frac{\Gamma\left(\sum_{l\in\{h_0,\ldots,h_{\|X_j\|}\}} \sum_{x_i} (m(x_i,i,l)+1)\right). \prod_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} \prod_{x_i} \Gamma(m(x_i,i,l)+1)}{\left(\prod_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} \Gamma\left(\sum_{x_i} (m(x_i,i,l)+1)\right)\right)\left(\prod_{x_i} \Gamma\left(\sum_{l\in\{h_0,\ldots,h_{\|X_j\|-1}\}} (m(x_i,i,l)+1)\right)\right)} \tag{8.10}$$

This metrics is especially suitable for split-and-conquer approach. You see that all the counts $m(x_i,i,l)$ we need to know for splitting of the leaf number h can be determined only among the individuals which match the conditions leading from the root to this leaf. Let us denote this subpopulation $D'(t)$. From this point of view the addition of a new split can be also seen as the recursive addition of a first parent variable X_j to the set Π_i in the classical BN. The metrics based on such idea can be written as:

$$gain(X_i,X_j) = \frac{\Gamma\left(\sum_{x_j}\sum_{x_i}(m(x_i,x_j)+1)\right).\prod_{x_j}\prod_{x_i}\Gamma\left(m(x_i,x_j)+1\right)}{\left(\prod_{x_j}\Gamma\left(\sum_{x_i}(m(x_i,x_j)+1)\right)\right).\left(\prod_{x_i}\Gamma\left(\sum_{x_j}(m(x_i,x_j)+1)\right)\right)} \tag{8.11}$$

Note that the population is splitted recursively, so the terms $m(x_i, x_j)$ now denote the counts of some patterns only within $D'(t)$ and N denotes size of $D'(t)$.

8.3.2 DT metrics implementation

Since the values of Γ function grow to large numbers for the real computation I use logarithm of this metric. All the arguments of Γ function are integer numbers smaller than *2*N*, so I can precompute the array of cumulative logarithms Γ'[1]…Γ'[*2*N*] such that Γ'[i]=$\log_2\Gamma$(i)= $\log_2$((i-1)!) by recursive formula

$$\begin{aligned} &\Gamma'[1] = 0 \\ &\forall i > 1 : \Gamma'[i] = \Gamma'[i-1] + \log_2(i-1) \end{aligned} \tag{8.12}$$

Now the metric can be written as

$$\begin{aligned} \log\, gain(X_i, X_j) = &\left(\Gamma'\left[\sum_{x_j}\sum_{x_i}(m(x_i,x_j)+1)\right]\right) + \left(\sum_{x_j}\sum_{x_i}\Gamma'\left[m(x_i,x_j)+1\right]\right) - \\ &-\left(\sum_{x_j}\Gamma'\left[\sum_{x_i}(m(x_i,x_j)+1)\right]\right) - \left(\sum_{x_i}\Gamma'\left[\sum_{x_j}(m(x_i,x_j)+1)\right]\right) \end{aligned} \tag{8.13}$$

From empirical analysis of (8.13) it follows that this metrics has problems to distinguish between splits when the numbers $m(x_i, x_j)$ are small (e.g. 0,1,2). This is caused by the properties of Γ function, because Γ'[1]=Γ'[2]=0. To achieve better distinguishability for small numbers I suggest the following modification

$$\begin{aligned} &\Gamma'[0] = 0 \\ &\Gamma'[1] = 0 \\ &\forall i > 1 : \Gamma'[i] = \Gamma'[i-1] + \log_2(i) \end{aligned} \tag{8.14}$$

This formula is based on factorials, such that Γ'[i]=$\log_2$(i!).

8.3.3 Example of DT metrics usage

Consider the population of parents $D(t)$={0000, 0000, 0101, 0101, 1001, 1101, 1101, 1110}. For example we want to build the decision tree for variable X_3 and we are interested in quality of X_1 split. The aggregation table for this case is:

	$x_3 = 0$	$x_3 = 1$	$m(x_1) = \sum_{x_3} m(x_3, x_1)$
leaf 0: $x_1 = 0$	$m(x_3=0, x_1=0) = 2$	$m(x_3=1, x_1=0) = 1$	$m(x_1=0)$=3
leaf 1: $x_1 = 1$	$m(x_3=0, x_1=1) = 1$	$m(x_3=1, x_1=1) = 4$	$m(x_1=1)$=5
$m(x_3) = \sum_{x_1} m(x_3, x_1)$	$m(x_3=0) = 3$	$m(x_3=1) = 5$	N=8

According to equations (8.14) and (8.13) the score for this aggregation table is

$$\begin{aligned} \log\, gain(X_3, X_1) = &\Gamma'[8+4] + \Gamma'[2+1] + \Gamma'[1+1] + \Gamma'[1+1] + \Gamma'[4+1] - \\ &-\Gamma'[3+2] - \Gamma'[5+2] - \Gamma'[3+2] - \Gamma'[5+2] \simeq 1.92 \end{aligned}$$

The whole recursive building of decision tree for x_3 can be shown as:

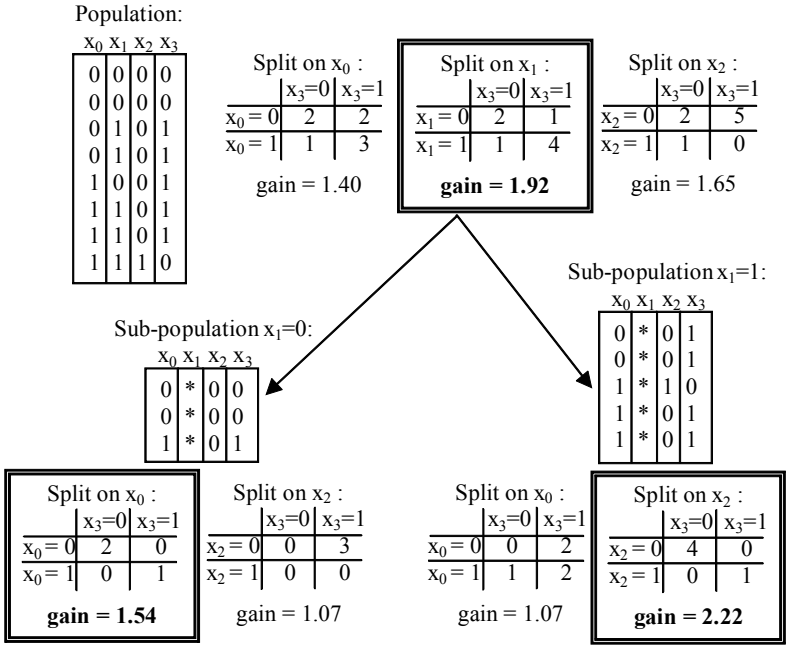


Fig. 8.4*: Example of recursive building of decision tree for x_3.*

And the obtained decision tree is

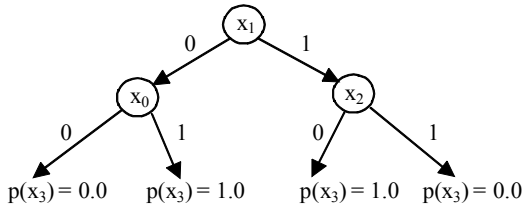


Fig. 8.5*: Example of final decision tree for x_3.*

8.3.4 Model complexity penalization

In principle the described decision tree algorithm can grow each branch of the tree just deeply enough to perfectly classify the training examples. In the previous example in Fig. 8.5 we obtained fully determined decision tree. Sampling of probabilistic model with fully determined trees will produce the duplicate of original population – no uncertainty is present. We must avoid overfitting, such that more simpler trees are produced.

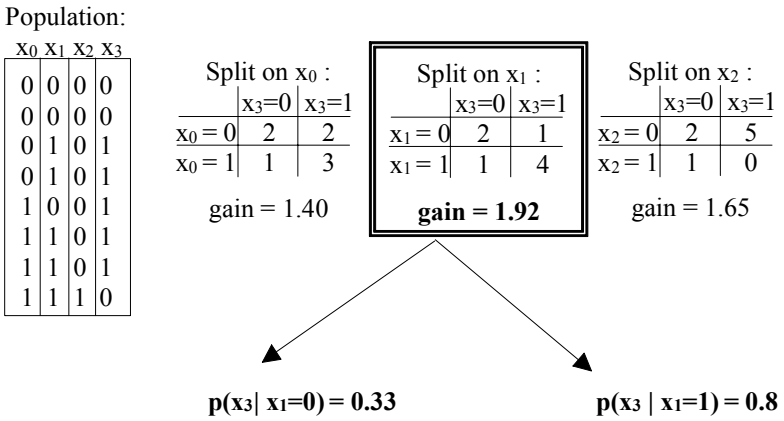


Fig. 8.6*: Decision tree with reduced complexity.*

In 3.3.6 the prior probability for binary decision graph was stated as

$$p(B_s^h \mid \xi) = c.2^{(-0.5\log_2 N)\sum_i |L_i|},$$

where c is a normalization required for the prior probabilities of all networks to sum to 1 and $|L_i|$ is the number of leaves in *i*-th graph. The value of a normalization constant does not affect the result, since we are only interested in relative comparisons of networks and not the absolute value of their likelihood. This assignment is sufficient to bias the model construction to networks with less parameters and avoid superfluously complex network structures.

Now consider non-binary trees. The relative increase of penalty score after removing one leaf number *h* and adding new leaves $h_0,\ldots,h_{||Xj||-1}$ is

$$penalty = \frac{c.2^{(-0.5\log_2 N)((\sum_i |L_i|)-1+2)}}{c.2^{(-0.5\log_2 N)\sum_i |L_i|}} = \frac{2^{(-0.5\log_2 N)((\sum_i |L_i|)-1+2)}}{2^{(-0.5\log_2 N)\sum_i |L_i|}} = 2^{-0.5\log_2 N} \quad (8.15)$$

For practical purposes I use logarithm of the metrics, thus the leaf penalty has also to be in logarithmic form.

$$\log penalty = -0.5\log_2 N \quad (8.16)$$

And for non-binary trees the penalty is

$$\log penalty = (-0.5\log_2 N)(\| X_j \| -1)(\| X_i \| -1) \quad (8.17)$$

Note that the population size *N* in the penalty equation is not sized down recursively, it is the size of original parent population.

8.3.5 Adaptation for continuous parameters

My initial DT metrics equation is restricted to attributes that take on a discrete set of values. First, the target attribute whose value is predicted by the learned tree must be discrete valued. Second, the attributes tested in the decision nodes of the tree must also be discrete valued.

This second restriction can easily be removed so that continuous-valued decision attributes can be incorporated into the learned tree. This can be accomplished by dynamically defining new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals. In particular, for an attribute Y_j that is continuous-valued, the algorithm can dynamically create a new Boolean attribute X_j that is true if $Y_j < E_j$ and false otherwise. The only question is how to select the best value for the threshold E_j. Clearly, we would like to pick a threshold, E_j, that produces the highest DT metrics score. By sorting the individuals according to the value of continuous attribute Y_j, then identifying adjacent examples that differ in their target variable, we can generate a set of candidate thresholds midway between the corresponding values of Y_j. These candidate thresholds can then be evaluated by computing the metrics associated with each. Such a technique can be used for each of the continuous candidate attributes Y_j, and the best can be selected. This dynamically created binary/Boolean attribute can then compete with the other discrete-valued candidate attributes available for growing the decision tree.

This dynamical approach produces far more smoother discretization than the histogram-based approach in [98] and [57].

Now let us focus on the first restriction – the continuous target attribute Y_i. We can also use dynamical discretization for it, assuming that the strongest dependencies are the most relevant, i.e. the threshold E_i which maximizes the metrics is the best one. But this approach becomes computationally expensive when we consider both continuous split attribute and continuous target variable – we should evaluate each combination of possible thresholds E_i and E_j. In this case I use a heuristics simplification instead – for given E_j I choose E_i using the following two steps:

1. Determine M_i as the median of Y_i values among the individuals satisfying the condition "$X_j<E_j$".
2. Determine E_i such that the number of all individuals satisfying the condition "$M_i<X_i<E_i$" resp. "$M_i>X_i>E_i$" is equal to $m/2$, where m denotes the number individuals satisfying the condition "$X_j<E_j$".

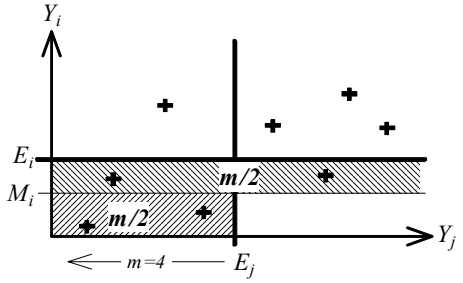


Fig. 8.7*: Determination of E_i threshold for given E_j threshold.*

Regardless of population size, this E_i can be determined for given E_j in constant time (using the sophisticated data structures for fast determination of medians). Practical experiments confirm that this assignment is acceptable in most situations. Note that the trivial choosing E_i as the median of Y_i values is similar to fixed-height histogram discretization and choosing E_i as the average of Y_i values is similar to fixed-width histogram discretization. Also note that the determined value E_i is used only for metrics evaluation, but can be omitted after that.

8.3.6 Adaptation for categorical and integer parameters

The metrics derived in 8.3.1 is inherently compatible with integer domains, creating $||X_j||$ leaves for integer attribute X_j. However, for better results it might useful to deal with integer parameters in the same way as with continuous parameters, such that the threshold E_j is found and only two leaves („$X_j \leq E_j$“ and „$X_j > E_j$“) are created in each step. Since the number of leaves is reduced, the elementary model estimation for each leaf takes, on average, a larger number of samples into account and thus is more robust.

Categorical parameters are special type of integer parameters. For example the *age* of the person is a common integer parameter, but the first digit of a zip-code of its address is the categorical parameter. It does not make sense to split the numerical identification of categories into two intervals, because the categories are not mutually comparable. Instead of using some threshold, it makes sense to use arbitrary partitioning of categories into subsets.

My MBOA algorithm uses a hill-climbing algorithm to split the set of possible X_j categories into left and right subset. It starts with the empty right subset and it tries to move categories from left subset to right subset until the maximum of DT metrics is reached. This results in a new binary attribute X'_j which is true if X_j belongs to the left subset and false otherwise. This new attribute then competes with the other candidates for split. Also the target variable X_i can be of categorial type, but in this case the number of target categories *of* X_i does not affect the complexity of the tree, so X_i remains unchanged.

Tab. 8.1. An example of contingency table with partitioned domain of categorical attribute X_j.

	$\mathbf{X_i = 0}$	$\mathbf{X_i = 1}$	$\mathbf{X_i = 2}$	$\mathbf{X_i = 3}$
$\mathbf{X_j \in \{0,2\}}$	$m(X_i=0, X'_j=0)$	$m(X_i=1, X'_j=0)$	$m(X_i=2, X'_j=0)$	$m(X_i=3, X'_j=0)$
$\mathbf{X_j \in \{1,3\}}$	$m(X_i=0, X'_j=1)$	$m(X_i=1, X'_j=1)$	$m(X_i=2, X'_j=1)$	$m(X_i=3, X'_j=1)$

8.4 Optimizing multimodal functions

8.4.1 Using fitness for preserving divergence

I proposed an additional technique which enables MBOA to optimize multimodal functions. The fitness value is allowed to act as a split in decision trees, like the ordinary continuous variable. This “pseudo-fitness” value is always treated as an independent parameter, it’s value is generated first and the genes are generated thereafter. By shifting the distribution of “pseudo-fitness” towards the lower values we can slightly bias the exploitation of search space towards the regions where the minor solutions occur. Due to this advancement MBOA achieves better divergence. Of course, after generation of the whole chromozome the “pseudo-fitness” value is replaced by the true fitness value.

8.4.2 Restricted tournament replacement

The next advantage of MBOA is the utilization of RTR (Restricted Tournament Replacement) proposed by Martin Pelikan in [97]. In the replacement stage each new individuals competes with the Euclidean-nearest individual among 20 randomly selected individuals. This also ensures better niching.

8.5 Experiments

During the experiments I compared the efficiency of MBOA approach with the other EDA and IDEA algorithms. First of all I proved the backward-compatibility with the binary domain. I compared the performance of MBOA with the Pelikan's original simple BOA algorithm on several deceptive functions:

Tab. 8.2. Results for 3-deceptive and trap-5 function (see 7.4)- the average number of fitness evaluations required to reach the global optima in 10 runs, with min. population size L.

Algorithm\Benchmark	*3-deceptive function, n=90*	*trap-5 function, n=90*
MBOA	*43530 evaluations, L=3800*	***53280** evaluations, L=5500*
Simple BOA	***43510** evaluations, L=3800*	*54310 evaluations, L=5500*

Secondly, I have used some continuous benchmarks to compare MBOA with the best IDEA algorithms in [19]. IDEA1 uses non-clustered normal mixture of 10 PDFs, IDEA2 and IDEA5 use Euclidean 2-means clustering with normal mixtures of 10 PDFs, IDEA3 uses Mahalanobis leader 3½ clustering with normal mixtures of 10 PDFs, IDEA4 uses non-clustered normal mixture of 5 PDFs and IDEA6 uses Mahalanobis leader 5 clustering with normal mixtures of 5 PDFs.

Tab. 8.3. Griewank function (see 7.5), problem size n=5, domain= $[-5,5]^5$.

Algorithm	*Fitness evaluations to get the global optima (avg. for 10 runs, 7-digit precision)*
MBOA with univariate Gaussian leaves	***19790**, L=120*
IDEA1	*51832, L=1100*
IDEA2	*72552, L=1750*
IDEA3	*89718, L=2250*

Tab. 8.4. Michalewicz function (see 7.5), n=5, domain= $[0,\pi]^5$.

Algorithm	*Fitness evaluations to get the global optima (avg. for 10 runs, 7-digit precision)*
MBOA with univariate Gaussian leafs	***7690**, L=120*
IDEA4	*28903, L=1300*
IDEA5	*16596, L=750*
IDEA6	*22118, L=1000*

Thirdly, I tested the scalability of MBOA on continuous two-deceptive function (see 7.5) and I examined how the number of fitness evaluations grows with the problem size.

Tab. 8.5. The scalability for continuous two-deceptive function.

Algorithm	*Continuous two-deceptive function, 5-digit precision*
MBOA with Gaussian-kernels leafs	***O(n$^{1.79}$)*** *(n=10-50, L=300-3500)*
Simple BOA with fixed-width histogram discretization, see [98]	*O(n$^{2.1}$) (4 bins, n=10-40, L=1000-7400)*
Simple BOA with fixed-height histogram discretization, see [98]	*O(n$^{2.1}$) (16 bins, n=10-30, L=3500-17000)*

Fourthly, I prepared own mixed continuous-discrete benchmark (defined in 7.6), which is very hardly sensitive to correctness of DT building algorithm. Dependencies between mixed parameters have to be necessarily discovered to find the global optimum. Because of the benchmark deceptiveness the algorithm fails if the dependency metric prefers certain domain over another domain.

Tab. 8.6. The average number of evaluation for mixed benchmark.

Problem size	*n=10*	*n=20*	*n=30*	*n=40*	*n=50*
MBOA with Gaussian-kernels leafs	*23100*	*67800*	*126000*	*214800*	*291000*
	L=600	*L=1200*	*L=1800*	*L=2400*	*L=3000*

The empirical results show that in the field of discrete optimization problems the MBOA is backward-compatible with the original BOA algorithm and provides similar results. The results for continuous Griewank and Michalewicz function confirm the assumption that MBOA overcomes the IDEA approach.

On the two-deceptive continuous benchmark I showed that the scalability of MBOA is better than the scalability of discretization approach based on histogram models [98]. The MBOA was also able to solve the difficult mixed continuous-discrete benchmark.The results indicate that the main potential of my algorithm lies in solving high-dimensional problems with stronger parameter nonlinearities.

8.6 Application of BDT for Boolean function modelling

This chapter briefly describes the application of BDe-driven BDT-building algorithm for modelling of Boolean functions. For further details see the papers [ii] and [iii].

Binary Decision Diagrams (BDDs) are commonly used for representation of Boolean functions. This representation is very suitable for practical purposes, for example it can be directly converted into circuit design. Each decision node represents the Shannon's expansion of the Boolean function

$$F = (x_i F_1 + \overline{x_i} F_0) \tag{8.18}$$

where i is the index of the decision node, F_0 and F_1 are the functions of the nodes pointed to by 0- and 1- edges, respectively. Terminal nodes represent the logic values (1/0).

In the area of circuit design there are many heuristics used to construct optimal decision graph for given function. If the Boolean function is given in the form of complete truth table, we can adopt our decision-tree building algorithm, which works with the function table in the same manner as with the population of individuals. Only one tree is built – the function value acts as the target variable and the input variables are candidates for splits:

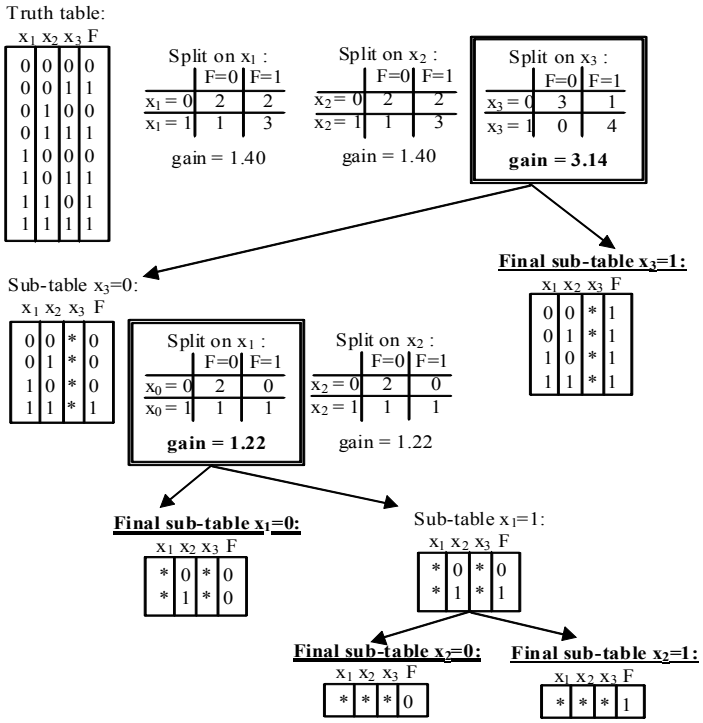


Fig. 8.8*: Building decision tree for* $F= x_1x_2+x_3$

The motivation for using BDe metrics is straightforward – we believe that during the topdown tree induction the inputs which affect the function value most should be used first, which hopefully leads to less complex trees. Note that we use statistical methods to measure the relationship between function value and input value, but the resulting tree for Boolean function must be fully determined (without avoiding overfitting).

My decision tree building algorithm is able to induce decision trees only. BDDs or oriented BDDs (OBDDs) can be constructed from decision trees using two basic reduction rules: 1) sharing all equivalent sub-graphs and 2) reduction of nodes with identical ancestor nodes.

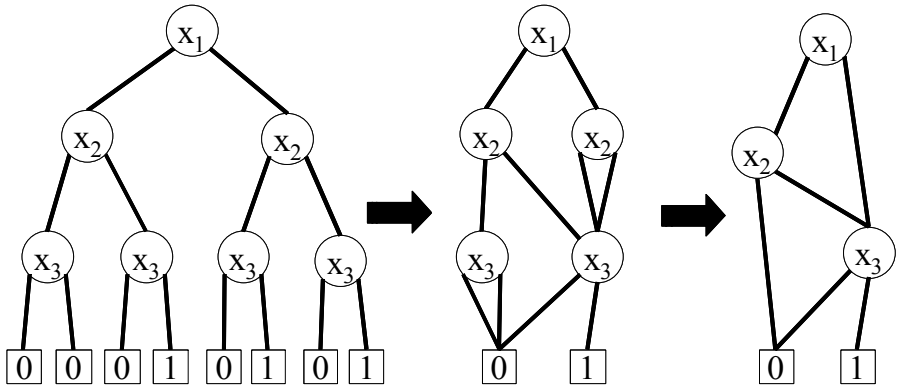


Fig. 8.9*: Conversion from BDT to BDD*

The time complexity of our BDT construction is $O(n^2 2^n)$, where n is the number of function inputs. This is significant improvement when compared with the classical greedy BDD construction - $O(n!2^n)$ or with the dynamic programming approach - $O(n^2 3^n)$. We must include the complexity of conversion from BDD to BDT, but it does not exceed the BDT construction complexity.

The typical benchmark for BDD construction algorithms is the multiplexor function $F=x_0x_1+x_2x_3+\ldots+x_{n-2}x_{n-1}$. See the following figure where an example with n=4 is shown.

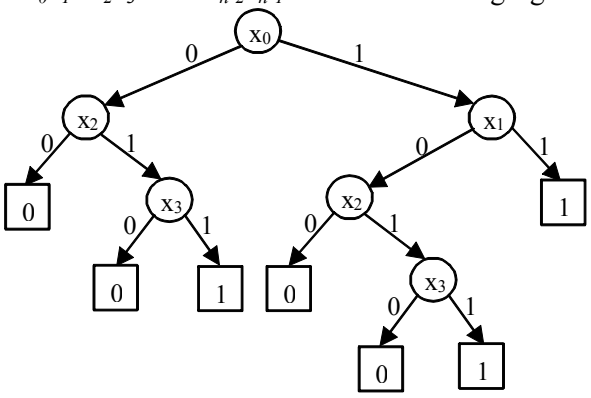


Fig. 8.10*: BDT-model of Boolean function* $F=x_0x_1+x_2x_3$

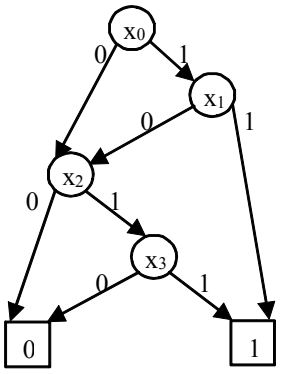


Fig. 8.11*: BDD after removing duplicate subgraphs*

This multiplexor benchmark is very sensitive to proper ordering of variables during topdown recursion. Improper ordering would lead to exponentially large decision graph:

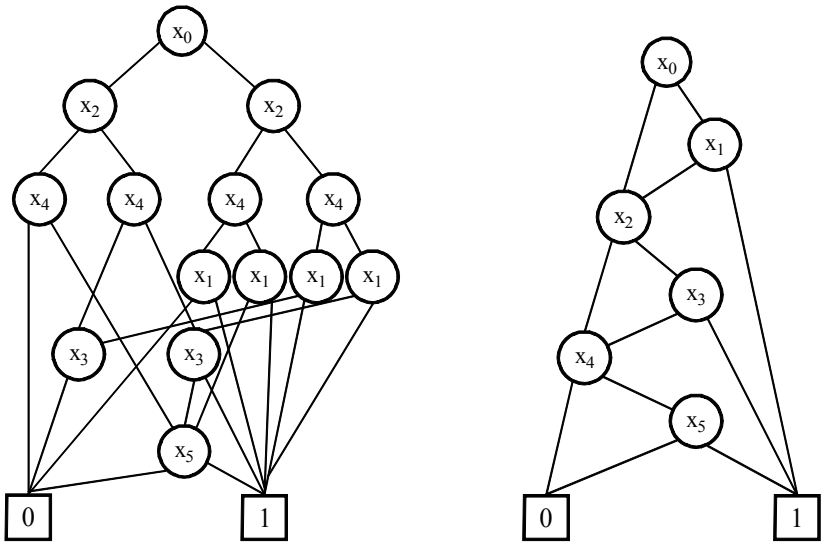


Fig. 8.12*: The effect of variables ordering*

The results in Tab. 8.7 confirm that our algorithm was able to correctly construct the decision tree and reduce it to decision graph with size complexity O(*n*).

Tab. 8.7. The results for multiplexor benchmark

Number of variables *n*	4	6	8	10	12	14	20
Number of BDT decision nodes	6	14	30	62	126	254	2046
Number of reduced BDD decision nodes	4	6	8	10	12	14	20

We are also able to build models for multi-valued functions when using the metrics for integer domains:

x0	**x1**	**x2**	**F**
0	0	0	3
0	0	1	5
0	1	0	3
0	1	1	4
1	0	0	4
1	0	1	4
1	1	0	4
1	1	1	5

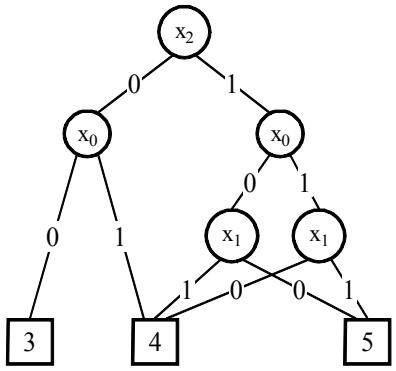


Fig. 8.13*: An example of tree for multi-valued function a) truth table, b) BDD*

Our results confirm that BDe metrics can be utilized for many engineering tasks like Boolean function modelling. The advantage of suggested approach lies in low BDT construction complexity. However, further improvements are needed to deal with symbolic representation of functions and to obtain ordered BDD.

9 Design of parallel EDA algorithms

9.1 Classical concept for parallel GAs

The usual concept of classical parallel GA is inspired by nature. The population is divided into a few subpopulations or demes, and each of these relatively large demes evolves separately on different processors. Exchange between subpopulations is possible via a migration operator. The term *islands model* is easily understandable; the GA behave as if the world was constituted of islands where populations evolve isolated from each other. On each island the population is free to converge toward different optima. The migration operator is supposed to mix good features that emerge locally in the different demes.

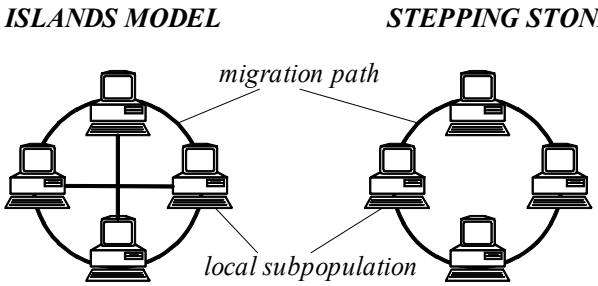


Fig. 9.1: Classical models of parallel GAs

Many topologies can be defined to connect the demes, but the most common models are the island model and the stepping stones model. In the basic island model, migration can occur between any subpopulations, whereas in the stepping stone demes are disposed on a ring and migration is restricted to neighbouring demes. From literature it is known that the topology of the space is not so important as long as it has high connectivity and small diameter to ensure adequate mixing as time proceeds.

9.2 Parallelization of EDA

The most significant difference between classical GA and the EDA algorithm lies in the character of creation of new population. Classical GA uses pairwise recombination whereas EDA uses the whole parent population for estimation of probabilistic model. In case of classical GA the crossover can be performed even locally, but in case of EDA the classical models of parallelism are useless, because the accuracy of estimated model decreases rapidly with decreasing size of parent population. For example 10 probabilistic models estimated from local subpopulations of size $N/10$ would produce worse quality offspring than the sequential model estimated from population of size N. The only way how to design a parallel EDA algorithm is to propose new paradigms for parallel construction and sampling of single probabilistic model in the pseudo-sequential manner.

9.2.1 Time profile of BOA

I investigated the execution profile of BOA. This empirical analysis identifies the main performance bottleneck of BOA – the construction of probabilistic model.

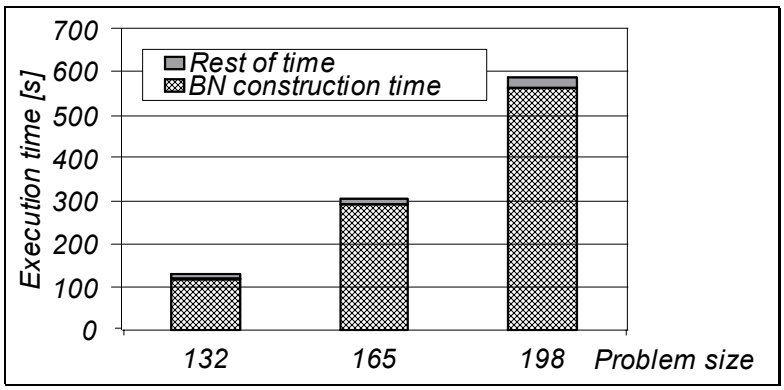


Fig. 9.2: The time profile of average run of BOA algorithm for 3-deceptive function

Nearly all the execution time of sequential BOA is spent to find the dependence graph of Bayesian network. The remaining time includes estimation of local CPDs parameters, BN sampling, fitness computation and selection/replacement. In comparison to the construction of dependence graph many of those remaining tasks are easy to be done in parallel, in obvious manner or in similar manner as in the classical GAs, but they take only less than 5% of the overall time. Thus, some method for parallel BN construction needs to be proposed in following chapters.

The sequential Bayesian network construction starts with an empty network (all variables independent, like in UMDA algorithm). It performs a greedy addition of edges into the network, which works in the following way:

```
Start with an empty network B;
for each edge do                                          ...  O(n^2)
begin
   Compute the score of its eventual appending to B;      ...  O(N)
end
while any edge can be appended to B do                    ...  O(k.n)
begin
    for each appendable edge do                           ...  O(n^2)
    begin
        Select the edge (j,i) giving the highest score;
    end
    Append the edge (j,i) to the network B;
    for each appendable edge ending in node i do          ...  O(n)
    begin
        Update the score of its eventual appending to B;  ...  O(2^k.N)
    end
end
```

By the term „appendable edge“ I mean the edge which does not violate the acyclicity of dependence graph and does not exceed the maximum number of incoming edges k. The resulting time complexity of sequential BN construction can be expressed by the term:

$$O(n^2) \,.\, O(N) + O(k.n) \,.\, (O(n^2) + O(n) \,.\, O(2^k.N)) = \mathbf{O(k \,.\, n^3) + O(k \,.\, n^2 .\, 2^k .\, N)}$$

As shown in [87], the time complexity for generating ***N''*** new individuals (offspring) is only **O(*knN'*)**, where ***N''*** is proportional to the number of parents ***N.***

The time complexity for offspring evaluation depends on the complexity of fitness computation itself and in the case of additively-decomposable functions is ***O(nN')***.

If we consider k to be constant and N resp. N'' to be proportional to n (see [93] and [99] for discussion about relation between population size and problem size), then the time complexity of sequential BN construction is $\mathbf{O}(n^3)$, whereas the time complexities for offspring generation and evaluation are $\mathbf{O}(n^2)$. This confirms our empirical observations about time profile of BOA.

9.2.2 Parallel BN construction

The goal is to utilize more processors when searching for a good network. Our consolation is that the BDe metric is separable and can be written as a product of n factors, where i-th factor expresses the influence of edges ending in the variable X_i. Thus, for each variable X_i the set of parent variables Π_i can be determined independently. It is possible to utilize up to n processors, each processor corresponds to one variable X_i and it examines only edges leading to this variable. Note, that even for the estimation of local CPD each processor needs the full copy of parent population.

Usually we can use only m processors. If $m<n$, then each processor evaluates only edges ending in its subset of nodes. For example see Fig. 9.3 where n=6 and m=2, such that each processor manages incoming edges of n/m=2 nodes.

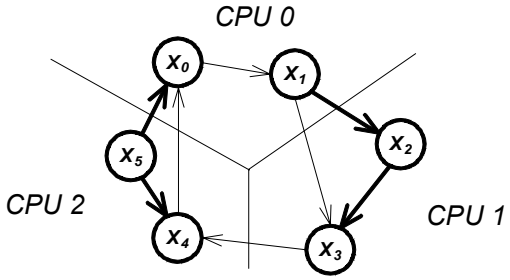


Fig. 9.3: CPU_0 is adding edges ending in nodes 0 and 1; CPU_1 is adding edges ending in nodes 2 and 3; CPU_2 is adding edges ending in nodes 3 and 4. Note that in the case of naïve parallel BN construction the acyclicity might be violated (dot line)!

In Fig. 9.3 you see that the naïve parallel BN construction might produce the unwanted cycles. The addition of edges is parallel, so we need an additional mechanism to keep the dependence graph acyclic. The most advantageous way how to handle this problem is to predetermine the topological ordering of nodes in advance. At the beginning of each generation, the random permutation of numbers $\{0,1,\ldots,n\text{-}1\}$ is created and stored in the $\boldsymbol{o}=(o_0,o_1,\ldots,o_{n-1})$ array. Each processor uses the same permutation (for example the initial seed of permutation generator is distributed via set of processors in the initial phase such that all generated permutations are identical). The direction of all edges in the network should be consistent with the ordering, so the addition of an edge from X_{o_j} to X_{o_i} is allowed if only $j<i$. Evidently, the variable X_{o_0} has no predecessor and is forced to be independent, thus the space of possible networks is reduced. To compensate this phenomenon the processors generate new permutation after each generation.

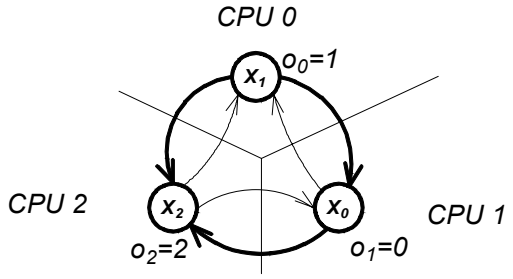


Fig. 9.4: Example of BN construction with m=n=3. Permutation vector ***o****=(1,0,2) determines the topological ordering of nodes such that the dashed edges are not allowed.*

9.2.3 Complexity of parallel BN construction

The advantages of predetermined ordering of nodes are noticeable. I completely removed the communication overhead even if the quality of generated network is almost the same as in the sequential case (see 9.2.4). Each processor can create its part of probabilistic model independently of the other processors, because the acyclicity constrains are implicitly satisfied. The parallel BN construction can be written as:

```
Start with an empty network B;
Generate random permutation o=(o0,…,on-1);
Assign ending nodes to each of m≤n parallel processes;
Distribute the permutation o among the parallel processes;
for each ending node oi do in parallel                          ...  O(⌈n/m⌉)
begin
   for j := 0 to i-1 do                                         ...  O(n)
   begin
     Compute the score of eventual appending (oj,oi) to B;      ...  O(N)
   end
   while incoming_node_degree d(oi) < min(k,i) do               ...  O(k)
   begin
      for each unapplied starting node oj having j<i do         ...  O(n)
      begin
        select the edge (oj,oi) giving the highest score;
      end
      Append the selected edge (oj,oi) to the network B;
      Remove the node oj from the set of unapplied parents of oi;
      for each unapplied starting node oj having j<i do         ...  O(n)
      begin
        Update the score of eventual appending (oj,oi) to B;... O(2^k.N)
      end
   end
end
```

The time complexity of parallel BN construction can be expressed by the term:

$$\mathbf{O(\lceil n/m \rceil).(O(n).O(N) + O(k).O(n) + O(k).O(n).O(2^k.N)) = O(k.n^2.2^k.N / m)}$$

After simplification it gets $\mathbf{O(n^3/m)}$. Note that a linear speedup can be reached, when compared with time complexity $\mathbf{O(n^3)}$ of sequential BN construction.

9.2.4 Validation of proposed BN construction

Theorem 9.1 (*Generality of model with restricted dependencies*) :
The restricted ordering of nodes does not affect the generality of induced model.

Proof: From the chain rule of probability we know that each joint probability can be factorized as

$$p(X_0, X_1, ..., X_{n-1}) = \prod_{i=0}^{n-1} p(X_i \mid X_0, X_1, ..., X_{i-1}) \qquad (9.1)$$

Assume the ordering permutation ($o_0, o_1, ..., o_{n-1}$). After reordering the variables the chain rule can be still applied to obtain factorization which does not violate the nodes ordering:

$$p(X_{o_0}, X_{o_1}, ..., X_{o_{n-1}}) = \prod_{i=0}^{n-1} p(X_{o_i} \mid X_{o_0}, X_{o_1}, ..., X_{o_{i-1}}) \qquad (9.2)$$

Note, however, that this proof does not consider the case of limited order of dependencies in BN, which might cause slight degradation of created model in parallel BOA. It should be also noted that the BDe metrics is not sensitive to directionality of edges in the dependency graph, which also justifies the usage of predetermined ordering of nodes.

9.3 Design of pipelined PBOA algorithm

9.3.1 Computing platform

The pipelined algorithm PBOA is the first of my three parallel BOA algorithms. All these algorithms use parallel dependency graph construction based on the principle of restricted ordering of nodes, they differ mainly in the way how the offspring is being generated. PBOA was proposed for fine-grained type of parallelism. The target platform for its implementation can be for example the Transputers with their tightly-connected communication channels, or one-purpose MIMD processor. Each processing element has its own local memory with complete copy of the whole population.

9.3.2 Pipelined processing

Let us consider usage of m=n processors (resp. processing elements). First the BN is constructed. The i–th processor determines the set of parent variables $\boldsymbol{\Pi}_{o_i}$ and it estimates the parameters of CPD for $p(X_{o_i} \mid \boldsymbol{\Pi}_{o_i})$. Fig. 9.5 shows an example of parallel construction of BN with n=4 variables and m=4 CPUs, including the parallel estimation of local CPDs.

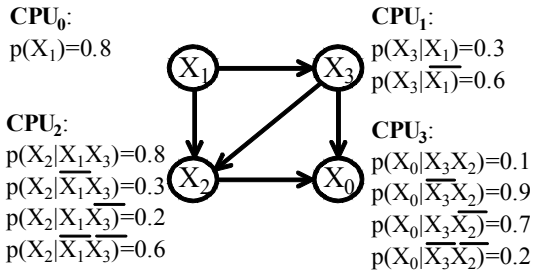


Fig. 9.5: An example - splitting of BN construction between 4 processing elements. The ordering permutation is $\boldsymbol{o}$*=(1,3,2,0), so the variable* X_1 *is independent. Parts of BN (including local CPD tables) which are estimated by different CPUs have different color shades.*

Now, let us focus on the pipelined offspring generation. PBOA can generate offspring in a linear pipeline way, because in the fine-grained architecture there are negligible communication latencies. It takes n cycles to generate the whole chromozome. For example let us consider the first chromozome. Its o_0-th bit is generated independently in processor number 0 at time t. Then, its o_1-th bit is generated in processor number 1 at time t+1 conditionally on the o_0-th bit received from neighbouring processor number 0, etc. Generally, X_{o_i} is generated in CPU number i at time $t+i$ conditionally on $\boldsymbol{\Pi}_{\boldsymbol{o}_i} \subseteq \{X_{o_0},...,X_{o_{i-1}}\}$. The advantage is that each CPD is sampled locally at the place where it had been estimated.

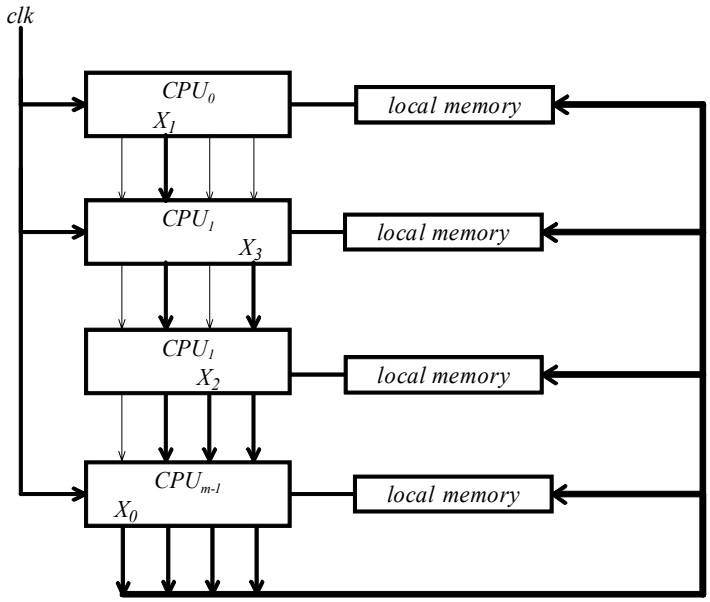


Fig. 9.6: Pipelined generation of ofspring according to the BN constructed in 9.5. The dashed arrows mean ungenerated variables.

After the offspring generation each processor has its full local copy of new population. The evaluation of new population is splitted into subsets and each processor evaluates different subset. The fitness values transfer uses the same channels as the generated chromozomes, so there is some slight delay before each processor receives each fitness value. Fortunately, most of this transfer can be overlapped with evaluation of next individuals, only small delay O(m) remains at the end. This delay is constant for given m, so this approach is scalable.

After the evaluation stage each processor replaces the better part of population to get the promising solutions required for estimation of local probabilistic model in the next generation. The complexity of *replace* operation is only $O(N \cdot \log N)$, so it is very fast and it can be done redundantly in all processors.

9.3.3 Timing diagram and workload balance

The detailed timing diagram for PBOA can be seen in Fig. 9.7. Note that in this concrete example the first processor deals with the variable X_1 independent, such that the set Π_1 is empty. Generally, the parents of variable X_{O_i} are chosen from the set of max. i-1 variables allowed. We see that the amount of work required to determine the parents of X_{O_i} increases with i. According to my empirical observation as well as the analysis of parallel BN construction algorithm from chapter 9.2.3, this amount of work grows almost linearly with i. This is why the number of evaluated individuals in each processor is inverse linearly proportional to i. This enables PBOA to reduce the synchronization delay before the final *select* operation, which can be understood as the barrier type of synchronization.

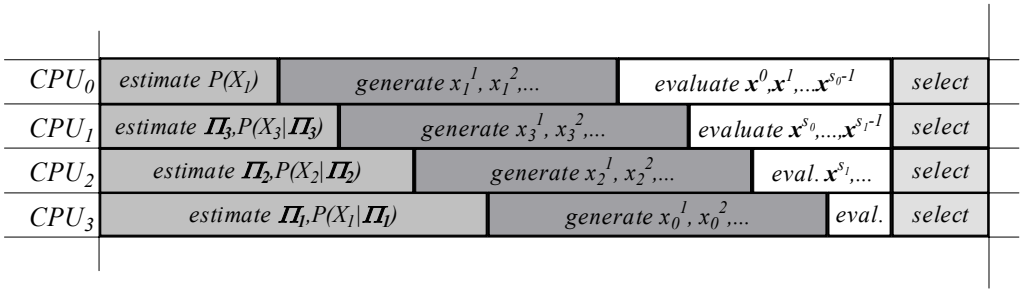


Fig. 9.7: The complete PBOA cycle for the BN example from Fig. 9.5.

For large problems the proportion between BN construction time and the remaining time increases, as follows from the complexity of parallel BN construction (see chapter 9.2.3). Sometimes the evaluation operation does not suffice to reduce the synchronization time before *select*.

In this case some workload balance is necessary. Remember that the time for determining Π_{O_i} is proportional to i. Let us for example consider $m=n/2$, so each processor is responsible for 2 nodes. We can re-arrange the pairs of nodes such that the first processor deals with X_{O_0} and $X_{O_{n-1}}$, the second processor deals with X_{O_1} and $X_{O_{n-2}}$, etc. Due to this re-arrangement the sum of work for estimating each pair of CPDs is the same, see Fig. 9.8. Of course, the offspring generation now requires bidirectional communication channels, because the ordering of processors which confirms topological ordering of nodes is now $CPU_0 \rightarrow CPU_1 \rightarrow \ldots \rightarrow CPU_{m-1} \rightarrow CPU_{m-1} \rightarrow CPU_{m-2} \rightarrow \ldots \rightarrow CPU_0$.

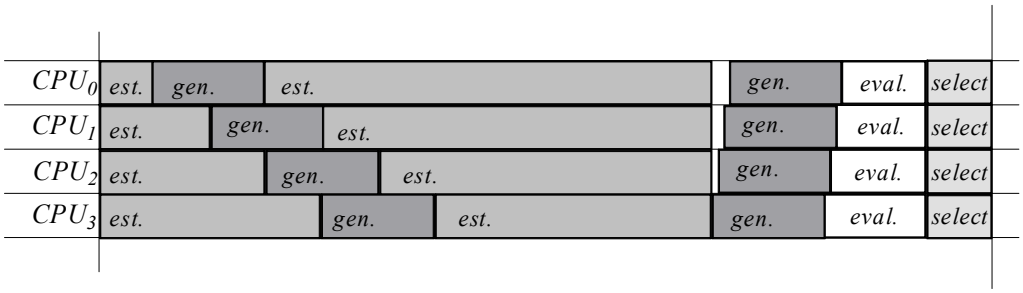


Fig. 9.8: Workload balance principle for pairwise re-arrangement.

It is also possible to use another type of balancing (non-equally sized groups of nodes) if $m<<n$, see Fig. 9.9. The i-th boundary for grouping the nodes can be computed approximately as $\sqrt{n^2 i / m}$.

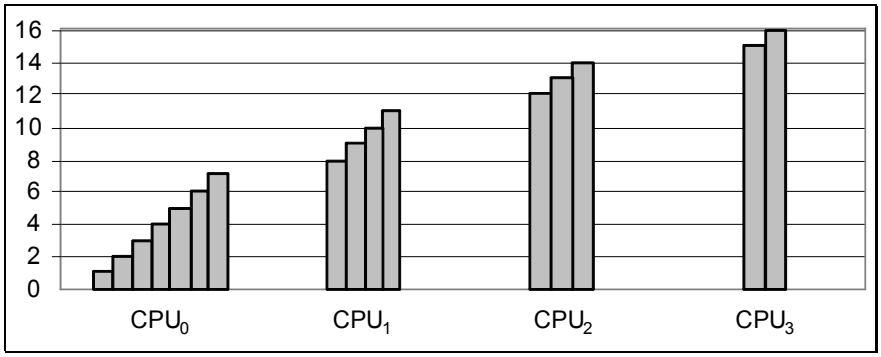


Fig. 9.9: Workload balance principle for m<<n.

9.3.4 Empirical results

No fine-grained architecture was available at the time of PBOA design, so I simulated the PBOA principles using the Sun Enterprise 450 server. The experiments were mainly focused on empirical proof of usability of restricted ordering of nodes (see the theorems in chapter 9.2.4). To investigate this I compared PBOA with the sequential Pelikan's BOA. Both algorithms use the limit of incoming edges k=3.

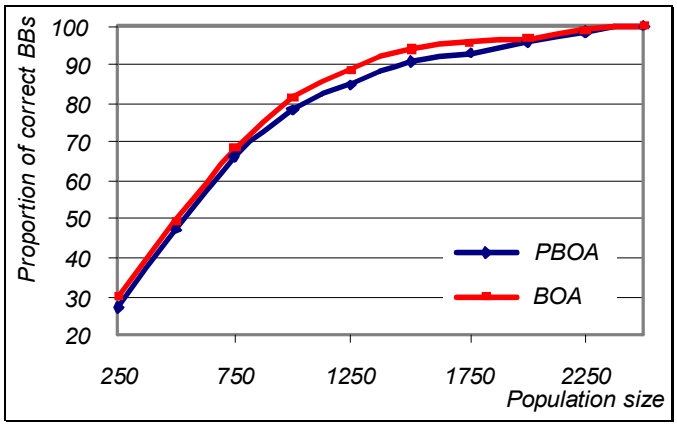


Fig. 9.10: Average final proportion of correct BBs for 3-deceptive function.

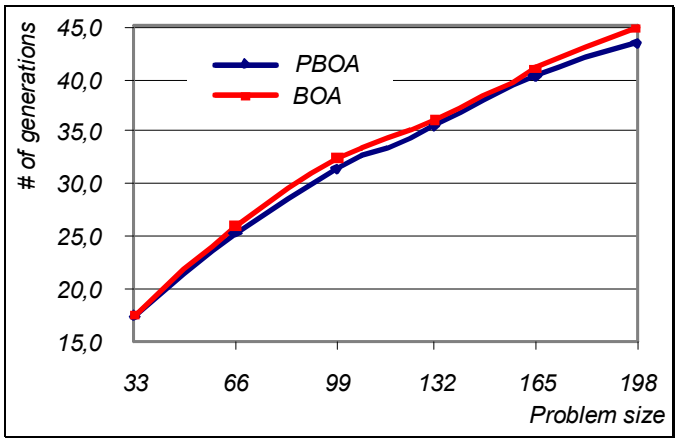


Fig. 9.11*: Average number of generations until convergence for 3-deceptive function. The population size was changed linearly according to problem size (from L=1000 for n=33 to L=6000 for n=198).*

Fig. 9.10 and 9.11 confirm that for 3-deceptive function the population size needed to get the same quality of solution as well as the number of generations until successful convergence are really not affected by the restricted ordering of nodes. The same result was also obtained for trap-5 function.

Tab. 9.1. The results of grid graphs bisectioning (see 7.1)

Graph name	Grid 100.2 with bottleneck	Grid 100.10
BOA: Min. population size for 90% success	2500	2400
PBOA: Min. population size for 90% success	2600	2500
BOA: Avg. # of generations until convergence	47,9	57,9
PBOA: Avg. # of generations until convergence	51,2	66,3
BOA: Average # of fitness evaluations	119750	138960
PBOA: Average # of fitness evaluations	133120	165750

The values in Tab. 9.1 indicate that for real problems like graph bisectioning the PBOA needs slightly higher population size and number of generations, but this is not critical because the additional work can be also done in parallel.

9.4 Design and implementation of distributed DBOA algorithm

9.4.1 Computing platform

My Distributed Bayesian optimization algorithm (DBOA) was proposed and implemented for coarse-grained type of parallelism. Its target platform is for example the cluster of workstations connected by fast Ethernet network. The experiments were run on the uniform cluster of Sun Ultra 5 workstations with UltraSPARC II processors at 270 MHz. The hardware multiport router has been Summit48 (from Extreme Networks), which is able to transfer up to 100 Mb/s between pairs of distinct workstations simultaneously. I used the MPI C++ library version 1.2 which provides the well known standard for message passing communication.

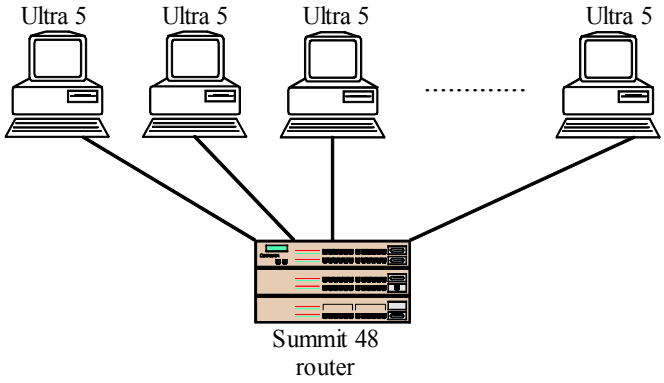


Fig. 9.12: DBOA platform.

9.4.2 Distributed processing

At the beginning of DBOA cycle each processor has the full copy of the parent population and it constructs the part of BN dependence graph, in the same way like PBOA. Unfortunately, the communication delays are too long to use pipelined generation of offspring. Thus, DBOA uses distributed generation of offspring, each processor generates the whole subpopulation of chromozomes. See Fig. 9.13 for comparison of both types of offspring generation.

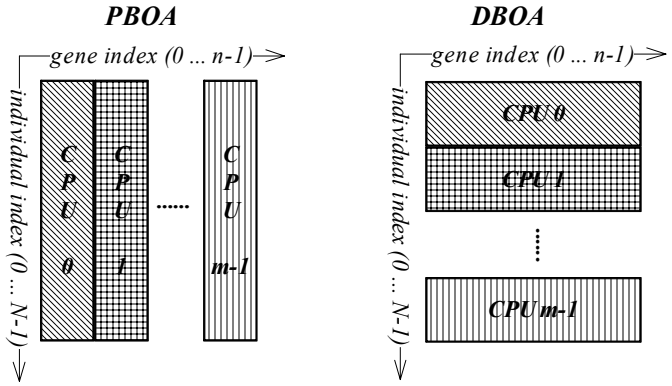


Fig. 9.13: Comparison of offspring generation. PBOA distributes the work between processors „horizontally“ and DBOA „vertically“.

Note that for this kind of offspring generation each processor needs to use the complete probabilistic model, which is constructed piecewise. Thus, a necessary step between model estimation and offspring generation is the gathering of local parts of model. Each processor exchanges its part of model with the other processors. To reduce the amount of communication between processors I propose to exchange only the parts of dependence graph, not the parameters of local CPDs. Remember that it is the most time consuming task in parallel BOAs to determine each set Π_i, but once this set was determined the parameters of local CPDs can be estimated quite fast. Note that Π_i can be specified using only array of k integers, where k is the maximal number of incoming edges, whereas parameters of local CPD grow exponentially with k, they need 2^k floating point numbers. For DBOA implementation it seems to be more advantageous to re-estimate the CPD parameters in each processor redundantly after receiving Π_i.

9.4.3 Implementation – MPI library

The Message Passing Interface (MPI) is a portable message-passing standard that facilitates the development of parallel applications and libraries. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in Fortran 77 or C. The available implementations run on both tightly-coupled, massively-parallel machines (MPPs), and on networks of workstations (NOWs).

Though much of MPI serves to standardize the ``common practice'' of existing systems, MPI has gone further and defined advanced features such as user-defined datatypes, persistent communication ports, powerful collective communication operations, and scoping mechanisms for communication. No previous system incorporated all these features.

9.4.4 Pairwise versus collective communication

MPI provides a set of pairwise send and receive functions that allow the communication of typed data with an associated *tag*. Typing of the message contents is necessary for heterogeneous support - the type information is needed so that correct data representation conversions can be performed as data is sent from one architecture to another. The tag allows selectivity of messages at the receiving end: one can receive on a particular tag, or one can wild-card this quantity, allowing reception of messages with any tag. Message selectivity on the source process of the message is also provided.

If the communication does not need to be overlapped with the computation, we can use *blocking* send and receive functions. The send call blocks until the send buffer can be reclaimed (i.e., until the sender can safely over-write the contents of message). Similarly, the receive function blocks until the receive buffer actually contains the contents of the message. MPI also provides *non-blocking* send and receive functions that allow the possible overlap of message transmittal with computation, or the overlap of multiple message transmittals with one-another. Non-blocking functions always come in two parts: the posting functions, which begin the requested operation; and the test-for-completion functions, which allow the application program to discover whether the requested operation has completed.

Question about the nature of the underlying protocol implementing the communication arises: if the send has completed, does this tell us anything about the receiving process? Can we know that the receive has finished, or even, that it has begun? Both blocking and non-blocking point to point communications have modes. The mode allows one to choose the semantics of the send operation and, in effect, to influence the underlying protocol of the transfer of data. In *standard* mode the completion of the send does not necessarily mean that the matching receive has started, and no assumption should be made in the application program about whether the out-going data is buffered by MPI. In *buffered* mode the user can guarantee that a certain amount of buffering space is available. The catch is that the space must be explicitly provided by the application program. In *synchronous* mode a rendezvous semantics between sender and receiver is used. Finally, there is *ready* mode. This allows the user to exploit extra knowledge to simplify the protocol and potentially achieve higher performance. In a ready-mode send, the user asserts that the matching receive already has been posted.

Collective communications transmit data among all the processes specified by a communicator object. One function, the barrier, serves to synchronize processes without passing data. Briefly, MPI provides the following collective communication functions.

- barrier synchronization across all processes
- broadcast from one process to all
- gather data from all to one
- scatter data from one to all
- allgather: like a gather, followed by a broadcast of the gather output
- alltoall: like a set of gathers in which each process receives a distinct result
- global reduction operations such as sum, max, min, and user-defined functions
- scan (or prefix) across processes

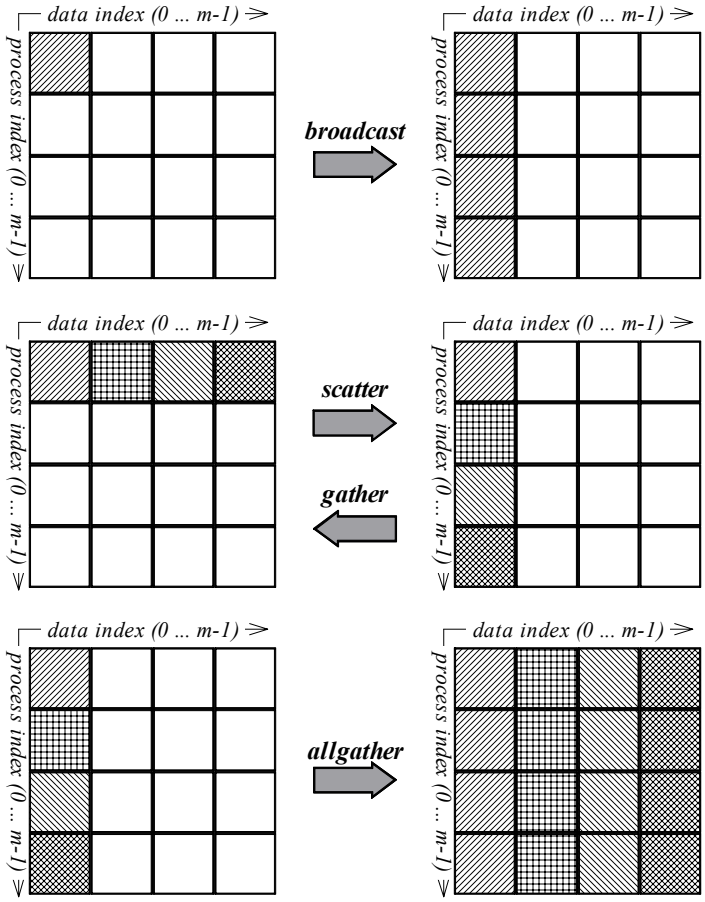


Fig. 9.14*: Comparison of collective communication.*

Figure 9.14 gives a pictorial representation of broadcast, scatter, gather and allgather. Many of the collective functions also have „vector“ variants, whereby different amounts of data can be sent to or received from different processes.

9.4.5 Workload balancing and usage of nonhomogeneous computational resources

For DBOA it is also possible to use workload balancing shown in Fig. 9.9, but this fixed balancing is unusable if the computing environment is shared with other tasks. For example imagine that all the Sun workstations are dedicated to Web browsing and only the background time is used for network computation. In this case a dynamic workload balancing needs to be used. I implemented the farmer-workers model. CPU_0 acts as a farmer which is responsible for job scheduling. Using the MPI routines, the whole DBOA cycle can be designed as follows:

```
Generate random permutation o=(o0,…,on-1) in CPU0;
Distribute the permutation o from CPU0 to CPU1…CPUm-1 using Broadcast();
Fill the job-queue in CPU0 with all job numbers 0…n-1 in decreasing order;
for k:=1 to m-1 do in each CPUk parallel
begin
  while job-queue in CPU0 is not empty do
  begin
    receive next job number i from CPU0;
    Determine Π_{o_i} ⊆ {X_{o_0},...,X_{o_{i-1}}} from Parents(t);
  end
  Put the dependence graph together from local sets Π_{o_i} using Algather();
  for j:=0 to n-1 do
  begin
    Estimate CPD parameters of p(X_{o_j} | Π_{o_j}) from Parents(t);
  end
  Generate subpopulation Offspringk(t);
  Put global Offspring(t) together from Offspringk(t) using Allgather();
  Evaluate subpopulation Offspringk(t);
  Put the fitness together from evaluated subpopulations using Allgather();
  Parents(t+1) := select the best (Parents(t) ∪ Offspring(t));
end
```

According to my empirical observations, the more time-consuming jobs (jobs of determining Π_{o_i} with higher number i) should be performed first, because the processes are better synchronized before the barrier if performing finer final steps.

CPU_0 denotes a farmer task which is not used for computation. Most of its time the farmer task is waiting for a job request. It is suspended until an interrupt from communication subsystem occurs. Fortunately, the „mpirun“ command used for executing DBOA allows us to specify which task is mapped to which workstation. I map the farmer task and the first worker task to the same workstation, such that only negligible computational power is wasted by farmer. For calculation of effectivity and speedup of DBOA I use the number of workstations instead of the number of tasks.

Note that in previous paragraphs I discussed only the balancing of workload during construction of dependence graph. Another balancing needs to be proposed for generating and evaluating of offspring. Each $Offspring_k(t)$ subpopulation can be of different size to better reflect the non-homogeneous computational environment. Unfortunately the size of each subpopulation has to be determined in advance before generating and evaluating starts. Thus, in the case of offspring generation and evaluation the workload distribution is not dynamical, it is changed only from one generation to another.

I used the timing routine *Wtime()* from MPI to measure the computational resources of each process and the results from the actual generation are used to modify the workload balance in the next generation.

I use the following equation which assumes that the amount of work is proportional to the number of offspring in the sub-population to be generated and evaluated. Let m denote the number of workstations, $s_i(t)$ denotes the size of subpopulation in i-th workstation in generation number t and $d_i(t)$ denotes the measured time of its generation and evaluation. Then, the $s_i(t+1)$ is calculated as

$$s_i(t+1) = N \cdot \frac{s_i(t)}{d_i(t).\sum_{j=0}^{m-1} s_j(t)/d_j(t)} \qquad (9.3)$$

This calculation is done in farmer and it is based on measurements from workers. Recomputed $s_i(t+1)$ values are sent to workers using *Scatter()* at the beginning of next DBOA cycle. During the first DBOA cycle the workload balance is uniform.

9.4.6 Collective communication improvement

In chapter 9.4.2 I already mentioned that the gathering of local parts of model Π_i is not buffer-space demanding operation, so classical *Allgather()* operation is sufficient. In the case of gathering of offspring subpopulations the situation gets more complicated. In typical problems the chromosome consists of several hundreds of genes and the population consists of several thousands of chromosomes. Thus, the population is quite large and it is difficult to gather it quickly among the processes. The collective MPI communication cannot be used because of performance degradation:

- the collective communication is far more slower than the pairwise communication
- the collective communication routines do not support non-blocking processing

I proposed and implemented a special stepwise algorithm to replace the *Allgather()* routine. In the i-th step each worker number j sends the known part of population to the worker number $j+2^{(i-1)}$ and receives the next part from worker number $j-2^{(i-1)}$. Thus, the communication distance and the number of transferred subpopulations in each step doubles. The principle is shown in the next figures.

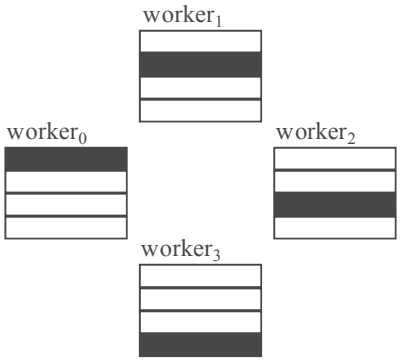


Fig. 9.15: Example of initial state with 4 distributed subpopulations (black fields).

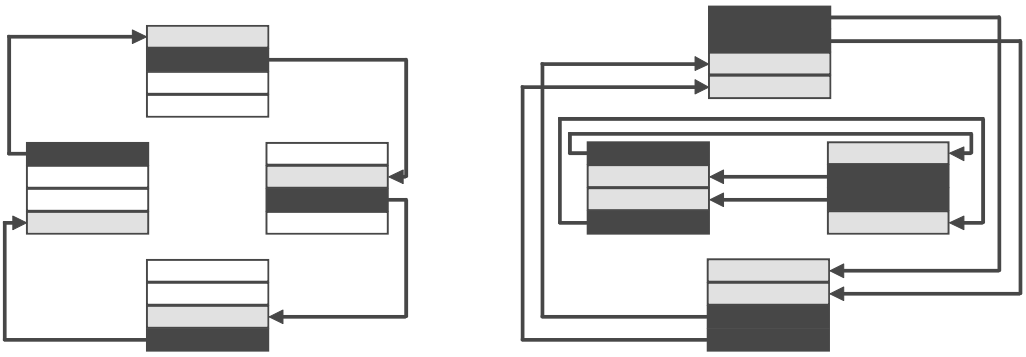


Fig. 9.16: Two steps of gathering of the offspring. The subpopulations generated or received in the previous step are marked black, the transferred subpopulations are gray.

Advantages of this approach are

1. The full pairwise transfer speed is used.
2. The operation is not atomic, but it consists of non-blocking operations, so the population transfer can be overlapped by fitness computation.
3. The number of initiation latencies is only log(m).

9.4.7 Timing diagram

Fig. 9.17 shows a timing example of one DBOA cycle run on 3 workstations. See that the last workstation seems to be less powerful or seems to be used also by other tasks, such that only residual computational resources are used for BN construction and BN sampling. The workload balancing methods ensure that all workstations finish the BN construction and the BN sampling in the same time, because BN gathering or offspring gathering steps can be considered as the barrier synchronization.

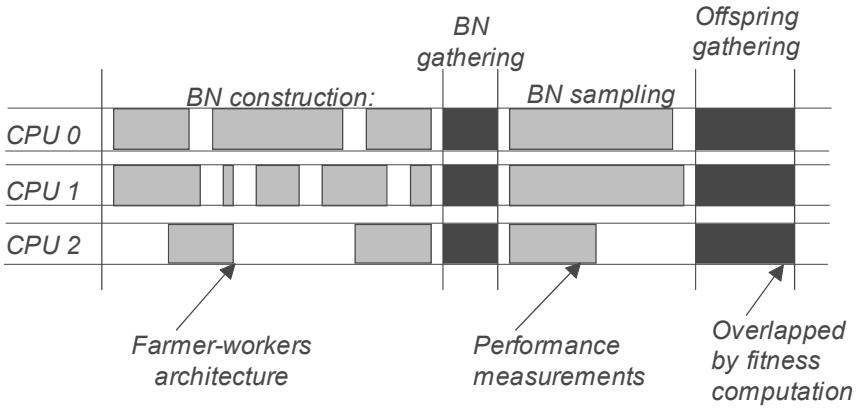


Fig. 9.17: Example of timing diagram for one DBOA cycle.

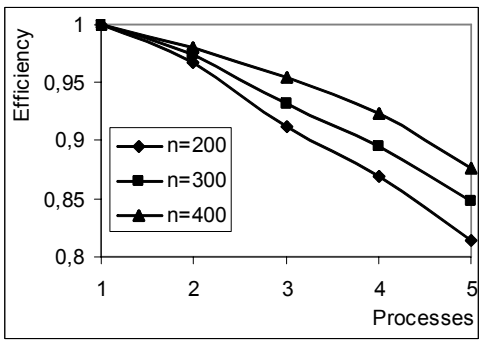
9.4.8 Results

For testing I used several deceptive functions and two combinatorial problems - knapsack problem and graph partitioning (see chapter 7). I examined the performance of whole DBOA and the performance of parallel BN construction alone.

Tab. 9.2. The results for 3-deceptive function (7.4), n=198, L=5000

	m=1	m=2	m=3	m=4	m=5	m=6
Avg. execution time for 1 generation [s]	42.64	22.03	15.59	12.25	10.47	9.22
Speedup S	**1.00**	**1.94**	**2.73**	**3.48**	**4.07**	**4.63**
Efficiency 100S/P [%]	100	96.8	91.2	87.0	81.4	77.1
Avg. BN construction time [s]	40.97	20.91	14.19	10.88	9.00	7.81
Speedup S	**1.00**	**1.96**	**2.89**	**3.76**	**4.55**	**5.25**
Efficiency 100S/P [%]	100	98.0	96.3	94.1	91.0	87.4

I also found that for the case of the geometric random graph partitioning the efficiency of the parallelization increases when the problem size grows (Fig. 9.18a) and that this efficiency is nearly not affected by the population size (Fig. 9.18b).



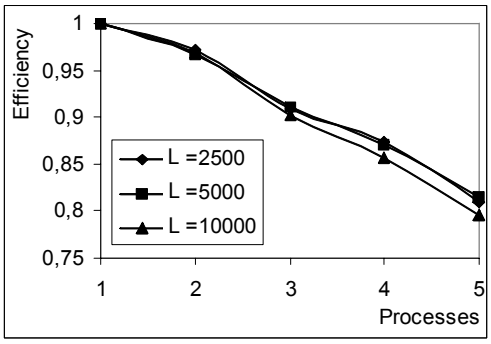


Fig. 9.18a -*Efficiency of the parallelization with the problem size n as a parameter (for fixed L=5000).*
Fig. 9.18b -*Efficiency of the parallelization with the population size L as a parameter (for fixed n=200).*

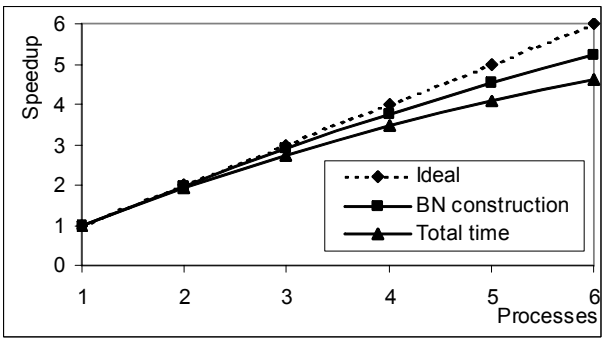


Fig. 9.19 *– Speedup results for 3-deceptive function, n=198, L=5000 (see Tab. 9.2)*

9.5 Design and simulation of multithreaded MBOA algorithm

9.5.1 Parallel MBOA analysis

In MBOA algorithm a set of binary decision trees/graphs is used to express the probability model. Many ideas for parallel MBOA algorithm can be adopted from the DBOA design, see 9.4. Namely the concept of restricted ordering of nodes can be used again to keep the dependencies acyclic and to remove the need for communication during parallel construction of probabilistic model.

An ordering permutation $(o_0,o_1,...,o_{n-1})$ means that the variables $\{X_{o_0},...,X_{o_{i-1}}\}$ can serve as splits in the binary decision tree of target variable X_{o_i}.

The MBOA differs from BOA in the heterogeneous model parameters. The decision trees can be used also for continuous or mixed domains, so the decision nodes in the trees can have parameters of various types: real numbers for splitting on real-coded parameters, sequences of integers for splitting on categorical parameters, etc. Also the leaf nodes in the trees can contain various structured parameters: a list of Gaussian kernels, a list of Gaussian network coefficients, probability bias for binary leafs, etc. You see that the disadvantage of decision trees lies in their complicated structure. It is very difficult task for each process to convert the parameters of decision tree into one linear data stream, which has to be exchanged with the other processes.

It would be far more comfortable if each decision tree could be used exactly in the same place where it has been built. Thus, the goal of this chapter is to design the distributed MBOA algorithm, which does not need to exchange the decision trees between processes. It uses the distributed generation of offspring in the "vertical" manner (see Fig. 9.14 left), but it was designed for use on cluster of workstations. Multithreaded processing was proposed to overcome large communication latencies.

9.5.2 TRANSIM tool

I was looking for suitable way to implement multithreaded MBOA. Unfortunately the current implementation of MPI standard does not contain support for multithreaded processing. Another possibility was to use multithreaded nature of Unix system, but I was looking for concept at higher level of abstraction. Finally, I decided to simulate the multithreaded MBOA algorithm using the well known TRANSIM tool.

Transim is a CASE tool used in the design of parallel algorithms. It can be used as a performance prediction tool for sizing parallel systems, a rapid prototyping tool and an aid to conceptualization. The major function of Transim is the prediction of time performance early in the life cycle before coding has commenced - obtaining this information early enables the sizing of parallel systems to be carried out prior to purchasing hardware or writing extensive code.

Although the general ideas behind the toolset could be applied in many fields of parallel processing, Transim itself is applicable only to loosely coupled medium grain machines - specifically those consisting of transputers connected with Inmos links.

The internal model, although somewhat stylised, is based on the transputer T800 family. Transim can simulate a single processor or a complex network. It can simulate a single process or a large application in which many individual processes are to be active on each processor (a processor is often termed a 'node' in Transim parlance). Parallel execution, alternation, channel communication, time-slicing, priorities, interruption, concurrent operation of links and the effects of external memory are taken into account.

The statements of the input language are of five categories: control, compile-time arithmetic for setting global parameters, software description including timing constructs, hardware description and mapping statements. The software description language is a subset of transputers' Occam language with various extensions. It is not intended as a general purpose programming language, it is too small subset for that - but to provide a control structure whereby the performance properties of an application may be expressed. The software and hardware are mapped together in a separate mapping section at the end of input file.

9.5.3 Multithreaded processing

The whole architecture of multithreaded MBOA is shown in Fig. 9.20. The *farmer* thread is responsible for dynamic workload assignment, *builder* threads are responsible for building decision trees and *generator* threads are responsible for generating new genes. The *buffer* threads are used only for buffering dataflow between builders and generators, because Transim does not support buffered channels. All the threads $builder_i$, $buffer_i$ and $generator_i$ are mapped to the same i-th workstation.

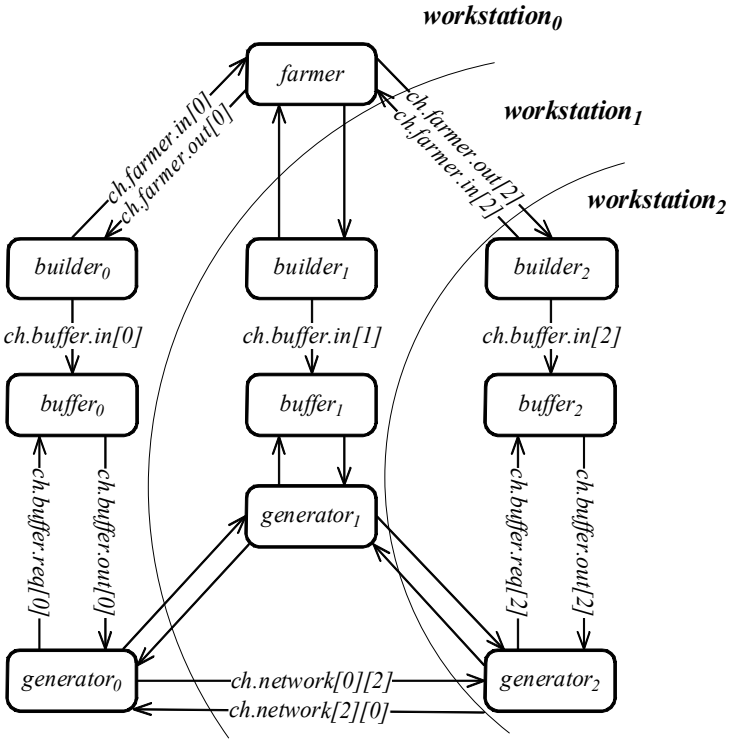


Fig. 9.20: Architecture of multithreaded MBOA in Transim. The dashed curves separate processes mapped to different hardware.

The Transim processes communicate through pairwise unidirectional channels. The array of external channels *ch.master.in* is used by *farmer* to receive the job requests and the array of external channels *ch.master.out* is used to send the jobs to *builders*. The two-dimensional array of external channels *ch.network* is used to transfer the partially-generated population between *generators*. The internal channels *ch.buffer.in*, *ch.buffer.req* and *ch.buffer.out* are used for exchange of local parts of model (decision tree) between *builder* and *generator*.

The principles of the most important threads *builder* and *generator* will be illustrated by fragments of Transim input file. The non-intuitive statements of input language will be explained as they appear.

First see the code of *builder* thread:

```
PLACED PAR i=0 FOR NUMWORKERS
  INT prev,next,j,prefetch.prev,prefetch.next,prefetch.j:
  SEQ | builder
    ch.farmer.in[i]  ! i | REQ.LENGTH
    ch.farmer.out[i] ? prefetch.j
    ch.farmer.out[i] ? prefetch.prev
    ch.farmer.out[i] ? prefetch.next
    WHILE prefetch.j < n
      SEQ
        j    := prefetch.j
        next := prefetch.next
        gene := prefetch.gene
        PRI PAR
          SEQ | getjob
            ch.farmer.in[i]  ! i | REQ.LENGTH
            ch.farmer.out[i] ? prefetch.j
            ch.farmer.out[i] ? prefetch.prev
            ch.farmer.out[i] ? prefetch.next
          SEQ | build
            ch.buffer.in[i] ! prev | WORK.LENGTH
            SERV(j*TIMESLICE)
            ch.buffer.in[i] ! j | WORK.LENGTH
            ch.buffer.in[i] ! next | WORK.LENGTH
```

Each *builder* requests for a job via *ch.farmer.in[i]* and it receives job info from *ch.farmer.out[i]*. More precisely, it receives the job number *j*, the number of workstation working on job *j-1* and the number of workstation requesting the *farmer* for job *j+1*. The useful computation is simulated by a *SERV()* command. You see that the time for building the decision tree for variable X_{o_j} is proportional to *j* and is scaled by *TIMESLICE* constant. The building of decision tree is overlapped by high priority communication (see the sequence named „getjob“) which serves for prefetching next job from *farmer*.

Now see the code of *generator* thread. This fragment is very simplified, ignoring the initial stage of generating independent variable and the final stage of broadcasting of full population:

```
PLACED PAR i=0 FOR NUMWORKERS
  INT from,to,population,j:
  SEQ | generator
    WHILE TRUE
      SEQ
        ch.buffer.req[i] ! i | WORK.LENGTH
        ch.buffer.out[i] ? from
        PAR
          SEQ | recvpred
            ch.network[from][i] ? population
          SEQ | recvmodel
            ch.buffer.out[i] ? j
            ch.buffer.out[i] ? to
        SERV(GENSLICE)
        ch.network[i][to] ! population | j * N
```

Each *generator* receives from *builder* (via buffered channels) the number of workstation from which it receives the population with genes $\{X_{o_0},...,X_{o_{j-1}}\}$ generated. Simultaneously it also receives from *builder* the decision tree for gene X_{o_j} and the number of consecutive workstation. Then the *SERV()* command simulates sampling of decision tree. According to my empirical observations the sampling time is almost constant, independent of *j*. At the end, the population with generated $\{X_{o_0},...,X_{o_j}\}$ is sent to consecutive workstation.

Note that the priority of threads is crucial. The *generator* thread should have high priority and the workstation processor should use the *builder* threads to overlap the communication latency between *generators*. But in principle, it is impossible to remove all the communication latencies completely (for example there remain no decision trees to build when generating the last gene $X_{o_{n-1}}$).

9.5.4 Simulated results

The Transim is able to simulate the capacity of real communication channels, so the speed of external channels was set to 100Mb/s (like the speed of Summit48 Ethernet switch) and their communication latency was set to 0.5 ms (like the average software latency in MPI). The speed of internal channels was set to maximum, because in the real implementation the *builder* and *generator* would communicate via shared memory space. According to my empirical experience with sequential MBOA on Athlon-600 processor, the TIMESLICE constant was set for about 7.5 ms and the GENSLICE was set for 10 ms. The job assignment does not need extra communication buffers, so the constants WORK.LENGTH and REQ.LENGTH were set to 1.

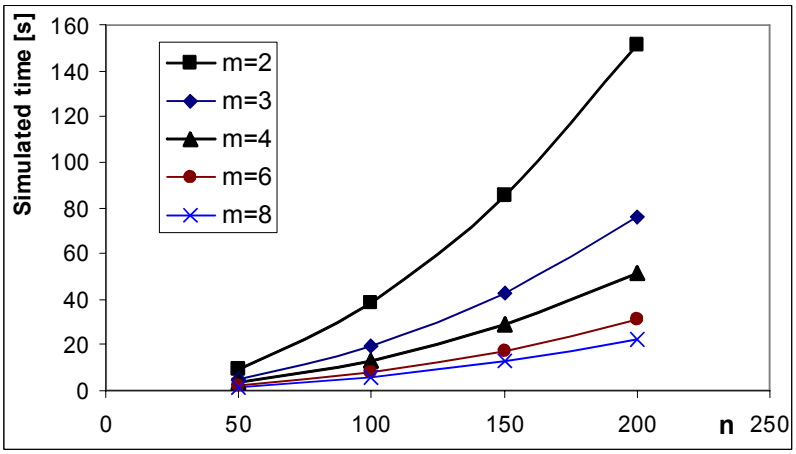


Fig. 9.21: The total simulated time of multithreaded MBOA for varying problem size n and varying number of processors m for fixed population size L=2200.

9.6 Knowledge-based KBOA algorithm

Previous sections discussed parallel processing as the brute force approach to shorten the optimization time. Another possibility to accelerate BOA is to use an additional knowledge about the problem, such that sooner convergence is achieved. This capability can be added to sequential BOA as well as it can be combined with parallel processing.

For many combinatorial problems the information about the structure of the problem is available. If the information is complete, the dependence graph can be constructed in-hand by the expert. This method is used in the FDA (see 4.4.1). An useful benchmark for knowledge-based experiments is a hypergraph partitioning (see 7.1). The structure of partitioning problem is given by the list of edges between hypergraph nodes, but the decomposition of the problem in the sense of FDA is not simple because the variables of the cost functions are overlapping. We can only expect that adjacent nodes in the hypergraph are dependent, but we have no exact knowledge about the degree of independence between non-adjacent nodes. The purpose of this chapter is to design a Problem Knowledge-based BOA algorithm (KBOA) which uses the partial (local) knowledge about the linkage to improve the quality of probabilistic model.

The KBOA algorithm was proposed in a cooperation with my supervisor. He is the first author of very successful paper [xi], which was cited in IlliGAl reports [96] and [99].

9.6.1 Utilization of prior knowledge

The prior information about the problem can be used in the BDe metric (see 3.3.5) in two ways.

First, let us consider the term $p(B|\xi)$ - the prior probability of the network B. According to [47] we use a simple assignment $p(B|\xi) = c\kappa^{\delta}$, where c is a normalization constant, $\kappa \in (0,1]$ is a penalty constant factor, and δ is the number of edges in the Bayesian network having no match in given hypergraph. Remember that for practical reasons we use logarithm of BDe metric, so

$$\log p(B|\xi) = \log(c) + \delta.\log(\kappa) \qquad (9.4)$$

If we omit the logarithm of normalization constant, we get $\delta.\log(\kappa)$. In fact, this means that we increase the value of BDe metric by adding some penalty value $\log(\kappa)$ for all added edges that have no counterpart in the partitioned hypergraph. A very strong penalization $\log(\kappa)=10^6$ is used, so the greedy algorithm that constructs the dependency graph prefers edges only between adjacent nodes of hypergraph (local information about the problem). As lately as no neighbouring couples of nodes remain, the addition of edges continues for non-adjacent nodes of the hypergraph.

Secondly, the prior knowledge about the optimized problem is also represented by numbers $m'(\pi_{X_i})$ and $m'(x_i,\pi_{X_i})$. It can be shown, that the values of $m'(\pi_{X_i})$ and $m'(x_i,\pi_{X_i})$ express our belief in the accuracy of the values $m(\pi_{X_i})$ and $m(x_i,\pi_{X_i})$ (see 3.3.3 for the introduction to Bayesian estimation). When these numbers increase their value, the BD metric will tend more towards the prior assignment of distribution, when those numbers decrease the influence of sampled values $m(\pi_{X_i})$ and $m(x_i,\pi_{X_i})$ becomes more significant. However, in our preliminary experiments the usage of this type of knowledge has not significantly affected the optimization performance, so we omitted this second method.

9.6.2 Injection of building blocks

The standard BOA uses the random set of solutions in the initial population. In KBOA the initial solutions are affected by injection of clusters of predefined size. A predefined part of a population is generated by simple heuristics - the neighbouring nodes are aggregated into supernodes. An example of the detection of such a supernode in a hypergraph is shown in Fig. 9.22.

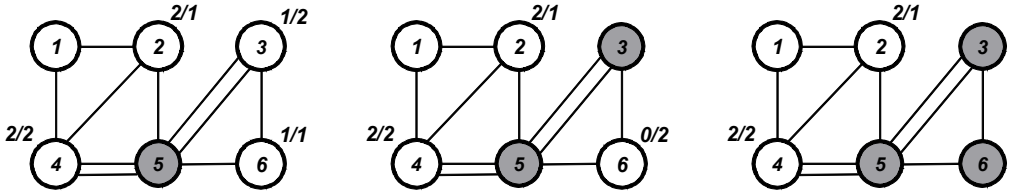


Fig. 9.22: Clustering technique based on the ratio of external/internal edges. Node 5 is the seed of the cluster, node 3 with the minimal ratio 1/2 is added, then node 6 with minimal updated ratio 0/2 is added.

First, a random node is selected to be the seed of the cluster. Then the effect of addition of each neighbouring node is expressed by the number of eventual external and internal hyperedges incident of this node. Each step the node having minimal ratio external/internal edges is selected. When maximum size of cluster is reached or all neighbours are selected, next cluster is created.

The cluster injection serves as a source of low-order building blocks. In the case of hypergraph bisection it stops when one half of the nodes are included in clusters. Then the nodes from clusters are assumed to be in one partition and the remaining nodes are in the second partition.

9.6.3 Results

The behaviour of KBOA was tested on the set of benchmarks, such as grid and random geometric graphs as well as real hypergraphs. We have performed various experiments to demonstrate the efficiency of the developed algorithm KBOA with comparison to the original BOA algorithm.

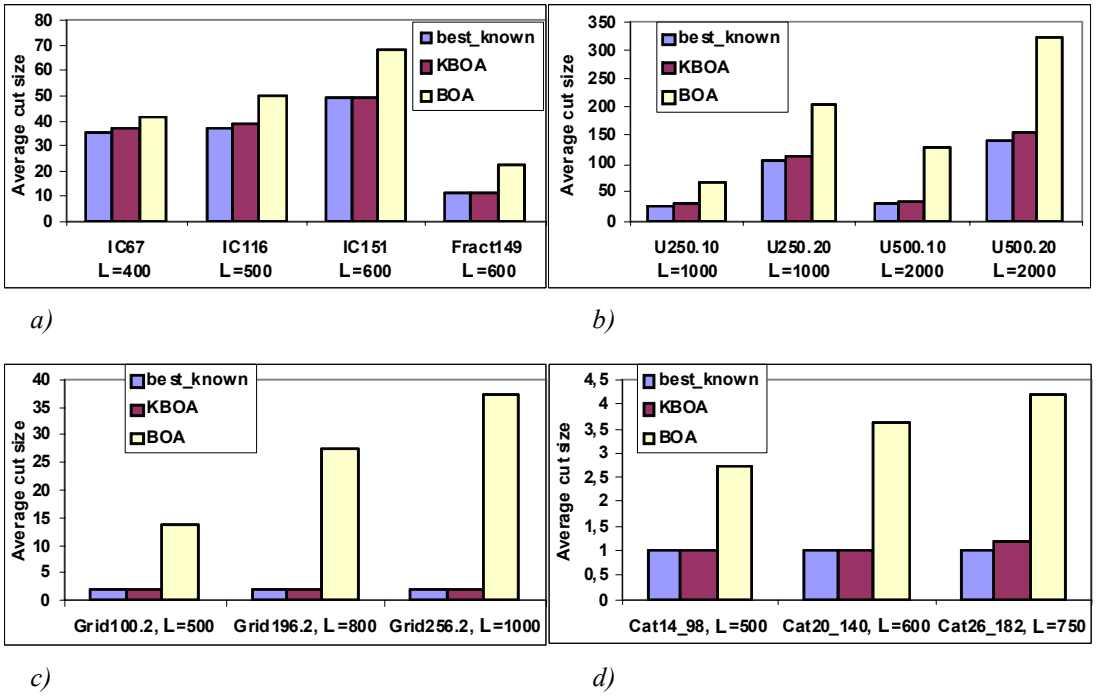


Fig. 9.23: Average and best known cut size for BOA and KBOA a) on real hypergraphs, b) on geometric random graphs c) on grid graphs, d) on caterpillar graphs

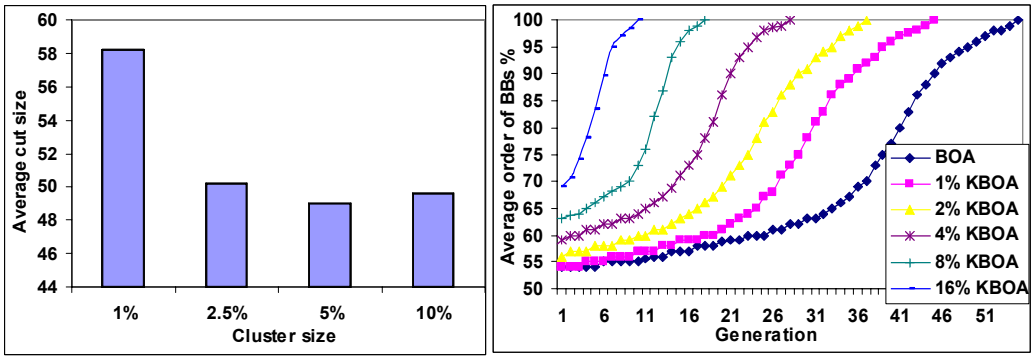


Fig. 9.24: Cluster size versus average cut size for IC151 *Fig. 9.25: Evolution of building blocks*

In Fig. 9.23 the comparison of BOA and KBOA is shown on four groups of graphs. The best known solutions are determined by KBOA itself or well known hMetis program [56]. We performed ten independent runs for KBOA and BOA on each of the 14 test graphs with different type and size. According to our results in [xiv] we limited the maximal incoming degree of BN nodes to k=3. The population size is the same for BOA and KBOA and it is settled to the value so that the KBOA was successful for all ten runs for the case of grid graphs. For the next types of graphs the population size is scaled linearly with the size of the problem (as suggested in [93]). The KBOA outperforms BOA in all tests of graph bisectioning. The time consumption for both algorithms is comparable. Let us note that BOA algorithm is theoretically capable to reach the similar cut size as KBOA but with about five times greater population size (for our test graphs) and rapid increase of computation time.

In Fig. 9.24 the performance of KBOA algorithm on the hypergraph IC151 for different size of injected clusters is shown. The size of clusters is expressed as a percentage of the number of hypergraphs nodes. The proper value is about 5% of what we used in all our trials. In case of smaller clusters the injection of proper building blocks is weak, in case of larger clusters the injection leads into local optima.

In Fig. 9.25 a presentation of average order of building blocks in the population is depicted. This experiment is done for grid graph Grid100.2 with cluster size as a parameter. Because two complement optimal bisections are known, we separate all the chromozomes into two groups according to their similarity to each of the possible optima and the calculation is based on the Hamming distance. It can be seen that for small cluster size the average order of BBs in the first generation is about 50%. For this experiment the population size is set to L=300-2400 for KBOA and to L=2400 for BOA to get the global optimum for all sizes of cluster. A strong influence of the injection of clusters on the speed up of the evolution can be recognized. Let us note that the 1% KBOA is affected only by the usage of prior probability (penalization) of the Bayesian network (see chapter 5) and only this phenomenon differs 1% KBOA from the original BOA.

The concept of cluster injection into the initial population detected on the hypergraph structure seems to be a really promising tool for enhancement of population genotype. This phenomenon leads to the meaningful reduction of number of cost function evaluations and population size.

10 Design of multiobjective EDA

Many real-world problems have multiple non-commensurable and often competing objectives. While in the case of single-objective optimization the optimal solution is simply distinguishable, this is not true for multiobjective optimization. The desired set of solutions should contain all possible trade-offs among the multiple, competing objectives. These solutions are optimal, nondominated, in that there are no other solutions superior in all objectives. The main attention in this chapter is focused on the incorporation of well known multiobjective niching techniques into classical structure of BOA to find the so called Pareto optimal set. I have designed two variants of multiobjective BOA according to Pareto-strength concept and epsilon-dominance concept.

10.1 Introduction

10.1.1 Motivation

Research in the design of multi-objective evolutionary algorithms has mainly focused on the fitness assignment and selection part. In contrast, the variation operators could be used from the single objective case without modification, which gave them only little attention. Some studies indicate, however, that the existence of multiple objectives influences the success probabilities of mutations which in turn has consequences for the choice of the mutation strength [105],[72]. For recombination it is also unclear whether combining parents that are good in different objectives improve the search as they could create good compromise offspring [58], or whether they contain such incompatible features that a combination does not make sense, thus advocating mating restrictions. The fact that recombination generally is a „contracting“ operator might also conflict with the goal to reach a broad distribution of Pareto-optimal solutions.

In this chapter I investigate the use of EDAs for multi-objective optimization problems to overcome the aforementioned difficulties when creating offspring from a set of diverse parents from different trade-off regions. Moreover, the utilizing of Estimation of Distribution Algorithms reduces the disruption of building blocks and the removes necessity of ad hoc setting of parameters like crossover, mutation and selection rate.

10.1.2 Problem specification

To formalize the multiobjective optimization a number of relevant concepts (see [123]) should be defined.

Definition 10.1 (M*ultiobjective optimization problem*) :
A general multiobjective optimization problem MOP can be defined as a vector function $\boldsymbol{f}$ that maps a tuple of n parameters to a tuple of m objectives:

$$\begin{aligned} &\textit{optimize} && \boldsymbol{z} = \boldsymbol{F(x)} = (F_1(\boldsymbol{x}), F_2(\boldsymbol{x}), \ldots, F_m(\boldsymbol{x})) \\ &\textit{subject to} && \boldsymbol{g(x)} = (g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), \ldots, g_k(\boldsymbol{x})) \preceq \boldsymbol{0} \end{aligned} \tag{10.1}$$

where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1}) \in \boldsymbol{A}^n$, $\boldsymbol{z} = (z_1, z_2, \ldots, z_m) \in \boldsymbol{R}^m$. Then $\boldsymbol{x}$ is called decision vector, $\mathbf{A}^n$ is the parameter space, $\boldsymbol{z}$ is the objective vector, and $\boldsymbol{R}^m$ is the objective space. The constraint vector $\boldsymbol{g(x)} \preceq \boldsymbol{0}$ determines the set of feasible solutions.

The set of solutions of MOP includes all decision vectors for which the corresponding objective vectors cannot be improved in any dimension without degradation in another - these vectors form so called Pareto optimal front. The idea of Pareto optimality is based on the Pareto dominance relation.

Definition 10.2 (*Pareto dominance*) :
Let's consider – without loss of generality - a maximization MOP. A decision vector $\boldsymbol{a}$ dominates decision vector $\boldsymbol{b}$ *iff*

$$\forall i \in \{1,2,..,m\}: F_i(\boldsymbol{a}) \geq F_i(\boldsymbol{b}) \ \wedge \ \exists i \in \{1,2,..,m\}: F_i(\boldsymbol{a}) > F_i(\boldsymbol{b}) \qquad (10.2)$$

We denote this relation "$\boldsymbol{a} \succ \boldsymbol{b}$".

According to (10.2) the Pareto-dominance concept introduces irreflexive, antisymmetrical and transitive binary relation between objective vectors. If we add reflexivity (such that solutions with the same objective vectors are considered to dominate each other) then it forms a partial ordering relation on the space of objective vectors.

Definition 10.3 (*Pareto optimality)* :
The vector $\boldsymbol{a}$ is said to be Pareto optimal iff in a parameter space $\boldsymbol{A}^n$ there exists no vector $\boldsymbol{b}$ which dominates vector $\boldsymbol{a}$

$$\bar{\exists} \boldsymbol{b} \in A^n: \ \boldsymbol{b} \succ \boldsymbol{a} \qquad (10.3)$$

Definition 10.4 (*Pareto optimal set)* :
Pareto optimal set is the set of all Pareto optimal solutions

$$\{ \boldsymbol{a} \in A^n \mid \ \bar{\exists} \boldsymbol{b} \in A^n: \ \boldsymbol{b} \succ \boldsymbol{a} \ \} \qquad (10.4)$$

In the objective space the set of nondominated solutions lie on a surface known as Pareto front. The goal of the optimization is to find a representative sampling of solutions along the Pareto front.

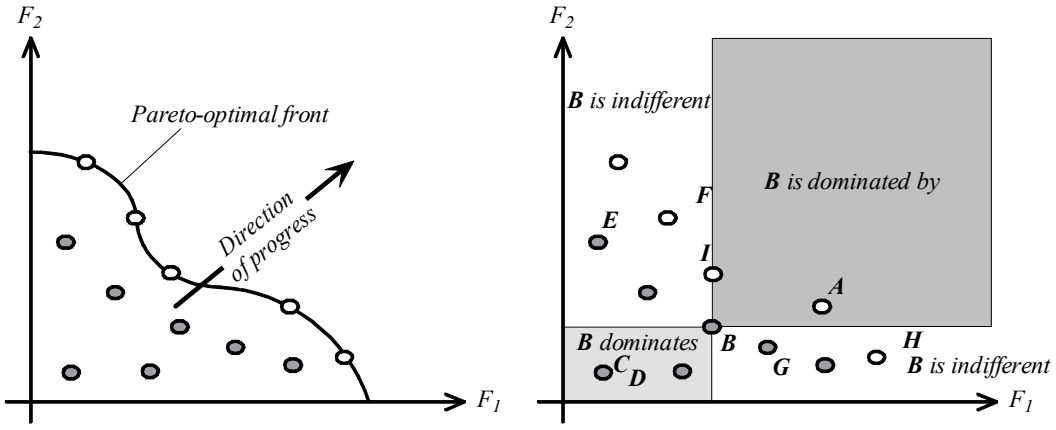


Fig. 10.1*: Illustration of Pareto optimality concept and Pareto dominance relation*

Definition 10.5 (*Pareto front)* :
Pareto front is the set of objective vectors of all Pareto optimal solutions

$$\{ \mathbf{F(a)}=(F_1(\mathbf{a}), F_2(\mathbf{a}),..., F_m(\mathbf{a})) \mid \mathbf{a} \in A^n \wedge \bar{\exists} \mathbf{b} \in A^n : \mathbf{b} \succ \mathbf{a} \} \quad (10.5)$$

Note that the Pareto optimal set is defined in the parameter space, while the Pareto front is defined in the objective space.

From example in Fig. 10.1 you see that with respect to solution *B* there exist some indifferent solutions (e.g. *E,F,G,H,I*), some solutions dominated by *B* (e.g. *C,D*) and some solutions which dominate *B* (e.g. *A*).

10.1.3 Traditional approaches

Historically, multiple objectives have been combined to form a scalar objective function through weighted sum of individual objectives or by turning objectives into constraints. Setting of weights and specification of penalty functions is not a simple task and these values can be found only experimentally.

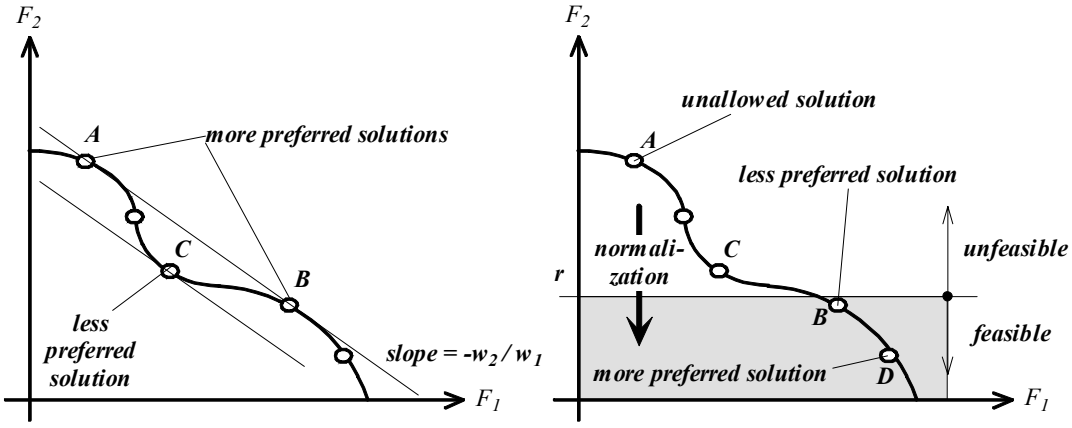


Fig. 10.2*: The disadvantages of traditional methods*

Weighted-sum method

In this approach the original vector-valued objective function is replaced by a scalar-valued objective function. The objective function of the individual $\boldsymbol{x}$ is computed as a weighted sum of all objective functions:

$$F(\boldsymbol{x}) = \sum_{i=1}^{m} w_i F_i(\boldsymbol{x}), \quad (10.6)$$

where w_i are weight coefficients. It is well known the sensitivity of the optimization process to these values. These algorithms do not preserve Pareto-optimal solutions but provide mostly solutions from extremes of the Pareto-front.

In Fig. 10.2 (left) there is an example with two-objective vectors. You see that equally preferred solutions are connected with the line of slope $-w_2/w_1$. This is not as general as the Pareto-dominance relation, because the *C* solution from Pareto-front is less preferred.

Constraint method

In this approach only one objective function $F_1(\boldsymbol{x})$ is used and the other objective functions are replaced by normalization operator which modifies each individual to keep its value in the considered bound. This operation can unfavourably change the objective function F_1 of each individual. This effect may cause an extra genetic drift of the population.

In Fig. 10.2 (right) there is an constraint example with two-objective vectors. You see that solutions *A,C* are not allowed even if they are from Pareto-front. Moreover, *D* is more preferred than *B*.

10.1.4 Earlier MOEAs

From the theory of Pareto optimal set it is evident that the multiobjective optimization algorithms should be able to find as many Pareto optimal solutions as possible. Many studies have depicted different ways how multi-objective evolutionary algorithms (MOEAs) can progress towards the Pareto-optimal solutions.

After the very preliminary Vector Evaluated Genetic Algorithm (VEGA) [109] the early MOEAs were inspired by Goldberg's idea of nondominated sorting [41] along with a niching mechanism. These include Multi-Objective Genetic Algorithm (MOGA) [33], Nondominated Sorting Genetic Algorithm (NSGA) [113] and Niched Pareto Genetic Algorithm (NPGA) [52]. All these successful algorithms work in a similar manner: (i) the fitness of a solution was assigned using the extent of its domination in the population and (ii) the diversity among solutions was preserved using a niching strategy.

10.1.5 Advanced MOEAs

In order to ensure convergence to the true Pareto-optimal solutions, an elite-preservation operator was absent in the earlier algorithms. Thus, the latter approaches mainly concentrated on how elitism could be introduced in an MOEA. This resulted in a number of better algorithms - Strength Pareto Evolutionary Algorithm (SPEA) [123], Pareto-Archived Evolution Strategy (PAES) [59], NSGA-II [29], and others. A wide review of basic approaches and the specification of original Pareto evolutionary algorithms include the dissertations [26], [115], [123]. From the latest period let me also mention an interesting Pareto-Envelope based Selection Algorithm (PESA [60]) and the very good algorithm SPEA2 [125] which approximates the true Pareto front.

10.1.6 EDA approaches

As far as I know presently there are only two other research groups working on multiobjective EDAs. The first one is that of Bosmann and Thierens. Their Multi-objective Mixture-based Iterated Density Estimation Algorithm (MIDEA) [117] combines the IDEA approach described in 4.5.5 with the approach similar to NSGA. The selection operator picks the best samples according to domination count and the diversity among solutions in the objective space is implicitly preserved due to clustering of solutions before construction of probabilistic model.

The MIDEA was published later than my papers concerning Pareto-strength BOA described in the next chapter (see [vii], [viii]), so no empirical comparison was possible. But I believe both algorithms would perform similarly because MIDEA is also comparable to the original SPEA algorithm I was inspired by. Bosman and Thierens have not assured the ability of MIDEA to converge to the true optimal Pareto-set. The diversity preservation in MIDEA is questionable, because each generations the solutions are "re-clustered". The centers of clusters would "shrink" nearer and nearer to each other if the population is likely to converge to an ``easy to find" middle region of the Pareto set. In this case some epsilon-dominance concept (proposed for BMOA algorithm in 10.3) will improve MIDEA.

The second paper on multiobjective BOA [57] was recently published by Nazan Khan from Illigal laboratory. It combines BOA with NSGA-II principle. From the theoretical point of view this paper contains no additional proposals concerning the unique properties of EDAs, but the remarkable think is the usage of multiobjective deceptive benchmarks in the experiments. These benchmarks completely confuse the NSGA-II algorithm with classical recombination operators.

10.2 Multiobjective BOA with Pareto-strength fitness

10.2.1 Pareto-strength principle

Pareto-strength BOA algorithm is a modification of original Pelikan's BOA core [91] based on Bayesian network model. This standard BOA is able to find mostly one optimal solution at the end of the optimization process, when the whole population is saturated by phenotype-identical individuals. To overcome this problem a promising Pareto-strength niching technique is applied, published in [123]. The space of objective vectors is divided into rectangles corresponding to the combinations of solutions from Pareto-set. The Pareto-strength fitness assignment tries to replace the objective vector by the scalar fitness value according the following two rules (see Fig. 10.3):

1. *Individuals dominated by smaller number of nondominated individuals are preferred.*
2. *Dominated individuals having more neighbours in their „niche" are penalized.*

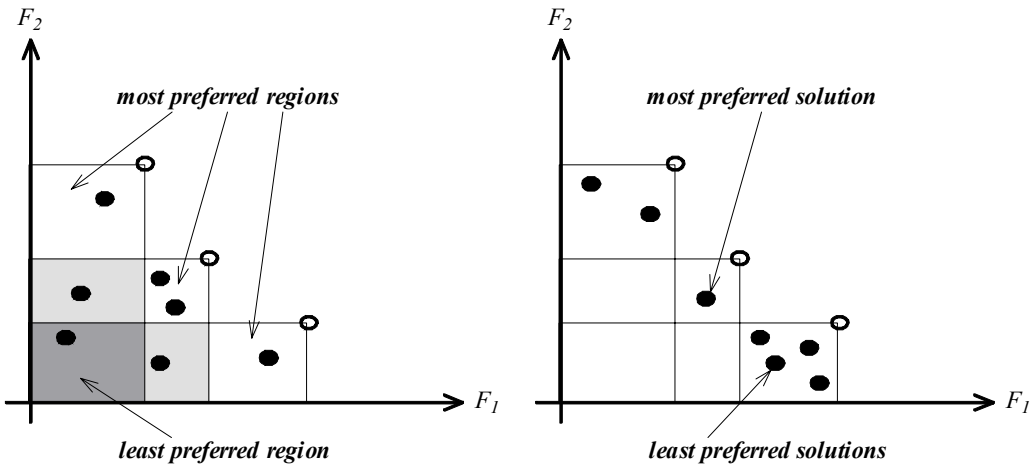


Fig. 10.3*: Illustration of intuitive rules for Pareto-strength fitness assignment*

10.2.2 Design of multiobjective BOA

Pareto BOA algorithm can be described by the following steps:

Step 1: ***Initialization:*** Generate an initial population P_0 of size L randomly.

Step 2: ***Fitness assignment:*** Evaluate the initial population.

Step 3: ***Selection:*** Select the parent population of size N as the best part of current population by 50% truncation selection.

Step 4: ***Model construction:*** Estimate the distribution of the selected parents using Bayesian network construction.

Step 5: ***Offspring generation:*** Generate new offspring (according to the distribution associated to the Bayesian network).

Step 6: ***Nondominated set detection and fitness assignment:*** Current population and offspring are joined, nondominated solutions are found, evaluated and stored at the top of the new population. Then dominated offspring and parents are evaluated separately.

Step 7: ***Replacement*:** The new population is completed by offspring and the best part of current population, so the worst individuals from current population are canceled to keep the size of the population constant.

Step 8: ***Termination:*** If maximum number of generations N_g is reached or stopping criterion is satisfied then the last Pareto front is presented, else go to Step 3.

The most important part of the Pareto BOA algorithm is the procedure for detection of nondominated solutions (current Pareto front) and sophisticated Pareto-strength fitness calculation. The procedure for current nondominated and dominated set detection is described in following steps:

1. For each individual $\boldsymbol{x}$ in the population P compute vector of the objective functions
$$\boldsymbol{F}(\boldsymbol{x}) = (F_1(\boldsymbol{x}), F_2(\boldsymbol{x}), \ldots, F_m(\boldsymbol{x})) \tag{10.7}$$
2. Detect subset of nondominated solutions
$$\overline{P} = \left\{ \boldsymbol{x}^j \mid \boldsymbol{x}^j \in P \wedge \exists \boldsymbol{x}^k \in P : \boldsymbol{x}^k \succ \boldsymbol{x}^j \right\} \tag{10.8}$$
Note: If two or more individuals have the same fitness vector $\boldsymbol{F}(\boldsymbol{x})$, then only one of them is accepted.
3. For each nondominated solution $\boldsymbol{x}^j$ compute its strength value as
$$s(\boldsymbol{x}^j) = \frac{\left| \left\{ \boldsymbol{x}^k \mid \boldsymbol{x}^k \in P \wedge \boldsymbol{x}^j \succeq \boldsymbol{x}^k \right\} \right|}{|P|+1} \tag{10.9}$$
The fitness for nondominated solutions is equal to the reverse of the strength value $F'(\boldsymbol{x}^j) = 1 / s(\boldsymbol{x}^j)$.
4. For each dominated solution $\boldsymbol{x}^i$ determine the fitness as
$$F'(\boldsymbol{x}^i) = 1 \Big/ \left(1 + \sum_{\boldsymbol{x}^j} s(\boldsymbol{x}^j) \right), \tag{10.10}$$
where $\boldsymbol{x}^j \in \overline{P} \wedge \boldsymbol{x}^j \succ \boldsymbol{x}^i$. In the original approach [123] all individuals dominated by the same nondominated individuals have equal fitness. I proposed an extension by adding a term $c.r(\boldsymbol{x}^i) / (|P|+1)$ into the denominator (10.10), where $r(\boldsymbol{x}^i)$ is the number of individuals from P (not only from nondominated solutions) which dominate $\boldsymbol{x}^i$ and

coefficient c is set to very small number, for example 0.0001. This term is used to distinguish the importance of individuals in the same "niche" (being dominated by the same nondominated solutions).

This type of fitness evaluation has the following advantages:

- For all nondominated individuals $F'(\boldsymbol{x}^i) \geq 1$, for dominated individuals holds $F'(\boldsymbol{x}^i) < 1$. If the replace-worst strategy is used, implicit Pareto elitism is included.
- Individuals from Pareto front dominated smaller set of individuals receive higher fitness, so the evolution is guided towards the less-explored search space.
- Individuals having more neighbours in their „niche" are more penalized due to the higher $s(\boldsymbol{x}^j)$ value of associated nondominated solution.
- Individuals dominated by smaller number of nondominated individuals are more preferred.

The whole structure of Pareto-strength based BOA algorithm is stated in Fig. 10.4.

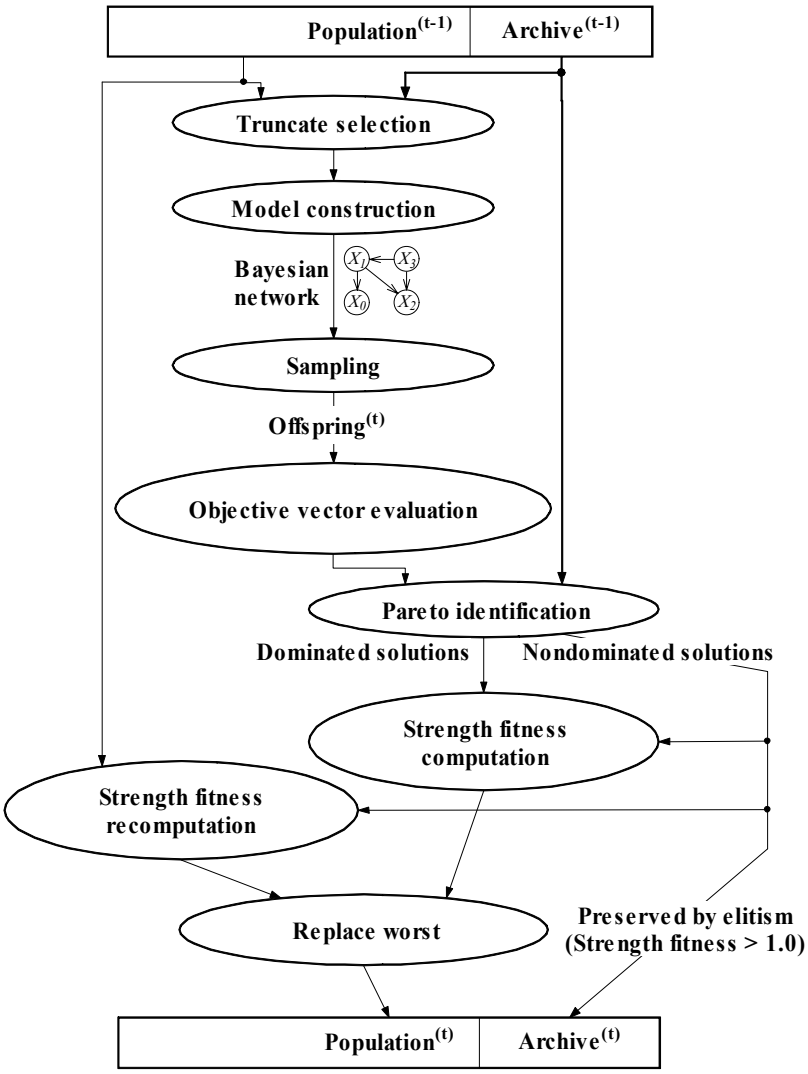


Fig. 10.4*: Architecture of Pareto-strength BOA*

10.2.3 Parallel Pareto-strength fitness assignment

In chapter 9.4 I proposed the distributed BOA algorithm. This approach can be extended to the distributed Pareto BOA. I propose the following modification of the procedure for Pareto-front detection and Pareto-fitness assignment:

First, each processor would compute the vector of objective functions for all individuals from its part P_k of population P. Then, each processor detects its local set of nondominated solutions $\overline{P}_k$ as

$$\overline{P}_k = \left\{ \boldsymbol{x}^j \mid \boldsymbol{x}^j \in P_k \wedge \overline{\exists} \boldsymbol{x}^i \in P_k : \boldsymbol{x}^i \succ \boldsymbol{x}^j \right\} \tag{10.11}$$

and the master processor creates the global nondominated set $\overline{P}$ from the union of local nondominated sets $\overline{P}_k$ as

$$\overline{P} = \left\{ \boldsymbol{x}^j \mid \boldsymbol{x}^j \in \bigcup_k \overline{P}_k \wedge \overline{\exists} \boldsymbol{x}^i \in \bigcup_k \overline{P}_k : \boldsymbol{x}^i \succ \boldsymbol{x}^j \right\} \tag{10.12}$$

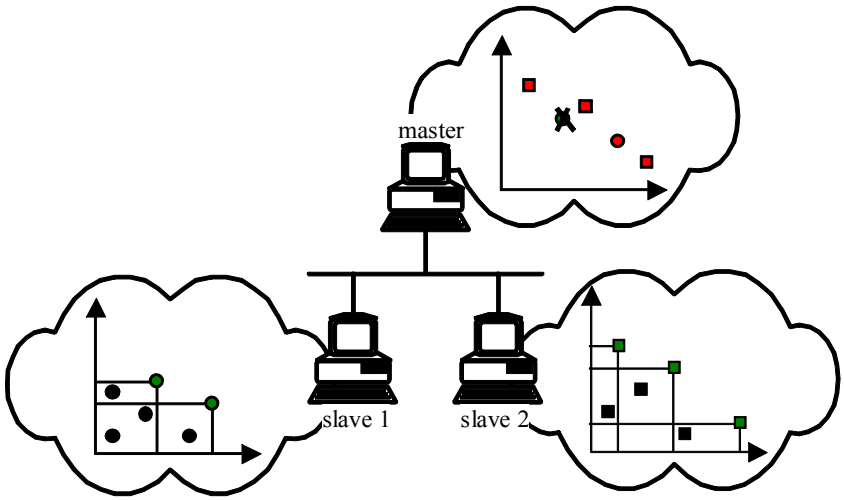


Fig. 10.5*: Parallel Pareto-set selection*

The strength values for nondominated solutions from $\overline{P}$ can be obtained as the sum of local strength values computed in parallel by all processors:

$$s(\boldsymbol{x}^j) = \frac{\sum_k \left| \left\{ \boldsymbol{x}^i \mid \boldsymbol{x}^i \in P_k \wedge \boldsymbol{x}^j \succ \boldsymbol{x}^i \right\} \right|}{|P|+1} \tag{10.13}$$

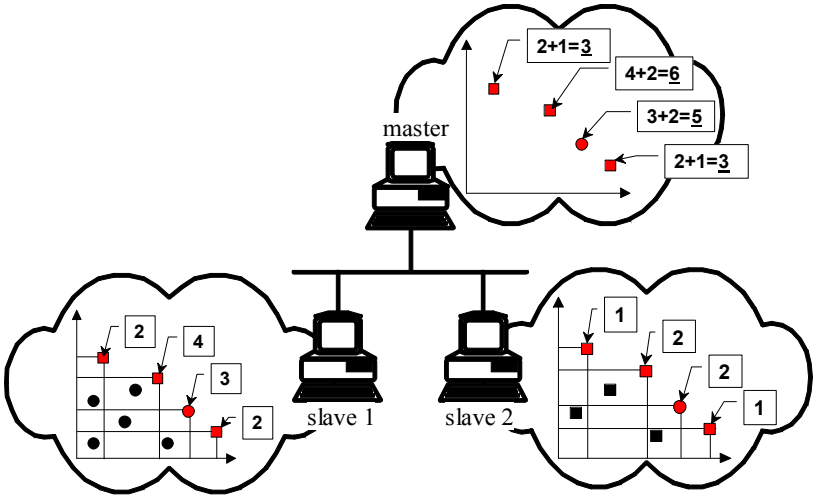


Fig. 10.6*: Parallel Pareto-strength computation*

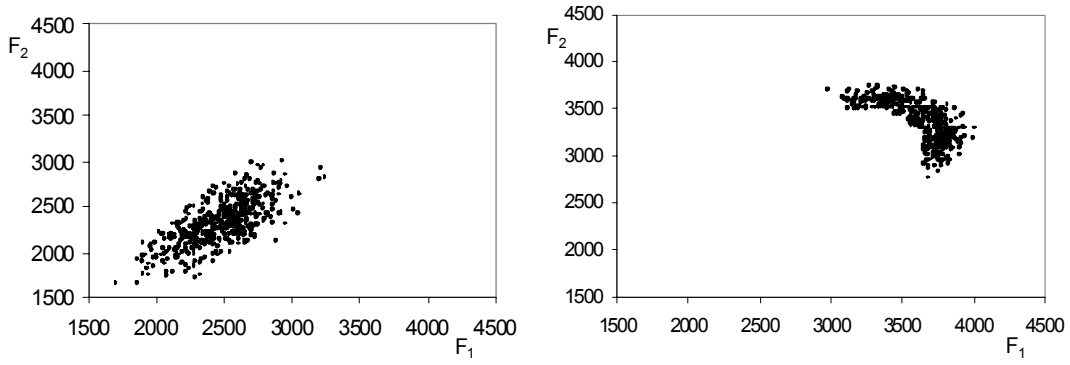
After that, all nondominated solutions and their strength values are known, so each processor is able to compute the Pareto fitness for all individuals from its part of population according to equations (10.9) and (10.10).

The first version of Pareto multiobjective algorithm is described in [vii], focused on bipartitioning of hypergraph. In the next chapter the problem of biobjective optimization of knapsack problem is described more in detail.

10.2.4 Comparison with advanced techniques on multiobjective knapsack problem

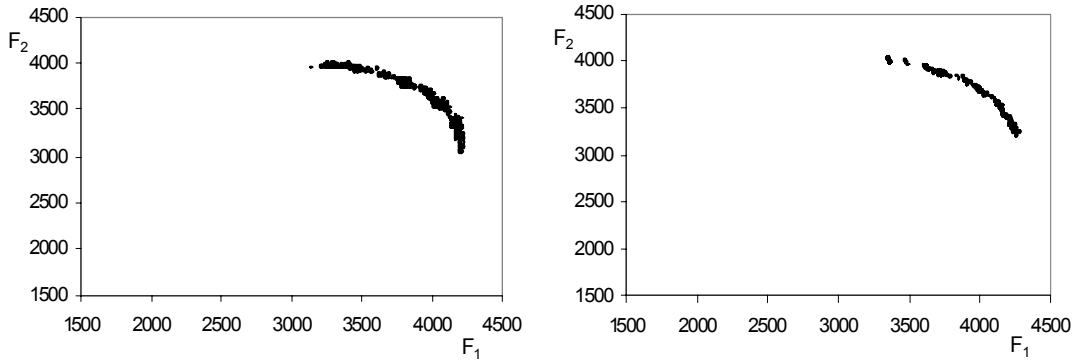
In this comparison I have focused on the bi-objective optimization of knapsack problem (see 7.3) which belongs to the well known test benchmarks. In order to be able to compare my algorithm with other published evolutionary algorithms I used two of the three benchmarks published on the web site [*http://www.tik.ee.ethz.ch/~zitzler/testdata.html*] specified for 100 (Kn100) and 250 items (Kn250) and two knapsacks. I have compared my results with published results achieved by two evolutionary algorithms SPEA [119] and NSGA [113]. These two algorithms represent the well working evolutionary multiobjective algorithms.

In Fig. 10.7 and 10.8 the history of evolution process for the benchmark Kn100 is depicted. The 1st, 25th, 50th and 100th generation of one run of Pareto optimal BOA is shown. In experiment I set the population size L=4000, but only 500 of the randomly chosen individuals/solutions are shown in the graph. The X-coordinate of each point equals to the function value $F_1(x)$ and the Y-coordinate equals to the function $F_2(x)$. This experiment shows the dynamics of the evolution process and the creation of Pareto front.



a) b)

Fig. 10.7 *Distribution of the solutions for Kn100 benchmark plotted in the a) 1st generation, b) 25th generation*



a) b)

Fig. 10.8 *Distribution of the solutions for Kn100 benchmark plotted in the a) 50th generation, b) 100 th generation.*

In Fig. 10.9 there is the comparison of the final Pareto front produced by Pareto BOA algorithm and by the SPEA [123] and NSGA algorithms [113] for the case of Kn100 benchmark and in Fig. 10.10 for the case of Kn250 benchmark. I performed 5 independent runs and constructed the final Pareto front from the 5 particular Pareto fronts. From Fig. 10.9 it is evident that for Kn100 the Pareto solutions produced by my Pareto BOA in the middle part of Pareto front are slightly better than the Pareto solutions produced by SPEA and NSGA. What is more important – my Pareto BOA produces more solutions in the Pareto front margins.

In Fig. 10.10 we see that for Kn250 the difference between Pareto fronts is more expressive – my Pareto BOA outperforms the SPEA and NSGA. I used the following setting for my algorithm: for the Kn100 I set the population size L to 2000, for the case of the Kn250 the population size L equals to 5000. The number of generations used is about 300. The computation time is about 15 minutes for the Kn100. In case of the Kn250 the time was 2 hours for L=5000 and about 40 minutes for L=2000. To reduce the computation time I consider the implementation of parallel version of the Pareto BOA algorithm. It would be also possible to shorten the computational time by decreasing the size of population, but I wanted to keep those predefined values used also in my previous experiments.

In the context of algorithm comparison an important question arises: What measure should be used to express the quality of the results so that the various evolutionary algorithms can be compared in a meaningful way. In [123] two measures are described. One of them denoted as *S* represents the size of the objective space covered, the second measure denoted *C* represents the coverage of two sets according to their dominance. I preferred in my comparison the topology/shape of the Pareto fronts.

All experiments were run on Sun Enterprise 450 machine (4 CPUs, 4 GB RAM), in future I consider the utilizing the cluster of Sun Ultra 5 workstations for parallel version of Pareto-BOA.

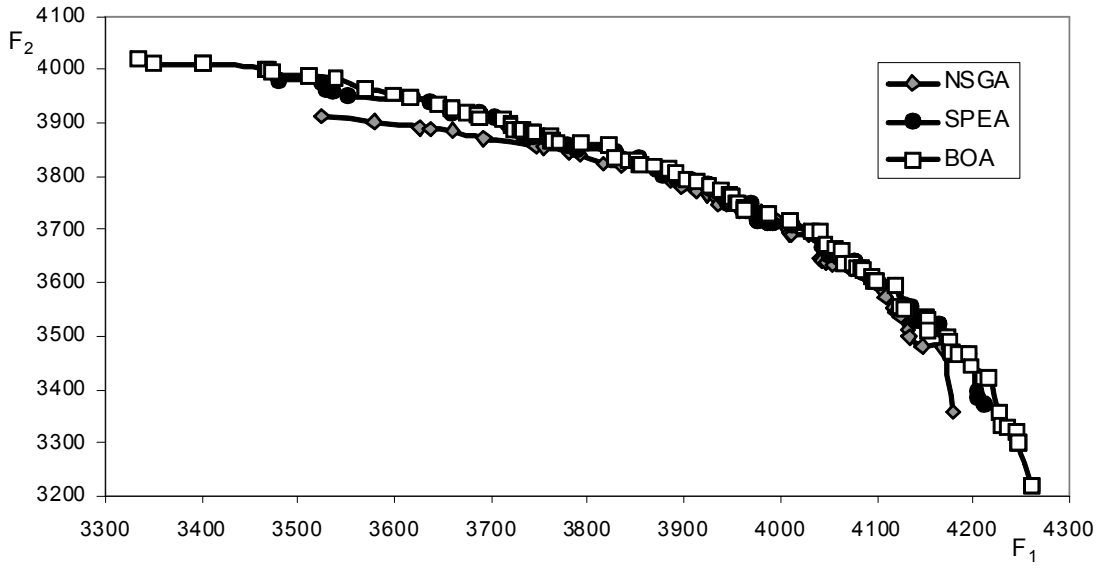


Fig. 10.9 *Final Pareto fronts for Kn100, population size L=2000*

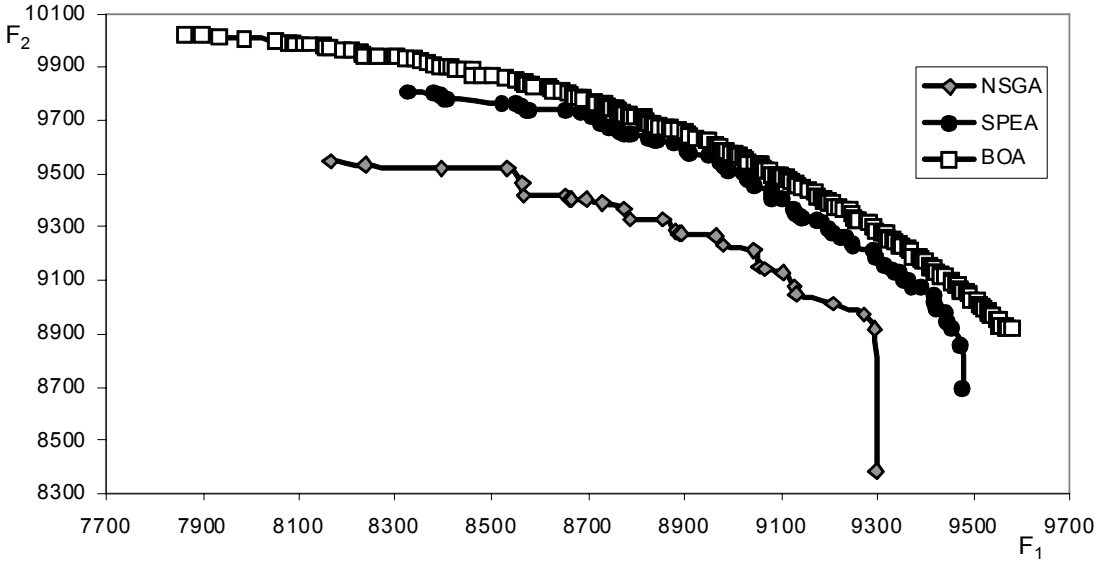


Fig. 10.10 *Final Pareto fronts for Kn250, population size L=5000*

10.2.5 Summary

I have modified the evaluation and replacement phase of the standard BOA algorithm by utilizing the concept of a Pareto-strength fitness published for SPEA algorithm in [123].

The Pareto BOA performance was compared to two well known evolutionary algorithms SPEA and NSGA on two multiple 0/1 knapsack problems Kn100 and Kn250. Let me note that the SPEA is modern multiobjective optimization algorithm which was able to surpass other existing algorithms on many test problems and was applied successfully various computer engineering design problems. That is why I compare the performance of my and SPEA algorithm. The Pareto solutions produced by my algorithm are uniformly distributed along the Pareto front which is broader than in the case of NSGA and SPEA algorithms. Many problems remain to be solved, namely the large computational complexity. To reduce the computational complexity I proposed the idea of the parallelization of Pareto BOA including the decomposition and detection of the Pareto front. This approach is an extension of distributed BOA proposed in chapter 9.4.

The next possible improvement lies also in more sophisticated niching technique, modification of replacement phase of the algorithm and introduction of problem knowledge into optimization process.

10.3 BMOA: Multiobjective BOA with epsilon-dominance

My earlier Pareto-strength BOA algorithm (see 10.2) using the original Pelikan's BOA core [91] with Bayesian network model was comparable to the best multiobjective-algorithms of its time - SPEA and NSGA. However, I decided to create new algorithm to be competitive to the recent SPEA2 [125] and NSGA-II [29] algorithms. The aim was also to develop completely new multiobjective technique for my perspective MBOA core based on mixed decision trees model. I started the joint research with Marco Laumanns from ETH Zűrich, one of the SPEA2 authors. Our new Bayesian Multi-objective Optimization Algorithm (BMOA) reflects all the new studies related to the theoretical convergence analysis [104],[72]. It has both properties of converging to the true Pareto-optimal front and maintaining a widely spread distribution of solutions. In our paper [i] we derived design guidelines for a successful selection scheme in EDAs. In connection with the construction of the probabilistic model we developed new selection operator based on ϵ-archives [72]. Chapters 10.3.1 - 10.3.5 are summary of our joint work on sequential BMOA. Chapter 10.3.6 describes a parallel extension of BMOA I proposed later.

10.3.1 Design guidelines

For the design of a multi-objective BOA the baseline algorithm we used is a generic convergent MOEA [72] with a guaranteed spread of solutions. Some important aspects have to be taken into account, some of which stemming from the peculiarities of multi-objective optimization problems, others from the necessity of the probabilistic model building techniques.

Preliminary tests with a simple ($\mu+\lambda$)-strategy and fitness assignment based on the dominance grade have shown that a trivial multi-objective extension leads to poor performance. The population is likely to converge to an ``easy to find'' region of the Pareto set and more and more duplicate solutions are produced. The resulting loss of diversity leads to an insufficient approximation of the Pareto set and is especially harmful for building a meaningful probabilistic model. Therefore the following design requirements are essential:

1. *Elitism (to preclude the problem of gradual worsening and enable convergence to the Pareto set),*

2. *Diversity maintenance in objective space (to enable a good approximation of the whole Pareto set), and*

3. *Diversity maintenance in decision space (to avoid redundancy and provide enough information to build a probabilistic model).*

10.3.2 A new selection operator

From the existing selection/archiving operators in evolutionary multi-objective optimization, the ϵ-archive [72] has been designed to meet the requirements 1 and 2 above. The aim is to approximate the set of Pareto-optimal solutions. The algorithm maintains a finite-sized archive of non-dominated solutions, which gets iteratively updated in the presence of every new solution. The non-domination check is performed with a new ϵ-dominance operator, which ensures that no two neighboring solutions (within a ϵ-neighborhood) are non-dominated to each other.
The use of ϵ-dominance also makes the algorithms practical by allowing a decision-maker to control the difference in objective function values (by choosing an appropriate ϵ) between two consecutive Pareto-optimal solutions.

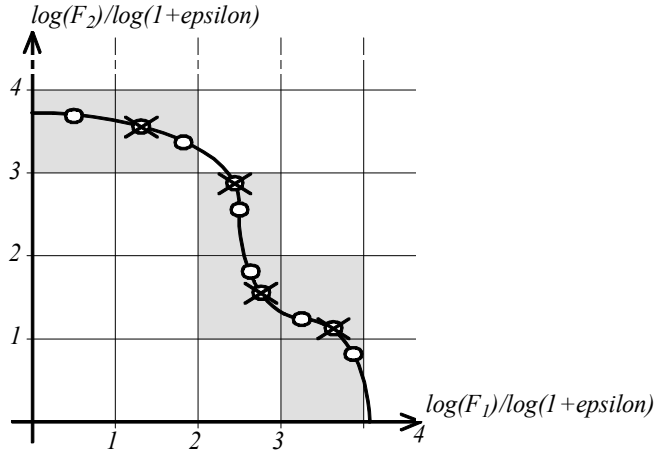


Fig. 10.11*: The basic idea of ϵ-dominance. The objective space is divided into ϵ-boxes. Each ϵ-box can contain only one solution. This avoids outbreak of "easy to get" solutions from the middle region.*

As the time goes to infinity it can be proven that for each objective coordinate the distance between "boxed" Pareto-front and the global Pareto-set will be less than $(1+\epsilon)$. For example by setting $\epsilon=10e^{-6}$ we reach 6-digit accuracy.

This method was proposed for use in classical MOEAs, because it maintains a minimal set of solutions that ϵ-dominate all other solutions generated so far. However, as this set can become very small, the scheme has to be modified to provide enough decision space diversity for BOA. The new selection operator is described in Alg. 1. The idea is that now also dominated individuals are allowed to survive, depending on the number of individuals that they are dominated by.

Algorithm 1 *Select*(*Ar*, *P*, μ, ϵ)
  Input: old parent set *Ar*, candidate set *P*,
    minimum size μ, approximation factor ϵ
  Output: new parent set *Ar'*
  for all $x \in P$ **do**
    $B := \{y \in Ar \mid \forall F_i: \lfloor \log F_i(y) / \log (1+\epsilon) \rfloor = \lfloor \log F_i(x) / \log (1+\epsilon) \rfloor \}$
    if $B = \emptyset$ **then**
      $Ar := Ar \cup \{x\}$
    else if $\exists y \in B$ such that $x \succ y$ **then**
      $Ar := Ar \setminus B \cup \{x\}$
    end if
  end for
  $Ar' := \{y \in Ar \mid \bar{\exists} z \in Ar: z \succ y\}$
  $D := Ar \setminus Ar'$
  if $|Ar'| < \mu$ **then**
    Fill *Ar'* with $\mu - |Ar'|$ individuals $y \in D$ in increasing order
      of $|\{z \in Ar' \cup D \mid z \succ y\}|$
  end if
  Return: *Ar'*

The algorithm consists of two parts – insertion of individuals from *P* to *Ar* and selection of most promising part of *Ar* into *Ar'*. The insertion rules for ε-archive are straightforward:

- a new solution is added if the ϵ-box is empty
- a new solution replaces the old solution in the ϵ-box if the new solution strictly dominates the old one
- otherwise the new solution is omitted

10.3.3 The $(\mu+\lambda, \epsilon)$ – BMOA

The combination of the selection operator (Alg. 1) and the variation based on the probabilistic model results in a Bayesian Multi-objective Optimization Algorithm described in Alg. 2. In this $(\mu + \lambda, \epsilon)$ - BMOA, μ denotes the (minimum) number of parents that survive to the next generation being the input to build the model, λ the number of samples from the model in one generation and ϵ the factor that determines the granularity of the approximation.

```
Algorithm 2 (μ+λ,ϵ) - BMOA
while  |Ar| < μ do
   Randomly create an individual x.
   Ar := Select(Ar, {x}, μ, ϵ)
end while
while Termination criteria not fulfilled do
   Create Bayesian Model M from Ar.
   Sample λ new individuals from M.
   Mutate these individuals and put them into B.
   Ar := Select(Ar,B,μ,ϵ)
end while
```

Each individual sampled from the model is additionally mutated by flipping each bit independently with probability $1/n$. From the theoretical point of view the mutation step is necessary to ensure the reachability of all solutions. It is not common in the EDA framework, but in conjunction with steady-state replacement and ϵ-archiving this guarantees (see [72]) that in the limit the non-dominated population members converge to a representative subset of the Pareto set in the sense that each Pareto-optimal point is ϵ-dominated by at least one population member.

10.3.4 Empirical study of BMOA

The behavior of the resulting Bayesian Multi-objective Optimization Algorithm (BMOA) was empirically investigated on the multi-objective knapsack problem with $m = 2$ objectives to demonstrate the applicability of the approach.

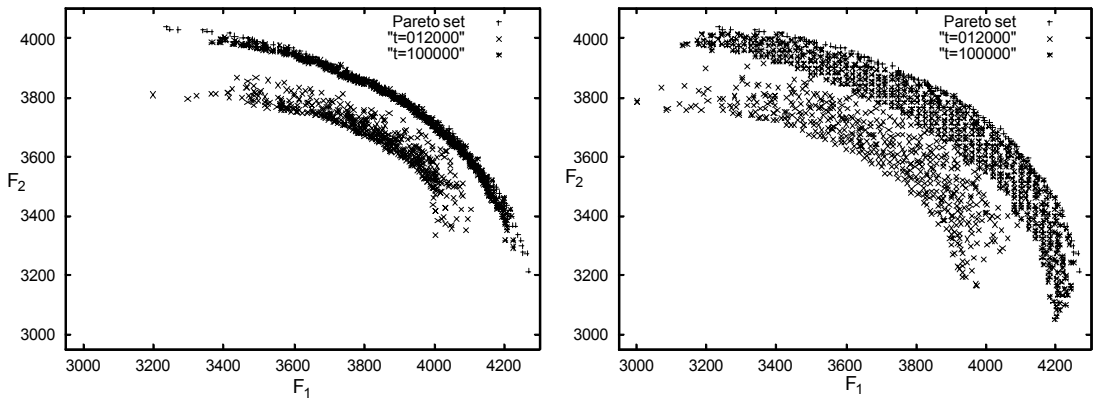


Fig. 10.12. *Progress of the population of (500+500, ϵ)-BMOA on the KP-100-2 for $\epsilon = 10^{-6}$ (left) and $\epsilon = 0.005$ (right) after t = 12000 and t = 100000 function evaluations.*

Fig. 10.12 shows the progress of the population for different ϵ values for a typical run. With a small ϵ, it can be seen that the population is more concentrated near the middle of the Pareto set compared to the larger ϵ value, where the population is distributed more evenly and broadly.

Fig. 10.13 displays the number of Pareto-optimal solutions found and the empirical approximation quality $\epsilon_{\min}$ over time.

$$\epsilon_{min} := \text{Min}\{\epsilon \in \mathrm{R}^{+} |\ \forall x \in \{0,1\}^{n}\ \exists\, y \in \text{Ar} \text{ with } (1+\epsilon)\, F(y) > F(x)\} \qquad (10.14)$$

It indicates how the algorithm can be tuned by different settings of the μ and ϵ values. For larger values of both parameters, more dominated individuals will be allowed in the population: Whereas a large ϵ value means that the individuals have to compete for less available "slots", a larger μ simply enlarges the storage. As discussed before, more individuals lead to a more accurate model estimation, but if the fraction of dominated individuals is large, a lot of sampling (and search) effort might be wasted on exploring previously visited regions and thereby increasing the running time.

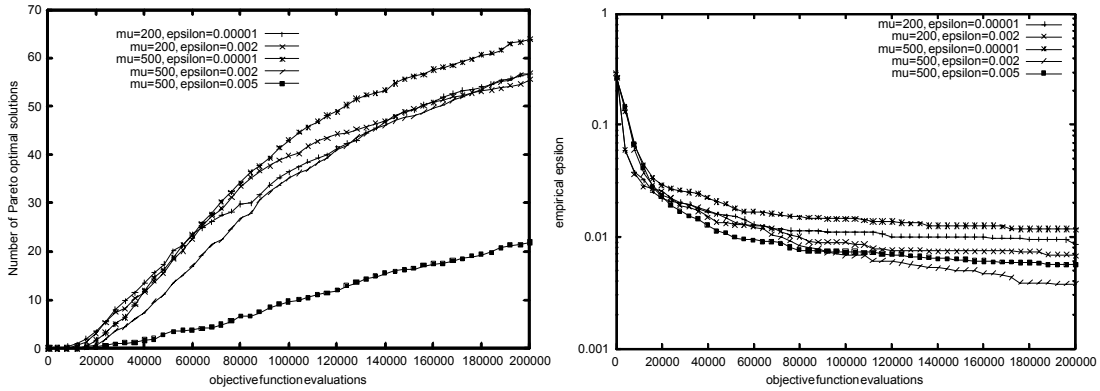


Fig. 10.13 *Average number of Pareto-optimal solution contained in the population (left) and empirical approximation quality ϵ_{min} (right) for KP-100-2 for different μ and ϵ values.*

10.3.5 Comparison with other MOEAs

In order to evaluate BMOA with respect to other MOEAs we use the results of the comparative case study from [125] and focus on the large instances of the knapsack problem with n=750.

Table 10.3 compares BMOA to different algorithms using the coverage metric $\mathcal{C}(\mathcal{A},\mathcal{B})$. It calculates the relative number of non-dominated individuals contained in the population of algorithm $\mathcal{B}$ that are dominated by at least one individual from the population of algorithm $\mathcal{A}$ of a given point in time. The (3000+3000, 10^{-6})-BMOA is able to dominate more that half of the other algorithms' populations on nearly all instances, with the best results on the four-objective problem. The other algorithms are not able to dominate any of BMOA's non-dominated points, but they generally find a broader distribution.

Tab. 10.3. *Results of the coverage measure $\mathcal{C}(\bullet,\bullet)$ to compare the (3000 + 3000,10^{-6})-BMOA with NSGA-II, PESA, SPEA, and SPEA2 after t = 480000 function evaluations, median of 30 runs.*

$\mathcal{C}$(BMOA, •), $\mathcal{C}$(•, BMOA)	KP-750-2	KP-750-3	KP-750-4
NSGA-II	0.71, 0.00	0.56, 0.00	0.72, 0.00
PESA	0.71, 0.00	0.64, 0.00	0.97, 0.00
SPEA	0.52, 0.00	0.63, 0.00	0.99, 0.00
SPEA2	0.58, 0.00	0.48, 0.00	0.80, 0.00

10.3.6 Parallel BMOA

Because of its relatively large population size, the BMOA proceeds much slower than other MOEAs and it requires more CPU time due to the estimation of the probabilistic model. To overcome this difficulty a parallel version BMOA can be designed on the basis of multithreaded BOA with decision trees (see 9.5). But this concept of parallelism needs some extension for efficient parallel manipulation with ε-archive.

I propose to replace the sequential *Select*(Ar, P, μ, ϵ) from Alg 1. by two parallel steps. The first step is the parallel insertion of P into Ar. It uses a hash function $h(\boldsymbol{x})$ which transforms m-dimensional coordinate of corresponding ϵ-box to integer number in the range $\{0,..,v\text{-}1\}$, where v is the number of processors.

$$h(\boldsymbol{x}) : A^n \rightarrow \{0,..,v\text{-}1\}, \text{ such that}$$

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in A^n : h(\boldsymbol{x}) = h(\boldsymbol{x}') \Leftrightarrow \forall i \in \{1,...,m\} : \left\lfloor \frac{\log(F_i(\boldsymbol{x}))}{\log(1+\epsilon)} \right\rfloor = \left\lfloor \frac{\log(F_i(\boldsymbol{x}'))}{\log(1+\epsilon)} \right\rfloor$$

The old parent set Ar, and the candidate set P is splitted into m groups $Ar_i = \{\boldsymbol{x} \mid \boldsymbol{x} \in Ar \wedge h(\boldsymbol{x})=i\}$, $P_i = \{\boldsymbol{x} \mid \boldsymbol{x} \in P \wedge h(\boldsymbol{x})=i\}$. Then in each processor i the insertion of individuals from P_i to Ar_i is performed independently. If the hash function is of good quality then each processor is responsible for almost the same number of ϵ-boxes. At the end of the first step the sets Ar_i are merged into one set Ar.

The second step is the parallel selection of individuals from Ar to Ar' according to increasing number of individuals who dominate them. For this task the methods from 10.2.3 can be adapted.

10.3.7 Summary

A Bayesian Multi-objective Optimization Algorithm ($\mu+\lambda,\epsilon$)-BMOA has been designed using a probabilistic model based on binary decision trees and a special selection scheme based on ϵ-archives. The convergence behavior of the algorithm can be tuned via the values of μ, the minimal population size to estimate the probabilistic model, and ϵ, the approximation factor.

The algorithm was empirically tested on different instances of the 0/1 multi-objective knapsack problem. The BMOA is able to model the distribution of the non-dominated points in the population well and to improve these. In order to find also the outer region of the Pareto set, large μ and ϵ values are required which slows down the optimization process considerably. To keep the optimization time acceptable I proposed the parallelization of BMOA.

Further research could assess BMOA on other multi-objective combinatorial optimization problems with stronger variable interactions (see [57]) and on continuous problems. From the decision-aid point of view it would be interesting to exploit the Bayesian model also outside the algorithm itself. The compact description of the model could assist a decision maker who can analyze the decision trees to get more insight into the structure of the Pareto set and to learn about correlations in the decision problem at hand.

11 Prototyping EDA applications

This part is an extension of Radim Kostka's master thesis [61] I supervised. We deal with the design of the modular development system DEBOA for the rapid prototyping of optimization applications on the basis of BOA algorithm with binary decision graphs. The general requirements on the development system are identified including modularity and visualization of results. The design of the development system itself is realized on the platform of Java language, which ensures great portability. DEBOA system can be easily extended to support new types of fitness functions as well as new types of visualizations.

11.1 Motivation

The creation of a well-performing classical evolutionary algorithms highly depends on problem encoding, genetic operators and control parameters. In the field of classical EAs there are many development environments for fast prototyping, but in the field of EDAs these tools are missing. We investigated the Pelikan's BOA code with binary decision trees and designed the DEBOA development system for fast prototyping of optimization tasks. This approach is also usable for MBOA algorithm with mixed decision trees.

In the next section an overview of several typical development systems based on classical evolutionary algorithms is discussed. In chapter 11.3 the common features of existing systems are summarized and the most important requirements for modern development system are specified. In chapters 11.4 and 11.5 the design and implementation details of the DEBOA development system are present and the most important parts from the user guide are included.

11.2 Particular EA tools

A number of EA tools have been investigated to identify the most important features of development systems.

11.2.1 G.O.D.

The Genetic Object Designer (GOD) [40] is a Windows based genetic algorithm shell using EOS (Evolutionary Object System) library. It facilitates the exploration of many variants of the traditional GA when optimizing a specific problem. GOD comes with a built-in interpreter of EPL language (Evolutionary Programming Language) for purposes of specifying the domain specific parts of genetic algorithm. The final GA can also be exported to application written in C++ using the "Generate C++ Code" option.

11.2.2 GADESIGN

The GADesign toolkit was created at Brno University of Technology by Libor Kriz [62] as a part of his master thesis. Its modular concept is based on the platform of object-oriented Pascal language in the frame of the Borland Delphi system. This development tool allows for design of proper genetic algorithm without detailed knowledge of genetic theory using only an intuitive process of specification of the genetic classes, functors and parameters.

The user code can be loaded from DLL or interpreted using built-in interpreter of Pascal-like language. The creation of stand alone application is possible by using the DLL library containing GADesign evolutionary engine.

11.2.3 GALIB

Galib [119] a source code library of genetic algorithm objects. It allows for using genetic algorithms in any C++ program. The library includes basic types of genetic algorithms, chromozome encoding and genetic operators.

11.3 Demands on development system

The most advantageous features of existing development systems can be generalized as follows.

11.3.1 Visualization

By the visualization I mean the ability of EA development system to provide as many information about the evolution as possible. The common visualization can be divided into two classes:

- visualization of EA evolution process
- visualization of current population distribution in solution space

11.3.2 Extensibility

Modern EA toolkits should be implemented in a modular way, such that a new operator, visualization or fitness function can be added. After building a final source code for solved optimization task two extensibility modes can be used:

- Interpreted code
- Binary code using DLL files

The interpreted mode uses a built-in interpreter of pseudo-programming language in which the user code can be written. The binary one is faster, because the precompiled binary code is dynamically loaded and executed (for example from .DLL file or Java .class file).

11.3.3 Reusability

Most of EA toolkits support only prototyping, but some of them can also export settings and operators into final application such that the optimization engine needs not to be programmed again. For example the core of reusable EA toolkit is contained in .DLL file and its API functions are called from the source code generated according to optimal settings found.

11.3.4 Portability

By the portability I mean the transfer of EA toolkit from one platform to another. Usually the commercial toolkits contain platform-specific binaries whereas the open-source toolkits are easily portable.

11.4 Design and implementation of DEBOA system

11.4.1 Java language

We adopted the original BOA package in C++ [96] and proposed new object oriented modular design based on Java language. It fulfills the following properties:

- redesign of BOA core
- reusability of BOA core for variety of applications
- visualization of BOA evolution process
- interactive parameter settings

Java contains several features that argue for it. It is widely distributed and has become one of the major programming languages. The development kit, including compiler and debugger, is freely available on a number of different computer platforms. The core libraries contain many functions which can be used directly and need not be adopted from external libraries, which is not the case in C++ for instance. Another important point for future work is the built-in support for multithreading and distributed computing.

Java is a strictly object-oriented language. By combining the object-oriented design with the other Java features, my project has several unique advantages:

- in the Java applet mode it is suitable for WWW presentations
- pre-compiled modules with new visualizations and new fitness computation can be dynamically loaded during run-time as new classes
- the class files are easily portable, no need to distribute the source code
- the class containing optimization core is reusable, it can be easily incorporated into the final project

11.4.2 Modular design

For object oriented design the "design patterns" technique was used. The simplified scheme of DEBOA system is shown in the following UML diagram.

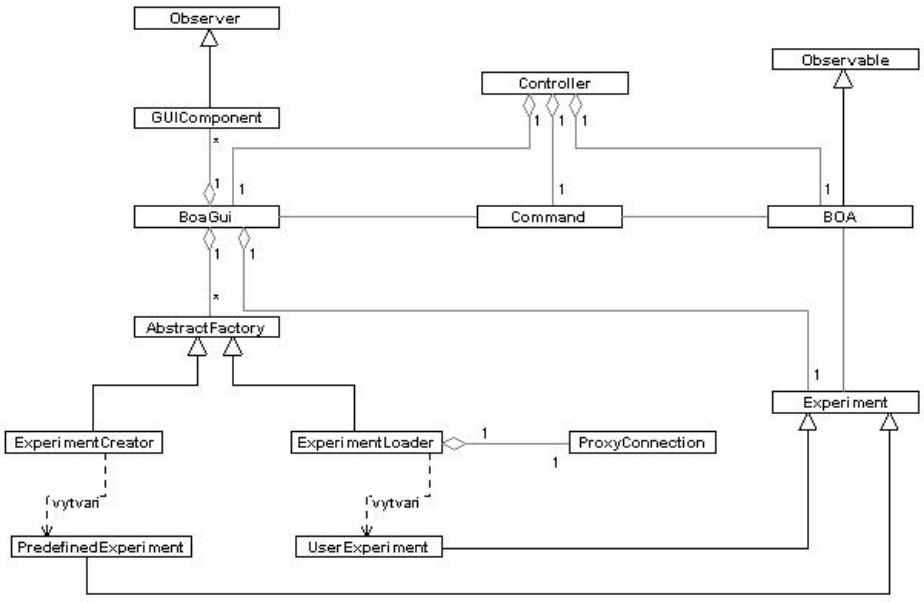


Fig. 11.1*: The UML diagram of DEBOA classes*

11.4.3 Visualization of characteristics

The following common visualizations were implemented:

- Statistics – basic statistics of optimization process
- Fitness – graph of minimal, average and maximal fitness function values
- Fitness Hist – fitness distribution histogram
- Similarity – minimal, average and maximal Hamming distance between individual genotype and global optimum genotype if known
- Similarity Hist – similarity distribution histogram

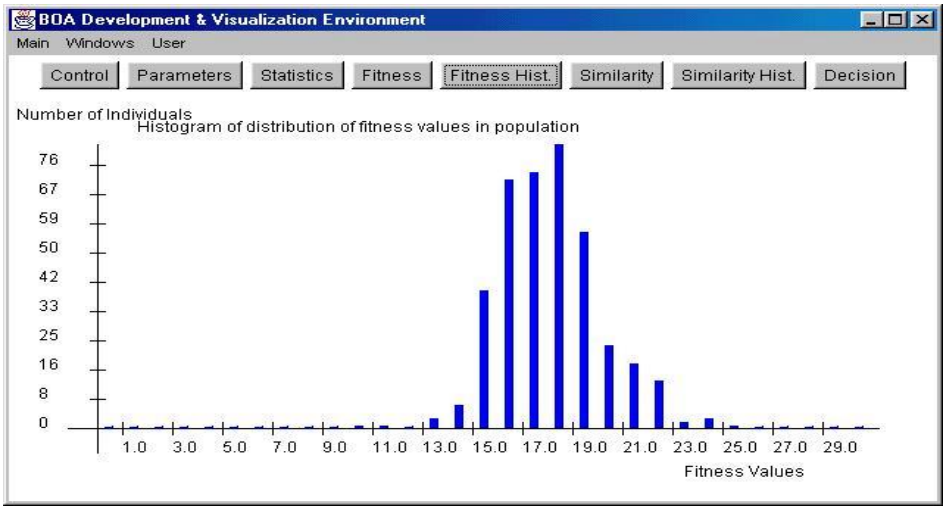
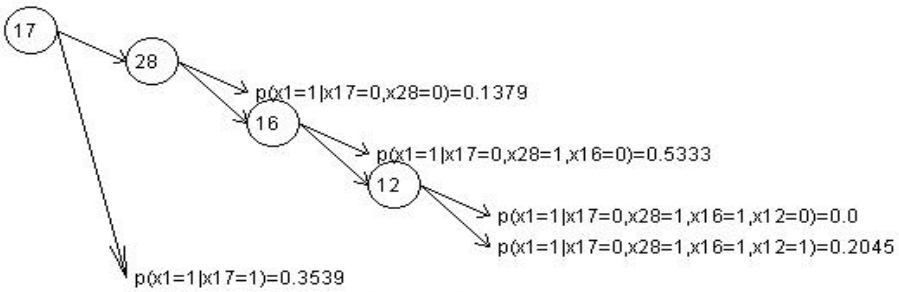


Fig. 11.2: Fitness distribution histogram

Moreover, the visualization of BOA probabilistic model was included:

- Decision – decision trees for each variable/gene of the chromosome



x12	x16	x17	x28	P
*	*	0	0	0.13793103
*	*	1	*	0.3539823
*	0	0	1	0.53333336
0	1	0	1	0.0
1	1	0	1	0.20454545

Fig. 11.3: Binary decision diagram from DEBOA

11.4.4 Tool for rapid prototyping

Due to modular architecture the system can be easily extended. New experiments are inherited from the class *develop.user.Experiment*. Both types of extensibility are supported. I created the class *ExperimentCreator* for loading of built-in experiments and class *ExperimentLoader* for dynamical loading of experiments during run-time. Thanks to the class *ProxyConnection* the dynamical loading is transparent and can be done from local filesystem as well as from URL.

To create new experiments the users can use these steps :

1) Create new file *NewExperiment.java*
2) Import interface *develop.boa* using the directive *import*, if methods working with populations or other data types defined in this package should be redefined
3) Create new class by extending already existing class *develop.user.Experiment* using keyword *extends*
4) Declare new variables and data types for new experiment
5) Define the method *Init()* that sets up the parameters of the experiment (name, menu items names, names of user graphs, etc.) and the flag *isBestDefined* if the best individual is defined
6) If required, redefine methods *FitnessFunction(), isOptimal(), howSimilarToOptimal(), Init(), updateStatistics(), alterPopulation(), LoadData(), SaveData(), DrawUserGraph1(), DrawUserGraph2()*
7) Compile new experiment using command: *javac NewExperiment.java*

Each experiment class provides support for common visualizations. For example the method *howSimilarToOptimal()* computes the Hamming distance between individual and global optimum, which is needed for Similarity graph. Moreover, the base class *develop.user.Experiment* declares methods *DrawUserGraph1(java.awt.Graphics g)* and *DrawUserGraph2(java.awt.Graphics g)*. The addition of application-specific visualizations can be implemented by redefining them.

11.4.5 Built-in benchmarks

The following implicit benchmarks were implemented:

- Graph partitioning (see 7.1)
- Knapsack problem (see 7.3)
- Optimization of artificial functions (see 7.4) - OneMax function, 3-deceptive function, etc.

11.5 User guide

11.5.1 DEBOA versions

The development system DEBOA can be used in two versions :

- Java Applet – this version is designed for the presentation of the system in WWW browsers, which include built-in JVM (Java Virtual Machine) interpreter.
- Java Aplication – the full version which allows for designing new applications.

To compile and run DEBOA it is necessary to use JDK 1.1.8 or higher (Java Development Kit), that includes Java interpreter, compiler and AppletViewer for running applets, and operating system with GUI, because the system runs only in graphical mode.

In case of Java applet the www browser is usable. But it is important to set access rights for unsigned Java Applets in the browser:

- read / write access to user files
- dialogs opening

If a browser is not capable of setting access rights for Java, it is still possible to run system with following constraints :

- System is not allowed to generate output files of BOA nor the output files defined by the user parameters „*Evolution history file*“ and „*Data file for experiment to save*“. Thus, in window „*Parameters*“ these settings must be set blank.
- System is also not allowed to read user defined classes with experiments nor user data files, thus also parameters „*Data file for experiment to load*“ and „*Experiment class filename*“ in the same window must be set blank. Only predefined experiments are accessible.

11.5.2 Running the system

After compiling and launching the program (applet or application version) the main window opens. There are two equivalent ways to switch between windows of the program :

- user menu
- button panel located below the menu bar

The “*Main*” window contains controls button for loading experiments and for loading or saving user data. The user can run, step and stop the optimization process thanks to control panel of the application that is accessible in the window „*Control*“. The menu „*Windows*“ gives the user access to all windows displaying the visualized characteristics. The menu „*User*“ gives the user access to user defined graphs. The user can define two user output graphs altogether with their names that will appear in the menu „*User*“. These items can differ from experiment to experiment, see section 11.4.4.

11.5.3 Loading user defined experiments

Let us suppose that you have created the new experiment called *MyExperiment*. After its compilation you get file *MyExperiment.class*. Furthermore let us suppose that this experiment will load the data from file *C:\my.dat* and write the data to file *C:\my.out*.

1) Launch DEBOA
2) Switch to window „*Parameters*”
3) Set the parameter „*Data file for experiment to load :*“ to „*C:\my.dat*“
4) Set the parameter „*Data file for experiment to load :*“ to „*C:\my.out*“
5) Set the parameter „*Enter Experiment class filename or press button to locate it*“ to „*$path\MyExperiment.class*“, where *$path* is the path to file *MyExperiment.class*. In case that we want to run the experiment placed on the WWW we can do that by specifying the URL of the file instead of path, for example:
   „*http://www.experiments.com/MyExperiment.class*“
6) Switch to window „*Control*“ and run the experiment

11.5.4 Description of output files

Our DEBOA system produces the output files:

- **.fitness* - the history of min./max./avg. fitness
- **.log* – more detailed history, including best individuals
- **.model* – the history of model evolution, a set of decision graphs for each generation

Note: * is the name of output file specified in „*Parameters*“ window

11.6 Future work

DEBOA is based on Pelikan's BOA code with binary decision graphs model, but can be easily modified for MBOA code with mixed decision trees. The future work lies in the implementation of parallel BDTs construction using multithreaded features of Java language. This multithreaded parallel version of MBOA was proposed and simulated in chapter 9.5. It will speed up the execution time linearly with increasing number of processors used. Reusing of such a toolkit in the final project will result in creation of parallel application.

12 Conclusion and future work

The main contributions and conclusions of this thesis can be summarized as follows:

- A new paradigm for Evolutionary Computation was explored and the new formal specification of EDA algorithm was formulated.

- It was developed a new probabilistic graphical model composed of binary decision trees with mixed decision nodes, which is more accurate and reliable model than the pure Bayesian network, more general model than Gaussian network and more useful for building blocks evolution than mixtures of Gaussians.

- A new advanced EDA algorithm MBOA (Mixed Bayesian Optimization Algorithm) for solving problems with real-valued parameters was designed and implemented. Its main advantage against the mostly used IDEA and EGNA approach is the backward compatibility with discrete domains, so it is uniquely capable of learning linkage between mixed continuous-discrete genes. MBOA handles the discretization of continuous parameters as an integral part of the learning process and the divide-and-conquer approach used for construction of decision trees exhibits high performance. For experiments with MBOA a new mixed continuous-discrete benchmark F_{mixed} was established, which is very hardly sensitive to correctness of decision tree building algorithm. MBOA was able to find global optimum, so the dependencies within each pair of deceptive mixed parameters have been successfully discovered.

- Original concept of parallelization of BN construction was proposed on the basis of predefined ordering of BN nodes. Parents for each node can be found separately without checking for acyclic dependency graph. This completely removes the need for communication between parallel processes and enables nearly linear speedup. It should be emphasized that the usage of concepts of parallel construction of probabilistic model is not limited only to the area of Estimation of Distribution Algorithm. The graphical probabilistic model became an important tool in the artificial intelligence, expert systems, data mining, etc.

- Three different versions of parallel EDA algorithms were proposed. The pipelined Parallel BOA algorithm (PBOA) was proposed for fine-grained type of parallelism with tightly connected communication channels. Simulation results showed that the quality of generated network was almost the same as in the sequential case. The Distributed Bayesian Optimization Algorithm (DBOA) was proposed and implemented for coarse-grained type of parallelism using the message passing communication (MPI). The experiments were successfully run on the uniform cluster of Sun Ultra 5 workstations with UltraSPARC II processors connected by fast Ethernet multi-port router. In the multithreaded MBOA algorithm an efficient parallel construction and sampling of binary decision trees was proposed and successfully validated via simulation in Transim tool.

- Additional methods for utilization of prior knowledge were suggested to speed up the convergence. In a Problem Knowledge-based BOA algorithm (KBOA) the partial (local) information about the problem was used to adjust the prior of probabilistic model and injection of building blocks was used to improve the quality of initial population.

- Multi-criterial BOA algorithms were designed. The first one was the Pareto BOA algorithm based on promising Pareto-strength fitness technique. It was comparable to the best recombination-based multi-objective algorithms of its time - NSGA and SPEA. More recent is the Bayesian Multi-objective Optimization Algorithm (μ+λ,ϵ)-BMOA, which has been designed using a probabilistic model based on binary decision trees and a special selection scheme based on epsilon-archives. The convergence behavior of the algorithm can be tuned via the values of μ, the minimal population size to estimate the probabilistic model, and ϵ, the approximation factor. The algorithm was empirically tested on different instances of the 0/1 multi-objective knapsack problem and successfully compared to NSGA-II and SPEA2 algorithms.

- Visual environment DEBOA for rapid prototyping of BOA optimization applications was developed. It includes all the important features of evolutionary toolkits – visualization, extensibility, reusability and portability. Due to using of Java language this system fulfills all the requirements for modern development system. One of its unique properties is the ability to be executed as java applet for WWW presentations. As far as I know, DEBOA is the only development environment based on Bayesian optimization algorithm for efficient solution of complex optimization problems with high epistasis.

The future work will be oriented towards the application of proposed parallel construction of dependence graph and usage of proposed mixed decision tree model and its metrics in non-evolutionary tasks like artificial intelligence, expert systems and data mining.

References

[1] Altenberg, L.: NK fitness landscapes. In Handbook of Evolutionary Computation, Oxford University Press, Oxford, 1997, page B2.7.2.

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

[3] Baluja, S.: Population-Based Incremental Learning: A method for Integrating Genetic Search Based Function Optimization and Competitive Learning, Technical report CMU-CS-94-163, Carnegie Mellon University, 1994.

[4] Baluja, S.: An empirical comparison of seven iterative and evolutionary function optimizaion heuristics. Technical report CMU-CS-95-193, Carnegie Mellon University, 1995.

[5] Baluja, S., Caruana, R.: Removing the genetics from standard genetic algorithm. Proceedings of the International conference on Machine Learning, pp. 38-46, Morgan Kaufmann, 1995.

[6] Baluja, S.: Genetic algorithms and explicit search statistics. Advances in Neural Information Processing Systems, volume 9, pp. 319-325, 1997.

[7] Baluja, S., Davies, S.: Combining multiple optimization runs with optimal dependency trees. Technical report CMU-CS-97-157, Carnegie Mellon University, 1997.

[8] Baluja, S., Davies, S.: Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Proceedings of the 14th International Conference on Machine Learning, pp. 30-38, Morgan Kaufmann, 1997.

[9] Baluja, S., Davies, S.: Fast probabilistic modeling for combinatorial optimization. In AAAI-98, 1998.

[10] Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A.: Solving graph matching with EDAs using a permutation-based representation. In Larrañaga, P. and Lozano, J. A., editors, Estimation of Distribution Algorithms. A new Tool for Evolutionary Computation. Kluwer Academic Publishers, 2002.

[11] Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., Boeres, C.: Inexact graph matching using learning and simulation of Bayesian networks. An empirical comparison between different approaches with synthetic data. In Workshop Notes of CaNew2000: Workshop on Bayesian and Causal networks: From Inference to Data Mining, Fourteenth European Conference on Artificial Intelligencem ECAI2000, Berlin, 2000.

[12] De Bonet, J. S., Isbell, C. L., Viola., P.: MIMIC: Finding optima by estimating probability densities. Advances in Neural Information Processing Systems, volume 9, pp. 424. The MIT Press, Cambridge, 1997.

[13] Bosman, P. A. N., Thierens., D.: An Algorithmic Framework For Density Estimation Based Evolutionary Algorithms, Technical report UU-CS-1999-46, Utrecht University, 1999.

[14] Bosman, P. A. N., Thierens., D.: Linkage information processing in distribution estimation algorithms. Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, volume I, pp. 60-67, Orlando, Florida, 1999. Morgan Kaufmann Publishers, San Fransisco, CA.

[15] Bosman, P. A. N., Thierens., D.: Continuous iterated density estimation evolutionary algorithms within the IDEA framework, Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM workshop, Genetic and Evolutionary Computation Conference GECCO-2000, pp. 197-200. 2000.

[16] Bosman, P. A. N., Thierens., D.: Expanding from discrete to continuous estimation of distribution algorithms: The IDEA, Parallel Problem Solving From Nature - PPSN VI, pp. 767-776. Springer-Verlag, 2000.

[17] Bosman, P. A. N., Thierens., D.: IDEAs based on the normal kernels probability density function, Technical report UU-CS-2000-11, Utrecht University, 2000.

[18] Bosman, P. A. N., Thierens., D.: Negative log-likelihood and statistical hypothesis testing as the basis of model selection in IDEAs, Proceedings of the BENELEARN-2000 Tenth Belgium-Netherlands Conference on Machine Learning, pp. 109-116. 2000.

[19] Bosman, P. A. N., Thierens., D.: Mixed IDEAs. Utrecht University technical report UU-CS-2000-45, pp 1-71, Utrecht, 2000.

[20] Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees, Wadsworth, 1984.

[21] Buntine, W.: Theory refinement on Bayesian networks. Proceedings of Seventh Conference on Uncertainty in Artificial Intelligence, Los Angeles, CA, pp. 52-60. Morgan Kaufmann, 1991.

[22] Buntine, W.: A Guide to the Literature on Learning Probabilistic Networks from Data, IEEE Transactions on Knowledge and Data Engineering 8(2), 1996, pp. 195-210.

[23] Chickering, D. M., Geiger, D., Heckerman, D.: Learning Bayesian networks is NP-hard, Technical Report MSR-TR-94-17, Microsoft Research, Redmond, Wirginia, 1994

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

[25] Chipman, H., George, E.I., McCulloch, R.E.: Bayesian CART model search, Technical Report, Department of MSIS, University of Texas, Austin, 1997, pp. 1-15

[26] Coello, C. A.: An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design. PhD thesis, Department of Computer Science, Tulane University, New Orleans, LA, April 1996.

[27] Cooper, G. F., Herskovits, E. H.: A Bayesian method for the induction of probabilistic networks from the data, Machine Learning, Vol. 9, pp. 309-347, 1992.

[28] Corne, D., Knowles, J., Gates, M.: The Pareto envelope-based selection algorithm for multiobjective optimization. In Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature VI (PPSN-VI), pp. 839-848, Springer-Verlag, 2000.

[29] Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In Proceedings of the Parallel Problem Solving from Nature – PPSN VI, pp. 849-858, Springer-Verlag, 2000.

[30] Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. Foundations of Genetic Algorithms 2, pp. 93-108, 1993.

[31] DeGroot, M.: Optimal Statistical Decisions, McGraw-Hill, New York.

[32] Etxeberria, R., Larrañaga, P.: Global optimization using Bayesian networks. Second Symposium on Artificial Intelligence(CIMAF-99), pages 332-339, Habana, Cuba, March 1999.

[33] Fonseca, C. M., Fleming, P. J.: Genetic algorithms for mul-tiobjective optimization: Formulation, discussion, and generalization. In Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 416-423, 1993.

[34] Friedman, N., Goldszmidt, M.: Learning Bayesian Networks with Local Structure, In Jordan ed. Learning and Inference in Graphical Models, 1998.

[35] Friedman, N., Goldszmidt, M.: Sequential update of Bayesian network structure, In Proc. Thirteenth Conf. on Uncertainty in Artificial Intelligence (UAI), 1997.

[36] Friedman, N., Goldszmidt, M: Discretizing Continuous Attributes While Learning Bayesian Networks, In Proc. 13'th International Conference on Machine Learning (ICML), pp. 157-165, 1996.

[37] Gallagher, M.R., Frean, M., Downs, T.: Real-valued evolutionary optimization using a flexible probability density estimator. Proceedings of Genetic and Evolutionary Computation Conference, pp. 840-846, Morgan Kaufmann, 1999.

[38] Geiger, D., Heckerman, D.: Learning Gausian Networks, Technical Report MST-TR-94-10, Microsoft Advanced Technology Division, Microsoft Corporation, Seattle, Washington, 1994.

[39] Geman, S., Geman, D.: Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images. IEEE trans. on Pattern Analysis and Machine Intelligence 6, pp. 721-741, 1984.

[40] Genetic Object Designer – Documentation, Man Machine Interfaces, 1994.

[41] Goldberg, D. E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addision-Wesley Publishing Company, 1989.

[42] Goldberg, D.E., Korb, B., Deb, K.: Messy genetic algorithms: Motivation, analysis and first results, Complex systems, 3 (5), pp. 493-530, 1989.

[43] Gottvald, A.: Bayesian Evolutionary optimization. Proceedings of the Mendel'99 Conference, Brno University of Technology, Faculty of Mechanical Engineering, Brno, 1999, pp. 30-35.

[44] Harik, G., Goldberg, D. E.: Learning linkage, IlliGAL Report No. 96006, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 14 pp., 1996.

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

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

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

[48] Henrion, M.: Propagating uncertainty in Bayesian networks by probabilistic logic sampling, Uncertainty in Artifical Intelligence 2, North-Holland, Amsterdam, pp. 149-163, 1988.

[49] Höhfeld, M., Rudolph, G.: Towards a theory of population-based icremental learning. Proceedings of th 4th International Conference on Evolutionary Computation, pp. 1-5, IEEE Press, 1997.

[50] Holland, J.: Adaptation in Natural and Artificial Systems, Ann Arbor: University of Michigan Press, 1975.

[51] Holst A.: The use of a Bayesian Neural Network Model for Classification Tasks, PhD thesis, Royal Institute of Technology, Stockholm, Sweden, September 1997, pp 1-123.

[52] Horn, J., Nafpliotis, N., Goldberg, D.E.: *A Niched Pareto Genetic Algorithm for Multiobjective Optimization,* Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, volume 1, pp 82-87, Piscataway, New Jersey, June 1994. IEEE Service Center.

[53] Jansen, T.: On Classifications of Fitness Functions, In: Theoretical Aspects of Evolutionary Computing, pp. 371-385, Springer, 2001.

[54] Jaynes, E. T.: Bayesian Methods: General Background (174Kb), In Maximum-Entropy and Bayesian Methods in Applied Statistics, J. H. Justice (ed.), Cambridge Univ. Press, Cambridge, 1986.

[55] Kargupta, H: Revisiting the GEMGA: Scalable evolutionary optimization through linkage learning. In: Proceedings of 1998 IEEE International Conference on Evolutionary Computation, pp 603-608, IEEE Press, 1998.

[56] Karypis, G., Kumar, V.: Hmetis - A Hypergraph Partitioning Package, version 1.5.3, University of Minnesota, Department of Computer Science & Engineering, Army HPC Research Center, Minneapolis, http://www-users.cs.umn.edu/~karypis/metis/hmetis/download.shtml.

[57] Khan, N., Goldberg, D.E., Pelikan, M.: Multi-Objective Bayesian Optimization Algorithm, Illigal Report No. 2002009, March 2002, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 2002.

[58] Knowles, J. D., Corne, D. W.: *M-PAES: A memetic algorithm for multiobjective optimization.* Congress on Evolutionary Computation (CEC 2000)}, volume 1, pp. 325--332, Piscataway, NJ, 2000. IEEE Press.

[59] Knowles, J. D., Corne, D. W.: Approximating the non-dominated front using the Pareto archived evolution strategy. Evolutionary Computation Journal, 8(2):149-172, 2000.

[60] Knowles, J. D., Corne, D. W., Oates, M. J.: *The Pareto-Envelope based Selection Algorithm for Multiobjective Optimization,* In Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature - PPSN VI, pp. 839-848, Springer-Verlag, 2000.

[61] Kostka L.: The development system for the class of Bayesian Optimization Algorithm, Faculty of Informaton Technology, VUT Brno, Brno, 2001.

[62] Kříž L.: Development Tools for a Fast Genetic Algorithms Design, master thesis, Faculty of Informaton Technology, VUT Brno, Brno, 2000.

[63] Kvasnicka, V., Pelikan, M., Pospichal, J.: Hill climbing with learning (An abstraction of genetic algorithm), Neural Network World, volume 6: pp.773-796, 1996.

[64] Kvasnicka, V., Pospichal, J., Tiňo, P.: Evolučné algoritmy, STU Bratislava, 2000.

[65] Larrañaga, P., Lozano, J. A.: Estimation of Distribution Algorithms. A new Tool for Evolutionary Computation. Kluwer Academic Publishers, pp. 57-100, 2002.

[66] Larrañaga, P., Etxeberria, R., Lozano, J. A., Sierra, B., Inza, L., Peňa, J. M.: A review of the cooperation between evolutionary computation and probabilistic graphical models, Second Symposium on Artifical Intelligence, Adaptive Systems, CIMAF 99, Habana, Cuba, pp. 314-324, 1999.

[67] Larrañaga, P., Etxeberria, R., Lozano, J. A., Peňa, J. M.: Optimization by learning and simulation of Bayesian and Gaussian networks. Technical Report EHU-KZAA-IK-4/99, University of the Basque Country, December 1999.

[68] Larrañaga, P., Etxeberria, R., Lozano, J. A., Peňa, J. M.: Combinatorial optimization by learning and simulation of Bayesian networks. Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, pp. 343-352, Stanford, 2000.

[69] Larrañaga, P., Etxeberria, R., Lozano, J. A., Peňa, J. M.: Optimization in continuous domains by learning and simulation of Gaussian networks, Proceedings of the Genetic and Evolutionary Computation Conference 2000, Las Vegas, Nevada, July 2000.

[70] Larrañaga, P., Lozano, J. A., Bengoetxea, E.: Estimation of Distribution Algorithms based on multivariate normal and Gaussian networks. Technical report KZZA-IK-1-01, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 2001.

[71] Laumanns, M., Rudolph, G., Schwefel, H.P.: Mutation control and convergence in evolutionary multi-objective optimization. 7th International Conference on Soft Computing MENDEL'2001, pp. 24-29, Brno, Czech Republic, June 2001.

[72] Laumanns, M., Thiele, L., Deb, K., Zitzler, E.: On the convergence and diversity-preservation properties of multi-objective evolutionary algorithms. Technical Report 108, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, 2001.

[73] Lauritzen, S. L.: Graphical Models. Oxford University Press. 1996.

[74] Mahning, T., Mühlenbein, H.: Mathematical analysis of optimization methods using search distributions. Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM workshop, Genetic and Evolutionary Computation Conference GECCO-2000, pp. 205-208. 2000.

[75] Monmarché, N., Ramat, E., Desbarats, L., Venturini, G.: Probabilistic search with genetic algorithms and ant colonies. Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM workshop, Genetic and Evolutionary Computation Conference GECCO-2000, pp. 209-211. 2000.

[76] Mühlenbein, H.: The equation for response to selection and its use for prediction. Evolutionary Computation 5(3), pp. 303-346, 1997.

[77] Mühlenbein, H., Mahnig, T.: Convergence theory and applications of the factorized distribution algorithm, Journal of Computing and Information Technology 7(1), pp. 19-32, 1998.

[78] Mühlenbein, H., Mahnig, T.: The Factorized Distributon Algorithm for additively decomposable functions. Second Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99, pp. 301-313, La Habana, 1999.

[79] Mühlenbein, H., Mahnig, T.: FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. Evolutionary Computation, 7(4), pp. 353-376, 1999.

[80] Mühlenbein, H., Mahnig, T.: Evolutionary algorithms: From recombination to search distributions. Theoretical Aspects of Evolutionary Computating. Natural Computing, pp. 137-176, 2000.

[81] Mühlenbein, H., Mahnig, T., Ochoa, A. R.: Schemata, distributions and graphical models in evolutionary optimization, Journal of Heuristics, volume 5, pp.215-247, 1999.

[82] Mühlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions I. Binary parameters, Parallel Problem Solving from Nature - PPSN IV, pp. 178-187, Berlin, 1996. Springer Verlag.

[83] Mühlenbein, H., Voigt, H. M.: Gene pool recombination in genetic algorithms. Metaheuristics: Theory and Applications, pp. 53-62, 1996.

[84] Munetomo, M., Goldberg, D.E. Identifying linkage by nonlinearity check, IlliGAL Report No. 98012, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 1998.

[85] Ochoa, A. R., Mühlenbein, H., Soto, M.: Factorized Distribution Algorithm using Bayesian networks of bounded complexity, Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM workshop, Genetic and Evolutionary Computation Conference GECCO-2000, pp. 212-215. 2000.

[86] Ochoa, A. R., Mühlenbein, H., Soto, M.: A Factorized Distribution Algorithm using single connected Bayesian networks, Parallel Problem Solving from Nature – PPSN VI. pp. 787-796, Springer-Verlag, 2000.

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

[88] Pelikan, M., Goldberg, D. E., Cantú-Paz, E.: BOA: The Bayesian optimization algorithm. Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, volume I, pages 525-532, Orlando, FL, 1999. Morgan Kaufmann Publishers, San Fransisco, CA.

[89] Pelikan, M., Mühlenbein, H.: The bivariate marginal distribution algorithm. Advances in Soft Computing - Engineering Design and Manufacturing, pp. 521-535, London, 1999. Springer-Verlag.

[90] Pelikan, M., Muehlenbein, H.: Marginal Distributions in Evolutionary Algorithms. Proceedings of the International Conference on Genetic Algorithms Mendel '98, Brno, Czech Republic, pp. 90-95

[91] Pelikan, M. A Simple Implementation of Bayesian Optimization Algorithm in C++(Version1.0). Illigal Report 99011, February 1999, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 1999.

[92] Pelikan, M., Goldberg, D., E., Lobo, F.: A survey of optimization by building and using probabilistic models, IlliGAL Report No. 99018, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 1999.

[93] Pelikan, M., Goldberg, D. E., Cantú-Paz, E.: Bayesian Optimization Algorithm, Population Sizing, and Time to Convergence. IlliGAL Report No. 2000001, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois, 2000, 13 pages.

[94] Pelikan, M., Goldberg, D., E.: Hierarchical Problem Solving by the Bayesian Optimization Algorithm. IlliGAL Report No. 2000002, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois, 2000.

[95] Pelikan, M., Goldberg, D., E.: Genetic Algorithms, Clustering, and the Breaking of Symmetry, IlliGAL Report No. 2000013, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 2000.

[96] Pelikan, M., Goldberg, D., E., Sastry, K.: Bayesian Optimization Algorithm, Decision Graphs, and Occam's Razor, IlliGAL Report No. 2000020, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 2000.

[97] Pelikan, M., Goldberg, D. E.: Escaping Hierarchical Traps with Competent Genetic Algorithms. IlliGAL Report No. 2001003, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, January 2001.

[98] Pelikan, M., Goldberg, D. E., Tsutsui, S.: Combining the Strengths of the Bayesian Optimization Algorithm and Adaptive Evolution Strategies. IlliGAL Report No. 2001023, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 2001.

[99] Pelikan, M., Sastry, K. , Goldberg, D., E.: Evolutionary Algorithms + Graphical Models = Scalable Black-Box Optimization, IlliGAL Report No. 2001029, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 2001.

[100] Pelikan, M.: Bayesian optimization algorithm: From single level to hierarchy. Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL. Also IlliGAL Report No. 2002023, 2002.

[101] Peňa, J. M.: On Unsupervised Learning of Bayesian Networks and Conditional Gaussian Networks, PhD thesis, University of the Basque Country, San Sebastian, Spain, July 2001.

[102] Peňa, J. M., Lozano, J. A., Larrañaga, P.: Benefits of data clustering in multimodal function optimization via EDAs. In Larrañaga, P. and Lozano, J. A., editors, Estimation of Distribution Algorithms. A new Tool for Evolutionary Computation. Kluwer Academic Publishers, 2002.

[103] Rudlof, S., Köppen, M.: Stochastic hill climbing by vectors of normal distributions. Proceedings of the First Online Workshop on Soft Computing (WSC1), Nagoya, Japan, 1996.

[104] Rudolph, G.: Evolutionary search under partially ordered fitness sets. In Proceedings of the International Symposium on Information Science Innovations in Engineering of Natural and Artificial Intelligent Systems (ISI 2001), pp. 818-822, 2001.

[105] Rudolph, G.: On a multi-objective evolutionary algorithm and its convergence to the pareto set. *IEEE International Conference on Evolutionary Computation (ICEC'98)*, pp. 511-516, Piscataway, 1998. IEEE Press.

[106] Santana, R., Ochoa, A.: Dealing with Constraints with Estimation of Distribution Algorithms: The Univariate Case, Second Symposium on Artifical Intelligence, Adaptive Systems, CIMAF 99, La Habana, Cuba, pp. 378-384, 1999.

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

[108] Sastry, K., Goldberg, D. E.: On extended compact genetic algorithm. GECCO-2000, Late Breaking Papers, Genetic and Evolutionary Computation Conference, pp. 352-359, 2000.

[109] Schaffer, J. D.: Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms. PhD thesis, Nashville, TN: Vanderbilt University, 1984.

[110] Schwarz J.: Genetic algorithm for partitioning circuits. 4 International Mendel Conference, June 24-26, 1998, Brno, Czech Republic, pp.126-131.

[111] Sebag, M., Ducolombier, A.: Extending population-based incremental learning to continuous search spaces. Parallel Problem Solving from Nature – PPSN V, pp. 418-427, Springer-Verlag, 1998.

[112] Servet, I., Trave-Massuyes, L., Stern, D.: Telephone network traffic overloading diagnosis and evolutionary computation techniques. Proceedings of the European Conference on Artificial Evolution (AE-97), pp. 137-144. 1997.

[113] Srinivas, N., Deb, K.: *Multiobjective Optimization using Nondominated Sorting in Genetic Algorithm.* Evolutionary Computation, Vol.2, No.3, 1994, pp. 221-248.

[114] Surry, P. D.: A Prescriptive Formalism for Constructing Domain-specific Evolutionary Algorithms. PhD thesis, University of Edinburgh, 1998.

[115] Todd, D.S.: *Multiple Criteria Genetic Algorithms in Engineering Design and Operation. PhD thesis,* University of Newcastle, Newcastle-upon-Tyne, UK, October 1997.

[116] Thierens, H.: Analysis and design of genetic algorithms. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 1995.

[117] Thierens, D., Bosman, P.A.N.: Multi-Objective Mixture-based Iterated Density Estimation Evolutionary Algorithms, Proceedings of the Genetic and Evolutionary Computation Conference - GECCO-2001, pp. 663-670, Morgan Kaufmann Publishers, 2001.

[118] Tsutsui, S., Pelikan, M., Goldberg, D. E.: Evolutionary Algorithm Using Marginal Histogram Models in Continuous Domain. IlliGAL Report No. 2001019, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, March 2001.

[119] Wall, M. : GAlib C++ Library of Genetic Algorithm Components, version 2.4, Documentation Revision B, August 1996

[120] Zhang, B. T.: A Bayesian framework for evolutionary computation. Proceedings of the Congress on Evolutionary Computation (CEC 99). IEEE Press, pp. 722-727, 1999.

[121] Zhang, B. T.: Bayesian evolutionary algorithms for learning and optimization. Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM workshop, Genetic and Evolutionary Computation Conference GECCO-2000, pp. 220-222. 2000.

[122] Zhang, Q., Mühlenbein, H.: On global convergence of FDA with proportionate selection. Second Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99, pp. 301-313, La Habana, 1999.

[123] Zitzler, E.: Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications, PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, November 1999.

[124] Zitzler, E., Deb, K., Thiele, L., Coello, C. A. C., Corne, D. editors. Evolutionary Multi-Criterion Optimization (Lecture Notes in Computer Science 1993). Heidelberg: Springer, 2001.

[125] Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. EUROGEN 2001 - Evolutionary Methods for Design, Optimisation and Control, Athens, Greece, CIMNE, 2001.

Publications

[i] Laumanns, M., Očenášek, J.: Bayesian Optimization Algorithms for multi-objective optimization, In: Parallel Problem Solving from Nature - PPSN VII, Springer-Verlag, 2002, pp. 298-307, ISBN 3-540-444139-5, ISSN 0302-9743

[ii] Schwarz Josef, Očenášek Jiří: Bayes-Dirichlet BDD as a probabilistic model for logic function and evolutionary circuit decomposer. In: Proceedings of the 8th International Mendel Conference on Soft Computing, Brno, Mendel 2002, Technical University of Brno, 2002, pp. 117-124, ISBN 80-214-2135-5

[iii] Schwarz, J., Očenášek, J.: Ratio cut hypergraph partitioning using BDD based mBOA optimization algorithm, In: 5th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop 2002, Brno, Czech Rep., 2002, pp. 87-96, ISBN 80-214-2094-4

[iv] Očenášek, J., Schwarz, J.: Development system DEBOA for rapid prototyping of evolutionary applications, In: Proceedings of the 36th International Conference MOSIS'02, Ostrava, Czech Rep., MARQ, 2002, pp. 169-176, ISBN 80-85988-71-2

[v] Očenášek, J., Schwarz, J.: Estimation of Distribution Algorithm for mixed continuous-discrete optimization problems, In: 2nd Euro-International Symposium on Computational Intelligence, Kosice, Slovakia, IOS Press, 2002, pp. 227-232, ISBN 1-58603-256-9, ISSN 0922-6389

[vi] Očenášek, J., Schwarz, J.: The Distributed Bayesian Optimization Algorithm for combinatorial optimization, EUROGEN 2001 - Evolutionary Methods for Design, Optimisation and Control, Athens, Greece, CIMNE, 2001, pp. 115-120, ISBN 84-89925-97-6

[vii] Schwarz, J., Očenášek, J.: Evolutionary Multiobjective Bayesian Optimization Algorithm: Experimental Study, 35th Spring International Conference: Modelling and Simulation of Systems, Hradec nad Moravicí, Czech Rep., MARQ, 2001, pp. 101-108, ISBN 80-85988-57-7

[viii] Schwarz, J., Očenášek, J.: Multiobjective Bayesian Optimization Algorithm for Combinatorial Problems: Theory and Practice, In: NEURAL NETWORK WORLD, Vol. 11, No. 5, CZ, pp. 423-441, ISSN 1210-0552

[ix] Schwarz, J., Očenášek, J.: Pareto Bayesian Optimization Algorithm for the Multiobjective 0/1 Knapsack Problem, In: Proceedings of the 7th International Mendel Conference on Soft Computing, Brno, Czech Rep., 2001, pp. 131-136, ISBN 80-214-1894-X

[x] Očenášek, J., Schwarz, J.: The Parallel Bayesian Optimization Algorithm, In: Proceedings of the European Symposium on Computational Inteligence, Physica-Verlag, Košice, Slovak Republic, 2000, pp. 61-67, ISBN 3-7908-1322-2, ISSN 1615-3871

[xi] Schwarz Josef, Očenášek Jiří: A problem knowledge-based evolutionary algorithm KBOA for hypergraph bisectioning. In: Proceedings of the Fourth Joint Conference on Knowledge-Based Software Engineering, Brno, Czech Republic, IOS Press, 2000, pp. 51-58, ISBN 1-58603-060-4

[xii] Schwarz Josef, Očenášek Jiří: Partitioning-oriented placement using advanced genetic algorithm BOA. In: Proceedings of the 6th International Conference on Soft Computing, Mendel 2000, Technical University of Brno, 2000, pp. 145-150, ISBN 80-214-1609-2

[xiii] Očenášek, J.: The Acceleration of Estimation of Distribution Algorithms, In: Sborník prací studentů a doktorandů VI., CERM, BRNO, 2000, pp. 213-215, ISBN 80-7204-155-X

[xiv] Schwarz, J., Očenášek, J.: Experimental study: Hypergraph partitioning based on the simple and advanced genetic algorithms BMDA and BOA, In: Proceedings of the Mendel '99 conference, Brno University of Technology, Faculty of Mechanical Engineering, BRNO, 1999, pp. 124-130, ISBN 80-214-1131-7

[xv] Očenášek, J.: Advanced genetic algorithms for hypergraph partitioning, In: Sborník prací studentů a doktorandů V., BRNO, CZ, CERM, 1999, pp. 89-90, ISBN 80-214-1155-4, in czech

[xvi] Očenášek, J.: Implementation of advanced genetic algorithm, MSc. thesis, Department of Computer Science, FEI VUT Brno, CZ, 1999, in czech (dean price award)

[xvii] Očenášek, J.: Genetic algorithm for task scheduling, In: Sborník prací studentů a doktorandů IV., BRNO, CZ, CERM, 1998, pp. 53-54, ISBN 80-214-1141-4, in czech

Index