Linking Entropy to Estimation of Distribution Algorithms

Alberto Ochoa and Marta Soto

Institute of Cybernetics, Mathematics and Physics, Cuba
{ochoa,mrosa}@icmf.inf.cu

Summary. This chapter presents results on the application of the concept of entropy to estimation of distribution algorithms (EDAs). Firstly, the Boltzmann mutual information curves are introduced. They are shown to contain a lot of information about the difficulty of the functions. Next, a design method of discrete benchmark functions is presented. The newly developed approach allows the construction of both single and random classes of functions that obey a given collection of probabilistic constraints. This application and the next – the construction of low cost search distributions – are based on the principle of maximum entropy. The last proposal is the linear entropic mutation (LEM), an approach that measures the amount of mutation applied to a variable as the increase of its entropy. We argue that LEM is a natural operator for EDAs because it mutates distributions instead of single individuals.

1 Introduction

Entropy is a measure of the uncertainty of a random variable, whereas mutual information measures the reduction of the entropy due to another variable. These are fundamental quantities of information theory [3], the building blocks of a field that overlaps with probability theory, statistical physics, algorithmic complexity theory and communication theory, among others disciplines.

In this chapter, we explore several novel uses of the concept of entropy in evolutionary optimization. In particular, we investigate intersections of information theory and the field of estimation of distribution algorithms (EDAs) [26].

A major challenge of evolutionary optimization is the preservation of the right balance between exploitation and exploration. From an entropic point of view, exploitation can be seen as a low-entropy search, whereas exploration is better understood as a high-entropy search. This occurs both at the system and variable levels. At the system level, we see how the joint entropy is reduced as the run approaches the optimum. At the variable level, the mutual information comes into play, the reduction in uncertainty of a variable due to the remainder variables is an indicator of what kind of entropic balance should be enforced at that point. These are just few evidences about the fact that entropy is at the heart of the dynamics of artificial evolution. This has been a major motivation of our work. We believe that EDAs will

A. Ochoa and M. Soto: *Linking Entropy to Estimation of Distribution Algorithms*, StudFuzz **192**, 1–38 (2006)
www.springerlink.com

profit from greater efforts in this area of research. Keeping in mind these arguments, in this chapter we approach the following issues:

- A method for analysing the difficulty of the functions (Sect. 3).
- A design method of benchmark functions (Sect. 4).
- A method for learning low cost maximum-entropy distributions (Sect. 5).
- An entropic approach to mutation (Sect. 6).

Nowadays, simulation is a fundamental tool of verification, validation and comparison of evolutionary algorithms. For EDAs, the design of benchmark functions should emphasize, in the first place, the complexity of the probabilistic structure of the search distributions. We have developed a method, which gives the designer the possibility of specifying a collection of probabilistic constraints that have to be fulfilled by the search distributions. The method is connected to the concept of entropy because it constructs a maximum entropy distribution that satisfies the given constraints.

A good design method should be accompanied by a good analysis method. We introduce a new approach for function complexity analysis in the context of EDA optimization. Our approach investigates the mutual information of Boltzmann distributions as a function of the temperature parameter.

A critical problem of learning search distributions in an EDA, is the sample complexity. Large sample sizes mean large number of function evaluations. The challenge is to reduce the number of evaluations, without damaging the effectiveness and efficiency of the search. We use the concept of entropy to achieve this goal; the true search distribution is substituted by a maximum entropy approximation, which can be reliably computed with less population size.

EDAs have to approach mutation from a distribution perspective, in contrast with the genotype perspective of GAs. While a GA mutates single individuals, an EDA must mutate distributions. We have developed an approach that uses the concept of entropy to fulfill this requirement. The relation between entropy and mutation is quite intuitive: when a random variable is mutated, a certain degree of randomness is added to it. Therefore, it seems reasonable to measure the amount of mutation applied to a variable as the increase of its entropy.

The outline of this contribution is as follows. Section 2 presents the background material. Then we discuss the above problems in Sects. 3-6. Finally, the conclusions are given.

2 Background

This section introduces the general notation of the chapter. It also gives a short introduction to the theories that underlie our main results.

2.1 General Notation

In this chapter, X_i represents a scalar random variable and $p(x_i) = p(X_i = x_i)$ its probability mass function with $x_i \in \mathfrak{X} = \{0, 1, \ldots, K\}$. Note that $p(x_i)$ and

$p(x_j)$ refer to two different random variables, and have in fact different probability mass functions, $p(X_i = x_i)$ and $p(X_j = x_j)$, respectively. Similarly, $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$ denotes a *n*-dimensional random variable, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ is a configuration and $p(x_1, x_2, \ldots, x_n)$ represents a joint probability mass. The notation $\boldsymbol{X}_a$ and $\boldsymbol{x}_a$ is used to denote sub-vectors of $\boldsymbol{X}$ and $\boldsymbol{x}$ with indexes from $a \subset \{1, \ldots, n\}$. $p(\boldsymbol{x}_a) = \sum_{x_i, i \notin a} p(\boldsymbol{x})$ and $p(\boldsymbol{x}_a | \boldsymbol{x}_b) = p(\boldsymbol{x}_a, \boldsymbol{x}_b) / p(\boldsymbol{x}_b)$ define marginal and conditional distributions, respectively. $p(a)$ or p_a are used to denote $p(\boldsymbol{x}_a)$.

2.2 Boltzmann Estimation of Distribution Algorithms

At the center of most of the ideas and results of this chapter, lies the Boltzmann distribution. Some authors have considered it as the corner stone of the theory of estimation of distribution algorithms [19, 24, 25]. We believe that this chapter is new evidence that supports this way of thinking.

Definition 1 *For $\beta \geq 0$ define the Boltzmann distribution of a function $f(\boldsymbol{x})$ as*

$$p_{\beta,f}(\boldsymbol{x}) := \frac{e^{\beta f(\boldsymbol{x})}}{\sum_{\boldsymbol{y}} e^{\beta f(\boldsymbol{y})}} = \frac{e^{\beta f(\boldsymbol{x})}}{Z_f(\beta)}$$

where $Z_f(\beta)$ is the partition function.

We also use $Z_{\beta,f}$, but to simplify the notation β and f can be omitted. If we follow the usual definition of the Boltzmann distribution, then $-f(\boldsymbol{x})$ is called the free energy and $1/\beta$ the temperature of the distribution. The parameter β is usually called the inverse temperature.

Closely related to the Boltzmann distribution is Boltzmann selection:

Definition 2 *Given a distribution $p(\boldsymbol{x})$ and a selection parameter γ, Boltzmann selection calculates a new distribution according to*

$$p^s(\boldsymbol{x}) = \frac{p(\boldsymbol{x}) e^{\gamma f(\boldsymbol{x})}}{\sum_{\boldsymbol{y}} p(\boldsymbol{y}) e^{\gamma f(\boldsymbol{y})}}$$

Boltzmann selection is important because the following holds [25]:

Theorem 1 *Let $p_{\beta,f}(\boldsymbol{x})$ be a Boltzmann distribution. If Boltzmann selection is used with parameter γ, then the distribution of the selected points is again a Boltzmann distribution with*

$$p^s(\boldsymbol{x}) = \frac{e^{(\beta+\gamma) f(\boldsymbol{x})}}{\sum_{\boldsymbol{y}} e^{(\beta+\gamma) f(\boldsymbol{y})}}$$

The Boltzmann estimation of distribution algorithm (BEDA) was introduced in [25] on the basis of the above. Here, it is shown as Algorithm 1. BEDA is an algorithm with good theoretical properties, it has even a convergence proof. However, in the form in which it is shown in algorithm 1, it is just a conceptional algorithm.

Algorithm 1 BEDA – Boltzmann Estimation of Distribution Algorithm

Step 1 $t \leftarrow 0$, $\beta(t) \leftarrow 0$ and $p(x\boldsymbol{x}, t) = \frac{1}{Z_{\beta(t),f}}$

Step 2 $t \leftarrow t+1$, $\Delta\beta(t) \leftarrow \beta(t) - \beta(t-1)$ and

$$p(\boldsymbol{x}, t+1) \leftarrow \frac{p(\boldsymbol{x}, t) e^{\Delta\beta(t) f(\boldsymbol{x})}}{\sum_{\boldsymbol{y}} p(\boldsymbol{y}, t) e^{\Delta\beta(t) f(\boldsymbol{y})}} \tag{1}$$

Step 3 If the stopping criterion is not reached, go to step 2.

The reasons are twofold: the exponential complexity of the denominator of (1) and the lack of a method for updating $\Delta\beta(t)$.

The next lemma solves the second problem. The reader is referred to [19] for details.

Lemma 1 *$\Delta\beta(t) = c/\sqrt{Var_f(\beta(t))}$ leads to an annealing schedule where the average fitness, $W_f(\beta(t))$, increases approximately proportional to the standard deviation:*

$$W_f(\beta(t+1)) - W_f(\beta(t)) \approx c\sqrt{Var_f(\beta(t))}$$

where c is a constant and $Var_f(\beta(t)) = \sigma_f^2(\beta(t))$ is the variance of the fitness function. This annealing schedule has been called standard deviation schedule (SDS).

The exponential complexity of computing the partition function can be avoided if the Boltzmann distribution is approximated with a tractable distribution. There are several ways of accomplishing this approximation [23]. However, for the purposes of this chapter it is enough to restrict ourselves to the special case covered by the factorization theorem [25].

The factorization theorem defines how and under what conditions the search distributions associated to discrete functions can be factorized. The factorization follows the structure of the function and is only exact if the function obeys certain structural constraints.

Definition 3 *Let $s_i \subseteq \{1, \ldots, n\}$ $(1 \leq i \leq m)$ be index-sets and let $f^{(i)}$ be functions depending only on the variables X_j $(j \in s_i)$. Then, $f(\boldsymbol{x}) = \sum_{i=1}^{m} f^{(i)}(\boldsymbol{x}_{s_i})$ is an additive decomposition of the fitness function $f(\boldsymbol{x})$.*

Definition 4 *Given $s_1, \ldots, s_m$, the sets d_i, b_i and c_i $(i = 1, \ldots, m)$ are defined as follows: $d_0 := \emptyset$, $d_i := \bigcup_{j=1}^{i} s_j$, $b_i := s_i \setminus d_{i-1}$ and $c_i := s_i \cap d_{i-1}$.*

Theorem 2 *(Factorization theorem) For $\beta \geq 0$, let $p_{\beta,f}(\boldsymbol{x})$ be a Boltzmann distribution of a function $f(\boldsymbol{x})$, and $f(\boldsymbol{x}) = \sum_{i=1}^{m} f^{(i)}(\boldsymbol{x}_{s_i})$ be an additive decomposition. If $d_m = \{1, \ldots, n\}$ and the following holds*

$$\begin{array}{l} \forall i \in \{1, \ldots, m\}, b_i \neq 0 \\ \forall i \geq 2, \exists j < i \quad such\, that\ c_i \subseteq s_j \end{array} \tag{2}$$

then

$$p_{\beta,f}(\boldsymbol{x}) = \prod_{i=1}^{m} p\left(\boldsymbol{x}_{b_i} | \boldsymbol{x}_{c_i}\right) \tag{3}$$

The proof can be found in [25]. Assumption 2 is called the running intersection property [16].

In the simulations of this chapter we use mainly two algorithms: the factorized distribution algorithm (FDA) and the Boltzmann FDA. Both algorithms use the factorization (3) as a model of the search distributions. However, while the FDA uses truncation selection, the BFDA uses Boltzmann selection with SDS.

The following lemma is relevant to this chapter [19].

Lemma 2 *BFDA is invariant under linear transformation of the fitness function with a positive factor.*

2.3 Factorizations

As was said in the previous section, the factorization of probability distributions is a major concern of EDA researchers. In this chapter, Bayesian factorizations are specially relevant. They are connected with the concept of Bayesian network.

A Bayesian network (BN) [30, 31] is a directed acyclic graph containing nodes, representing the variables, and arcs, representing probabilistic dependencies among nodes. For any node (variable) X_i, and set of parents π_{X_i}, the Bayesian network specifies a conditional probability distribution $p(x_i \mid \pi_{x_i})$.

There are single-connected – no more than one undirected path connects two nodes – and multiple-connected BNs. The single-connected BNs are also called polytrees. In a polytree, a node may have several parents and many roots. Trees are special class of polytrees, which have at most one parent and one root. Polytrees describe higher-order interactions than trees, while retaining many of their computational advantages. In a polytree, structures like $X \rightarrow Z \leftarrow Y$ are often called head-to-head patterns. This type of pattern makes X and Y conditionally dependent given Z, which cannot be represented by a tree.

A junction tree [10, 14, 16] is an undirected tree, where each node contains a set of variables. The junction tree satisfies the *junction property*: for any two nodes a and b and any node h on the unique path between a and b, $a \cap b \subseteq h$. The arcs between the nodes are labelled with the intersection of the adjacent nodes; usually, they are called *separating sets* or *separators*.

Junction trees are important for inference and sampling because they have tractable algorithms for these tasks. Given a BN, it is possible to construct at least one junction tree. The reader is referred to [10, 14, 16] for a complete discussion on the issue.

2.4 Entropy and Mutual Information

The entropy $H\left(X\right)$ of a discrete random vector X is defined in [3] by

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{4}$$

Note that entropy is a functional of the distribution of X. It does not depend on the actual values taken by the random variable, but only on the probabilities. This means that $H(X)$ is a shortcut for $H(p(X))$. The logarithm in (4) is to the base two and entropy is expressed in bits. We use the convention that $0 \log 0 = 0$.

For a binary variable X, such that $p(X = 1) = p$, we have

$$H(X) = H(p(X)) = H(p) := -p \log p - (1-p) \log (1-p) \tag{5}$$

The entropy of a binary variable is a nonnegative, symmetric and concave function of the distribution. It has the maximum at the point $(0.5, 1)$ and it is zero for $p \in \{0, 1\}$.

The following theorem will be useful later on.

Theorem 3 *(Independence bound on entropy [3]). Let $p(\boldsymbol{x})$ be any joint probability mass of a set of discrete random variables $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$, then*

$$H(\boldsymbol{X}) \leq \sum_{i=1}^{n} H(X_i)$$

with equality if and only if the variables are independent.

The concepts of marginal and conditional mutual information will be intensively used in the chapter. The mutual information, $I(X, Y)$, is the reduction in the uncertainty of X due to the knowledge of Y. The conditional mutual information, $I(X, Y|Z)$, represents the reduction in the uncertainty of X due to the knowledge of Y given Z. The following theorem connects entropy and mutual information.

Theorem 4 *Between mutual information and entropy the following holds [3]:*

$$I(X, Y) = H(X) + H(Y) - H(X, Y) \tag{6}$$

$$I(X, Y|Z) = H(X|Z) - H(X|Y, Z) \tag{7}$$

The Maximum-Entropy Principle

The maximum-entropy principle (MEP) plays an important role in this chapter. It is used to build probability mass functions that fulfill a collection of marginal constraints. The ideas behind this concept can be shortly explained as follows.

Frequently, partial prior information is available outside of which it is desired to use a prior that is as non-informative as possible. For example, suppose some prior marginal distributions are specified, and among prior distributions with these marginals the most non-informative distribution is sought [12, 13]. If we have the joint distribution with the maximum-entropy of all the joints that fulfill a given collection of marginals, choosing a joint with less entropy amounts to add some information that is not justified by the constraints.

The iterative proportional fitting (IPF) algorithm can be used to find the maximum-entropy distribution [11, 12, 18, 32]. The proof that IPF converges against the maximum-entropy solution can be found in [4]. Unfortunately, the naive implementation of the IPF takes exponential time and space. Therefore, it is not suitable for computing distributions with many variables.

For large distributions, an efficient implementation of the maximum-entropy algorithm was developed in [15, 21]. The general idea is to improve the performance of IPF by combining it with the junction tree technique. It consists of performing IPF locally on the nodes and passing messages to the neighboring nodes. It has been proved that this converges to the unique maximum-entropy solution, so it is equivalent to IPF. The reader is referred to [29] for details on the implementation of the method for computing maximum-entropy distributions of polytrees.

3 Mutual Information and Functions Difficulty

This section presents preliminary ideas about a novel method for analysing the complexity of functions for evolutionary algorithms. The corner stone of the approach is the concept of mutual information, which is studied through its relation with selection.

3.1 Boltzmann Mutual Information Curves

The Goldberg's Deceptive3 function belongs to the class of the so called deceptive problems [6, 7] that are those having local optima which are easier to find than global optima. Deceptive problems contain deceptive attractors, which mislead the algorithm to search for sub-optima because their basins of attraction are much larger than the ones favoring global optima. Often, deceptiveness is considered a challenge to search algorithms. However, deception is a relative category that emerges solely in the context of the relationship problem-algorithm. In other words, a problem may be deceptive for one algorithm, but not for another.

Deception has been intensively studied in the context of genetic algorithms. In [6, 7, 9], the authors described ways to construct deceptive functions and gave sufficient conditions for deception. Figure 1 (left) shows the usual way of describing deceptive problems as a function of unitation. Note, the deep valley separating the optimum from the sub-optimum and the different sizes of their attractors.

In this section, we introduce a new method for analysing the function complexity in the context of EDA optimization. Our approach investigates the mutual information of Boltzmann distributions as a function of the parameter β. Given a function f, this method computes the Boltzmann distribution $p_{f,\beta}$ for $\beta > 0$. Then, it computes the marginal and the conditional mutual information on any sub-set of variables. We show that the Boltzmann mutual information curves, $I(\beta)$, contain a lot of information about the complexity of the function.

Table 1 shows the function Deceptive3 and its Boltzmann distribution for $\beta = 10.49$. On the other hand, Fig. 1 (right) presents the marginal mutual information

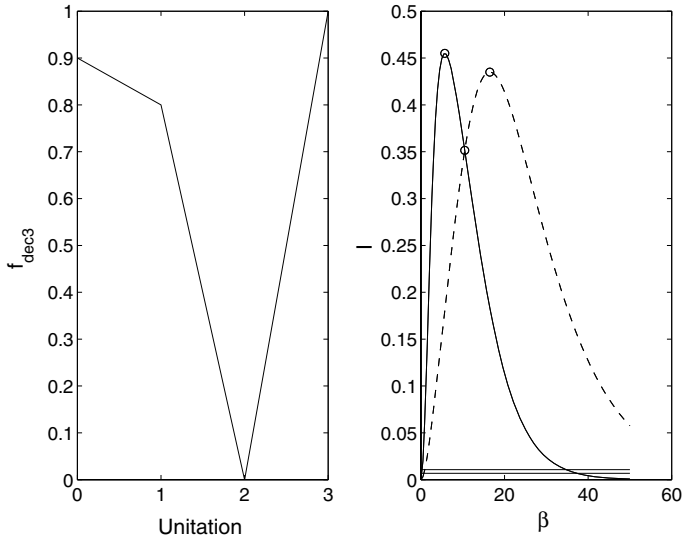

Fig. 1. Explaining the complexity of Goldberg's Deceptive3 function: (*left*) unitation approach – the optimum is isolated and separated from the sub-optima by a deep valley (*right*) mutual information approach – marginal (*dashed line*) and conditional (*solid line*)

Table 1. Goldberg's Deceptive3 function and its Boltzmann distribution for $\beta = 10.49$. At this value, $I(X,Y) = I(X,Y|Z)$

$x_3x_2x_1$	$f_{dec3}(\mathbf{x})$	$p_{\beta=10.49}(\mathbf{x})$	$x_3x_2x_1$	$f_{dec3}(\mathbf{x})$	$p_{\beta=10.49}(\mathbf{x})$
000	0.9	0.2038	100	0.8	0.0714
001	0.8	0.0714	101	0	0
010	0.8	0.0714	110	0	0
011	0	0	111	1	0.5820

and the conditional mutual information. Note that all edges have the same marginal and conditional values of mutual information, i.e. the function is symmetric. This property of the Deceptive3 simplifies its analysis.

To begin with, we recall a result that was presented in [35], which states that the difference between conditional and marginal mutual information is invariant to permuting the variables. Remarkably, the result holds for any three sets of variables X_a, X_b and X_c.

Proposition 1 *(Whittaker [35, Proposition 4.5.1]) Suppose that the partitioned random vector (X_a, X_b, X_c) has a joint density function f_{abc}. The difference between the divergence against the conditional independence of X_a and X_b given X_c and the marginal independence of X_a and X_b is invariant to permuting the symbols X_a, X_b and X_c.*

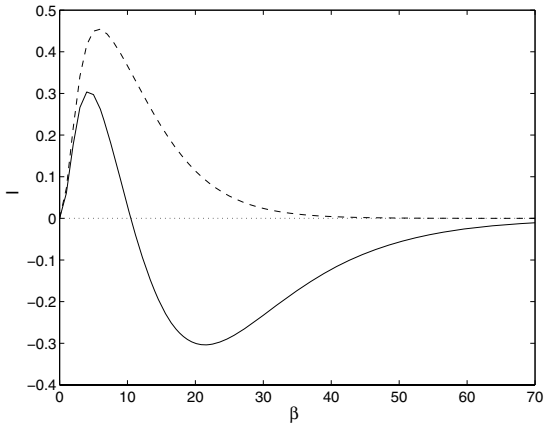

Fig. 2. Conditional information $I\left(X,Y|\,Z\right)$ (*dashed line*) and $G\left(X,Y,Z\right)=I\left(X,Y|\,Z\right)-I\left(X,Y\right)$ (*solid line*)

The above difference is denoted by $G\left(a,b,c\right)$. As a consequence of the proposition 1, the curve $G\left(a,b,c\right)$ and the three conditional information curves also contain all the marginal mutual information. Therefore, we also use pictures like Fig. 2 as tools for analysing the complexity of functions. In our framework, we refer to these curves as Boltzmann-mutual-information curves or simply Boltzmann-information curves.

From an evolutionary point of view, the Boltzmann-information curves show how selection influences the strength of the dependencies among the variables of the problem. If the algorithm uses Boltzmann selection as is the case of BEDAs, then β directly measures the selection pressure. Although for other selection schemes the connection is not direct, the information gathered from curves is still useful.

The curves are continuous, monotonously increasing up to their maximum values and decreasing to zero as β increases. This simple observation has an important implication for learning: there is a strong correlation between mutual information values at consecutive steps of the evolution.

Note in Fig. 1 (right), the horizontal lines at $I\approx 0.0069$ and $I\approx 0.0107$; they are thresholds for marginal and conditional independence[1]. We recall that $I\left(X,Y\right)=I\left(X,Y|\,\emptyset\right)$; it is assumed that the empty set has zero variables and thus $|\emptyset|=1$. The above thresholds were computed with a confidence level of 95% and a sample size of $N=280$ (this is the sample size used in the numerical simulations).

We now discuss the critical points of the Boltzmann-information curves. There are nine important critical points: the intersections of the threshold lines with

[1] Under the null hypothesis that conditional independence of X and Y given Z holds, the value $2NI\left(X,Y|\,Z\right)$ – which is called deviance against conditional independence – approximates a $\chi 2$ distribution with $|Z|\left(|X|-1\right)\left(|Y|-1\right)$ degrees of freedom, where N is the number of configurations in the sample and $|S|$ represents the number of possible values of the set of variables in S [35, Proposition 7.6.2]

the marginal and conditional Boltzmann curves determine two pairs of β values that define a marginal and a conditional dependence intervals, $[\beta^m_{min}, \beta^m_{max}]$ and $[\beta^c_{min}, \beta^c_{max}]$, respectively; the maximal values of the curves, β^m_M, β^c_M and β^G_M; the zero and minimum value of $G\,(1,2,3)$, β^G_z and β^G_m respectively.

3.2 Dissection of the Goldberg's Deceptive3 Function

In this section we investigate the separable function

$$F_{dec3} = \sum_{i=1}^{l} f_{dec3}\,(x_{3i-2}, x_{3i-1}, x_{3i})$$

and some other functions derived from it. As a rule we use the BFDA, but a few results are also presented for a FDA with truncation selection.

The notation used in the tables is as follows: N is the population size, n is the number of variables, $\%S$ is the success rate in 100 independent runs and G_c is the average generation where the optimum is found. For the average β values, we use β_{min} after the initial selection and β_{max} at the end of successful runs.

Deception and the Complete Bayesian Model

We start our investigation of the Deceptive3 by running the BFDA with the complete Bayesian model of the marginal distributions $p\,(x_{3i-2}, x_{3i-1}, x_{3i})$. In other words, it uses the factorizations

$$p\,(x_{3i-2}, x_{3i-1}, x_{3i}) = p\,(x_{3i-2})\,p\,(x_{3i}|\,x_{3i-2})\,p\,(x_{3i-1}|\,x_{3i-2}, x_{3i}) \qquad (8)$$

Equation (8) is the natural model for this function; any other model performs worse than it does. The following simulation confirms this behaviour. We run the BFDA 100 times, in a problem with 30 variables and 280 configurations. The algorithm always finds the optimum with $G_c = 12.97$. The average β at the end of the runs is 18.43, whereas the critical point β^G_z is reached as average at the generation 10. This means that for approximately $3/4$ of the evolution the conditional information is stronger than the marginal information.

As can be seen from Fig. 1 (right) the variables are marginally and conditionally dependent in the range of β observed in the simulation of $[0, 18.43]$. Note that this interval is completely included in $[\beta^c_{min}, \beta^c_{max}] \subset [\beta^m_{min}, \beta^m_{max}]$. We recall that for three variables the complete model is the only one that does not have any independence relation, i.e. it is the best for the pair BFDA-Deceptive3.

We believe that deceptiveness is a direct consequence of having high values of mutual information. As we pointed out before, deception is a relative category that emerges solely in the context of the relationship problem-algorithm. In this relationship the problem contributes with high values of mutual information, whereas the algorithm's contributions are the selection and the collection of dependencies that it

can deal with. The collection must be a proper sub-set of the problem's dependencies. We believe that the size and strength of the basins of attraction for any problem attractor depend on the amount of mutual information relevant to it. Without these three ingredients there can not be any deception at all. The amount of mutual information is a source of difficulty even when the right model or factorization is used.

BFDAs are perfect tools for studying the difficulty of the functions. They have everything that is needed:

- Selection is given explicitly through the parameter β.
- The collection of dependencies the algorithm can deal with are fixed by the factorization.
- The relation between mutual information and selection is given by the Boltzmann information curves.

In BFDAs, deception arises in the context of the relationship problem-factorization, i.e. a given problem may or may not be deceptive in relation to a particular factorization.

Reducing the Mutual Information

Let p_{f_{dec3},β_z^G} be the Boltzmann distribution of the Deceptive3 with β_z^G and Z_{f,β_z^G}, i.e. the distribution when the mutual and conditional information are the same (see Fig. 1).

In this section, we deal with the family of functions

$$f_{dec3}(\alpha) = \frac{log(p_\alpha)}{\beta_z^G} + \frac{log\left(Z_{f,\beta_z^G}\right)}{\beta_z^G} \tag{9}$$

where $\alpha \in \{0, 0.05, 0.20, 0.40, 0.50\}$ and p_α is a distribution that obeys the following entropic relation

$$H(p_\alpha) = (1-\alpha) H\left(p_{f_{dec3},\beta_z^G}\right) + 3\alpha$$

This type of entropic relation is discussed in Sect. 6.3. For the purposes of the current section it is enough to say that the mutual information in p_α decreases as α grows.

Table 2 shows the family of $f_{dec3}(\alpha)$ functions. Note that $f_{dec3}(0)$ is the Deceptive3. Besides, it is worth noting, that the symmetry of the Boltzmann information curves for the Deceptive3 is slightly broken in these functions. However, the difference is so small, that it is enough to show in Fig. 3 only the curves $I(1,2)$ and $I(1,2|3)$. The reader can easily check this by constructing the Boltzmann mutual information curves of these functions.

Table 3 presents the numerical results. The difficulty of the function decreases with increasing α, which means with increasing joint entropy and with decreasing mutual information. Note the influence of α in the convergence time: as α grows, G_c decreases. On the other hand, both β_{min} and β_{max} increase as α grows. We recall

Table 2. The family of $f_{dec3}\,(\alpha)$ functions

	$x_3x_2x_1$							
α	000	001	010	011	100	101	110	111
0.00	0.90	0.80	0.80	0.00	0.80	0.00	0.00	1.00
0.05	0.90	0.80	0.80	0.47	0.80	0.40	0.39	1.00
0.20	0.90	0.82	0.81	0.63	0.81	0.57	0.56	0.99
0.40	0.89	0.83	0.82	0.71	0.82	0.66	0.65	0.98
0.50	0.89	0.83	0.83	0.74	0.82	0.70	0.69	0.97

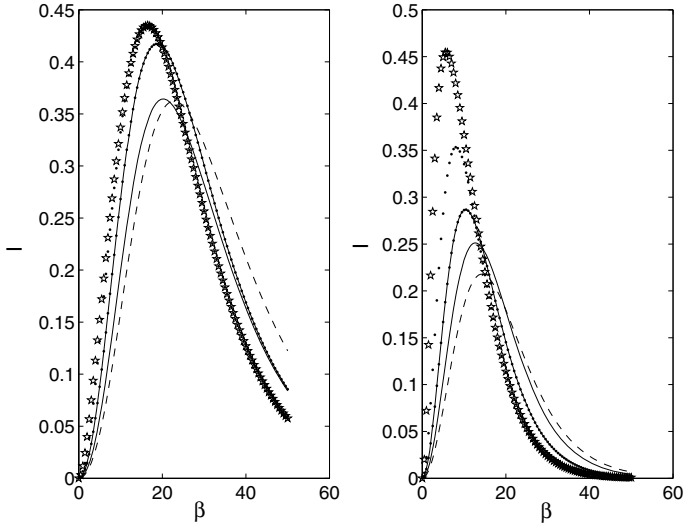

Fig. 3. Boltzmann mutual information curves for the family $f_{dec3}\,(\alpha)$: (*left*) marginal, (*right*) conditional. From *top* to *bottom*, f_{dec3}, $f_{dec3}\,(0.05)$, $f_{dec3}\,(0.20)$, $f_{dec3}\,(0.40)$ and $f_{dec3}\,(0.50)$

Table 3. BFDA runs with the $f_{dec3}\,(\alpha)$ with the complete Bayesian model. The average β values after the initial selection and at the end of successful runs are shown in columns β_{min} and β_{max}, respectively. Setting: $N = 280$, $n = 30$

α	$\%S$	G_c	β_{min}	β_{max}
0.00	100	12.97	0.75	18.43
0.05	100	10.31	1.41	18.25
0.20	100	9.32	2.14	21.11
0.40	100	8.39	2.96	23.14
0.50	100	8.14	3.53	25.86

that $\beta_{min} = \beta(1) = \Delta\beta(1) = c/\sqrt{Var_f(\beta(1))}$, i.e. the standard deviation of the fitness decreases in the first generation with increasing α. Besides, for all functions we have that the interval $[\beta_{min}, \beta_{max}]$ is included in their respective $[\beta^c_{min}, \beta^c_{max}]$.

If the reader constructs the unitation representation (Fig. 1) of the functions $f_{dec3}(\alpha)$, he or she will observe that only the depth of the valley at unitation equal to two changes significantly. For example, $f_{dec3}(0.05)$ is exactly equal to the Deceptive3, except in the case when the unitation is equal to two. This is remarkable because the definition of these functions did not consider any unitation argument.

Models with a Missing Arc

We investigate the performance of the BFDA when the marginal distributions of the form $p(x_{3i-2}, x_{3i-1}, x_{3i})$ are approximated with all Bayesian models with one missing arc. Consider the following factorizations:

$$p^{12-32}(x_1, x_2, x_3) = p(x_1)\, p(x_3)\, p(x_2 | x_1, x_3) \tag{10}$$

$$p^{13-32}(x_1, x_2, x_3) = p(x_1, x_3)\, p(x_2 | x_3) \tag{11}$$

$$p^{12-13}(x_1, x_2, x_3) = p(x_1, x_3)\, p(x_2 | x_1) \tag{12}$$

Due to the symmetry of the function with respect to the mutual information, it is enough to study these cases. For example, in the factorization 12-32 the arc 1-3 is missing and the arcs $1 \rightarrow 2$ and $3 \rightarrow 2$ are present. However, it behaves exactly as the factorizations 21-31 and 13-23.

The results are presented in the first row ($\alpha = 0$) of Table 4. The BFDA behaves much better with the factorization 12-32 than with the factorizations 12-13 and 13-32. The use of the last two factorizations leads to similar results. In what follows, we try to explain this behaviour in the context of Boltzmann information curves.

It is worth noting, that β_{max} is about 30 for all models, which is close to β^c_{max}. Furthermore, we have observed that the critical value β^G_z is reached as average in the generation 10 with the model 12-32 and in the generation 12 with the models 12-13 and 13-32. This means that a successful run occurs in range of β where both the marginal and the conditional information are above the independence thresholds, i.e. the variables are not independent. Moreover, during the first half of the evolution (before β^G_z is reached) $G(1, 2, 3) > 0$.

Table 4. BFDA runs with the $f_{dec3}(\alpha)$. The marginal distributions $p(x_{3i-2}, x_{3i-1}, x_{3i})$ are approximated with all two-arcs models. Setting: $N = 280$, $n = 30$

	12-32			12-13			13-32		
α	$\%S$	G_c	β_{max}	$\%S$	G_c	β_{max}	$\%S$	G_c	β_{max}
0	94	18.57	30.5	22	21.82	30.48	34	22.20	29.64
0.05	99	14.75	29.58	92	16.15	29.31	84	16.11	28.67
0.20	100	12.97	32.03	99	13.25	29.69	95	13.14	28.82
0.50	100	11.24	37.26	100	10.46	32.95	100	10.37	31.89

By comparing (10)–(12) with the chain rule, it is easy to see that each equation makes exactly one wrong assumption:

- Equation (10) assumes marginal independence of X_1 and X_3.
- Equation (11) assumes conditional independence of X_2 and X_1 given X_3.
- Equation (12) assumes conditional independence of X_2 and X_3 given X_1.

The conditional mutual information is farther away from its independence threshold than the marginal mutual information. The independence lines get closer as the sample size increases; for $N = 280$, their difference is just 0.0038. Therefore, we can assume that there is a unique threshold I_t. It is easy to see that $\int_0^{\beta_z^G} G(1,2,3)\, d\beta$ can be used as an estimate of the magnitude of the error of using the factorizations 12-13 or 13-32 instead of 12-32. In other words, the assumption of the model 12-32 is much less traumatic than the other assumptions when $\beta \in [\beta^c_{min}, \beta^c_{max}]$. The situation is reversed for $\beta > \beta^c_{max}$, but this happens when the first half of the evolution is already gone, thus having little impact in the outcome of the optimization.

We have also tested the above factorizations with the functions $f_{dec3}(\alpha)$. Table 4 presents the results. As was shown in Sect. 3.2, the reduction of the mutual information also implies a reduction of the difficulty of the function. Here, we can observe the effect on the convergence time as well as on the success rate. Note for example, that from $\alpha = 0$ to $\alpha = 0.05$ the success rate goes from 22% to 92% in the case of the factorization 12-13. Another interesting observation is about the difference between the performance of different factorizations as α grows. For example, the difference between the convergence time for the complete factorization (8), 12-13-32, and for the factorization 12-13 decreases as α grows: 8.85, 5.84, 3.93, 2.66 and 2.32. We believe that the last result is an evidence supporting the following statement: the reduction of the mutual information increases our choices in model selection.

Some Results with Truncation Selection

For the sake of completeness, Table 5 presents the results of running a FDA with truncation selection on the family of functions $f_{dec3}(\alpha)$. The reader can easily check the similarities of these results with those obtained with Boltzmann selection. For example, they also support the claim that the reduction of the mutual information amounts to a reduction of the functions difficulty.

4 Designing Test Functions by Maximum-Entropy

In spite of recent research advances in EDAs we still do not have a complete, sound, consistent and rigorous theory of evolutionary algorithms. In practice, this leads to the use of simulation as a fundamental tool of verification, validation and comparison of algorithms. One common simulation method is the use of test functions obtained by concatenation of elementary functions of small order. Usually, the design of such functions is focused on considerations about specific aspects of the complexity of

Table 5. FDA runs with the $f_{dec3}(\alpha)$. Setting: $N = 280$, $n = 30$ and truncation selection of 0.3

	12-13-32		12-32		12-13		13-32	
α	$\%S$	G_c	$\%S$	G_c	$\%S$	G_c	$\%S$	G_c
0	100	5.99	91	8.49	35	9.74	34	9.68
0.05	100	5.74	95	8.18	65	8.61	75	8.52
0.20	100	5.32	99	7.18	85	7.61	89	7.65
0.50	100	4.91	100	6.74	100	6.30	99	6.42

the elementary functions: multimodality, isolation of the optimum value, proximity of the function values of the good configurations, frustration of overlapped elementary functions, etc. In this scenario, it is important to know the properties of the elementary functions and how these properties are combined to define the properties of the whole function. Moreover, it would be useful to design functions that are not given as a combination of smaller elementary functions.

The design of benchmark functions for testing EDAs have to emphasize, in the first place, the complexity of the probabilistic structure of the search distributions. The fitness function, the intensity and type of selection determine for each configuration its probability of being in the selected set and consequently the probabilistic structure of the search distributions.

A successful EDA builds a probabilistic model that captures the important correlations of the search distribution, assigning high probability values to the selected configurations. Therefore, it would be convenient to design functions that enforce a given set of "important correlations", but do not enforce any other correlation constraint. In this section, we present an approach to this problem, where the designer gives a collection of probabilistic constraints that have to be fulfilled by the search distributions of the function. Our method is connected to the concept of entropy because it constructs a maximum-entropy distribution that satisfies the given constraints.

4.1 The General Framework

The corner stone of our approach to the design of benchmark functions for discrete optimization is what we have called the family of Boltzmann functions

$$f_{\beta}(\boldsymbol{x}) = \frac{log(p_{f,\beta}(\boldsymbol{x}))}{\beta} + \frac{log(Z_f(\beta))}{\beta} \tag{13}$$

Equation (13) comes from the definition of the Boltzmann probability mass $p_{f,\beta}(\boldsymbol{x})$. From the point of view of this model, (13) are members of the parametric class $\mathfrak{F}(\beta, Z, p_{f,\beta}(\boldsymbol{x}))$, which could be refined by including additional parameters of the distribution $p_{f,\beta}(\boldsymbol{x})$. For example, when the distribution factorizes and no factor contains more than K variables, we are dealing with the parametric sub-class $\mathfrak{F}(\beta, Z, p_{f,\beta}(\boldsymbol{x}), K)$.

Avoiding the Exponential Effort

The computation of the partition function is always problematic; it needs an exponential effort. Fortunately, in our approach this can be avoided. Note that in (13), the second term is a constant that is added in all configurations. It is a shift along the fitness dimension and has little to do with the complexity of the function. Therefore, nothing prevents us from fixing the value of the partition function. Moreover, for BFDA the following lemma holds.

Lemma 3 *The difficulty of (13) for a BFDA is completely determined by the distribution* $p_{f,\beta}(\boldsymbol{x})$.

Proof 1 *The proof follows immediately from lemma 2.*

If the distribution $p_{f,\beta}(\boldsymbol{x})$ is known and Z is set to an arbitrary value, then the function $f_\beta(\boldsymbol{x})$ is well defined for any β, i.e. for any configuration $\boldsymbol{x}$, the value $f_\beta(\boldsymbol{x})$ can be computed. This means that the computation of the function for all possible configurations is not necessary.

Usually, we use factorizations to deal with the exponential complexity of distributions. In the context of functions design, the factorizations also help to compute the optima and the central moments of the functions. This kind of information is useful to understand the functions' properties. Moreover, sometimes it is useful to have a fast procedure for computing the optima of benchmark functions when testing evolutionary algorithms. For example, when the benchmark functions are drawn from a distribution (Sect. 4.3) and the optima are needed to set the stopping criteria. The reader is referred to [27, 28] for a complete description of two methods that compute the above-mentioned values for junction tree factorizations.

Whenever we have a distribution we can build a Boltzmann function. For example, there are famous Bayesian networks (like the ALARM network [2]) that can be used for this purpose. However, in this chapter we are more interested in the case when, instead of having a distribution, we have a collection of probabilistic constraints that must be satisfied by the distribution.

Dealing with Mutual Information Constraints

We have already met the family of functions (13) in Sect. 3.2. Also we have learned that the mutual information of $p_{f,\beta}(\boldsymbol{x})$ contains a lot of information about the complexity of the function $f_\beta(\boldsymbol{x})$. Therefore, when dealing with complexity issues, it makes sense to design functions that fulfill mutual information constraints like:

$$\begin{aligned} &I(X_a, X_b | X_c) \geq A \\ &I(X_a, X_b | X_c) \leq B \\ &I(X_a, X_b | X_c) \leq I(X_d, X_e | X_f) \end{aligned} \tag{14}$$

In (14), the letters a, b, c, d, e and f denote sub-sets of indexes, and A, B are constants. Moreover, X_c and X_f may be empty, meaning that the expressions represent marginal information.

We formulate the general design problem as follows:

> Given a collection of mutual information constraints $\mathfrak{C} = \{c_1, \ldots, c_L\}$, find a function $f(\boldsymbol{x})$, whose Boltzmann distribution satisfies $\mathfrak{C}$ within a given temperature interval.

Our approach to the above-mentioned problem considers structural and parametric constraints. The structural constraints are specified by Bayesian or Markov networks, which use the separation and d-separation concepts [30] to codify statements of probabilistic independence. The parametric constraints are statements about the configurations' probabilities.

In our method, the inequality

$$A_k \leq \sum_{i=0}^{M-1} a_{ik} p\left(x^{(i)}\right) \leq B_k \tag{15}$$

denotes the k-th parametric constraint. The sum is for all configurations $x^{(i)}$ of X, i.e. M denotes the size of the space. A_k, B_k are real constants and $a_{ik} \in \{0, 1\}$.

It is worth noting, that some sub-sets of the inequalities (15) may define marginal distributions of $p(\boldsymbol{x})$ when $A_k = B_k$ for all inequalities in the sub-set. In this chapter, we deal only with this type of constraint. Therefore, the mutual information constraints (14) have to be mapped to marginal distributions. It is an interesting open question how to translate other types of constraints to probabilistic statements.

Once the collection of marginal constraints has been derived from the mutual information constraints it is necessary to compute the joint probability distribution. The next section presents the issue.

Computing the Joint Probability Distribution

Algorithm 2 presents the general scheme of the design of Boltzmann functions. In the step 2, the algorithm computes a junction tree from the given structural constraints. The computation of a junction tree out from a Bayesian or a Markov network is a well-studied problem [31]. In the step 3, is computed a maximum-entropy distribution that is compatible with the given structural and parametric constraints. There are two possibilities as it is explained below.

The classic implementation of the IPF algorithm can be used to compute the joint probability distribution when the number of variables is small. If the collection of marginals is consistent, the outcome of running the IPF is a maximum-entropy joint.

For larger number of variables, the IPF has to be combined with the junction tree technique. It is run locally on the nodes and the results are sent as messages to the neighboring nodes. It has been proved that this converges to the unique maximum-entropy solution, so it is equivalent to IPF. The interested reader is referred to [23,29] for details on the implementation of the method for computing maximum-entropy distributions on multi-connected Bayesian networks and polytrees.

Finally, in the step 4, the desired function is computed as the function that makes of $p(\boldsymbol{x})$ a Boltzmann distribution with parameters β, Z and $f(\boldsymbol{x})$.

Algorithm 2 A maximum-entropy method for designing Boltzmann functions

Step 1	Input β, Z, and the collection of structural and parametric constraints.
Step 2	Compute a junction tree compatible with the structural constraints.
Step 3	Compute the maximum-entropy junction tree distribution $p(\boldsymbol{x})$ that fulfill the parametric constraints.
Step 4	Output $f_\beta(\boldsymbol{x}) = \frac{log(p(\boldsymbol{x}))}{\beta} + \frac{log(Z)}{\beta}$

4.2 Designing the First-Polytree Functions

In this section, we take a closer look at our method through the design of three binary functions whose structure of the search distribution is single-connected. For obvious reasons, we say that they belong to the polytree class of functions. The functions have been called FirstPolytree3 (f^3_{Poly}), FirstPolytree5 (f^5_{Poly}) and OneEdge ($f^3_{OneEdge}$). Figure 4 presents their graph definitions, i.e. their structural constraints.

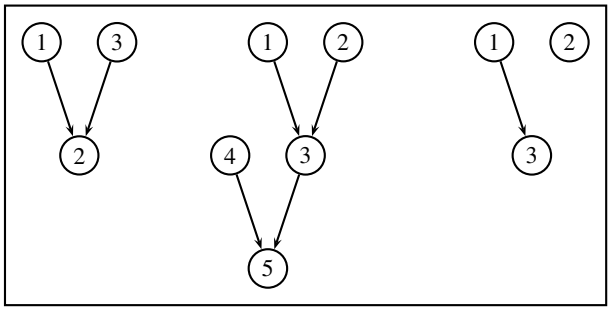

Fig. 4. Structural constraints of the first-polytree functions. From *left* to *right*: f^3_{Poly}, f^5_{Poly} and $f^3_{OneEdge}$

The polytree functions can be specified with simple mutual information constraints. The marginal mutual information of every pair of parents of a variable should be below the marginal independence threshold I_t, for the given confidence level. Similarly, the marginal mutual information of every child-parent pair should be greater than I_t.

We first list the marginal mutual information constraints:

OneEdge: $I(1,3) > I_t \; I(1,2) < I_t \; I(2,3) < I_t$

FirstPolytree3: $I(1,3) < I_t \; I(1,2) > I_t \; I(2,3) > I_t$

FirstPolytree5: $I(1,2) < I_t \; I(3,4) < I_t \; I(1,3) > I_t$
$I(2,3) > I_t \; I(3,5) > I_t \; I(4,5) > I_t$

Algorithm 3 Designing bivariate marginals

Step 1	Input $I(X,Y)$.
Step 2	Set the univariate probabilities to some random values p_x y p_y.
Step 3	**if** $I(X,Y) < I_t$, **then** set $p_{xy} = p_x p_y$.
Step 4	**if** $I(X,Y) > I_t$, **then** set p_{xy} as far as possible from $p_x p_y$.

Another type of constraints is needed to specify the orientation of the edges. The d-separation concept says that in the structure $X \rightarrow Z \leftarrow Y$, the variables X and Y are marginally independent and conditionally dependent given Z [5]. If I_t^c denotes the conditional independence threshold, then the second list of mutual information constraints is the following:

OneEdge: $I(1,3|2) > I_t^c \; I(1,2|3) < I_t^c \; I(2,3|1) < I_t^c$
FirstPolytree3: $I(1,3|2) > I_t^c \; I(1,2|3) > I_t^c \; I(2,3|1) > I_t^c$
FirstPolytree5: $\begin{array}{l} I(1,3|2) > I_t^c \; I(1,2|3) > I_t^c \; I(2,3|1) > I_t^c \\ I(3,4|5) > I_t^c \; I(3,5|4) > I_t^c \; I(4,5|3) > I_t^c \end{array}$

Designing Bivariate Marginals with Given Mutual Information

Once the list of constraints has been given, we construct a set of bivariate marginals that satisfy the constraints. The algorithm 3 does the job.

It is known, that the sufficient statistics for the specification of any binary bivariate marginal $p(x,y)$, are the values $p_x = p(X=1)$, $p_y = p(Y=1)$ and $p = p_{xy}(X=1, Y=1)$. Moreover, either $p_{xy} \in [\max(p_x + p_y - 1,\, 0),\; p_x p_y]$ or $p_{xy} \in [p_x p_y,\; min(p_x, p_y)]$. Taking the univariate probabilities p_x and p_y as input values, we proceed as follows: if $I(X,Y) < I_t$, then we just make $p_{xy} = p_x p_y$. Otherwise, we put p_{xy} as far as possible from $p_x p_y$ to maximize the mutual information. Finally, the bivariate marginal is given by

$$\begin{array}{ll} p_{xy}(00) = 1 - p_x - p_y + p_{xy} & p_{xy}(10) = p_y - p_{xy} \\ p_{xy}(01) = p_x - p_{xy} & p_{xy}(11) = p_{xy} \end{array} \tag{16}$$

After all univariate and bivariate marginals have been computed, the next step of the Algorithm 2 is the construction of the joint probability.

The classic implementation of the IPF algorithm can deal with our functions because the number of variables is small. If the IPF is run with the above marginals, a trivariate maximum-entropy joint is obtained. For larger number of variables we must resort to the junction tree implementation of the maximum-entropy algorithm.

Each node of the junction tree associated to a polytree is formed by one variable and the set of its parents. This means that the trivariate functions have only one clique and therefore, the simple IPF will be enough. The junction tree for the function FirstPolytree5 contains two cliques and therefore, the advanced implementation of the algorithm is needed. In this way, we have constructed high order marginals using only univariate and bivariate marginals. We must check that the second list of

constraints are also fulfilled. Moreover, to guarantee consistency the design of these marginals must satisfy additionally the following constraint [17]:

Let d be the number of variables in a junction tree node. For all $2 \leq k \leq d$ and all possible choices $j_1, \ldots, j_k$ of k elements out of $\{1, \ldots, d\}$ the condition

$$1 \geq \sum_{i=1}^{k} p_{j_i} - \sum_{i,l=1, i \neq l}^{k} p_{j_i j_l}$$

must be fulfilled.

We use the values 12.94, 16.40 and 87.97 as input values for the partition functions of $f^3_{OneEdge}$, f^3_{Poly} and f^5_{Poly}, respectively. The univariate probabilities also are set. For example, the values used in the function f^3_{Poly} are 0.79, 0.46 and 0.24 for X_1, X_2 and X_3, respectively. Finally, we set $\beta = 2$.

Tables 6, 7 and 8 present the resulting functions $f^3_{OneEdge}$, f^3_{Poly} and f^5_{Poly}, respectively. The Boltzmann distributions with parameter $\beta = 2$ are polytree distributions satisfying the structural and parametric constraints given above. The reader can easily check this by computing their Boltzmann distributions and then computing the mutual information values.

Table 6. OneEdge function

$x_3x_2x_1$	$f^3_{OneEdge}(\mathbf{x})$	$x_3x_2x_1$	$f^3_{OneEdge}(\mathbf{x})$
000	1.042	100	−0.083
001	−0.736	101	0.092
010	0.357	110	−0.768
011	−1.421	111	−0.592

Table 7. FirstPolytree3 function

$x_3x_2x_1$	$f^3_{Poly}(\mathbf{x})$	$x_3x_2x_1$	$f^3_{Poly}(\mathbf{x})$
000	−1.186	100	−4.391
001	1.074	101	−1.122
010	0.469	110	−0.083
011	0.096	111	0.553

Investigating the Polytree Functions

Figure 5 presents the Boltzmann conditional curves and the curve $G(1,2,3)$ for the FirstPolytree3 function. Note that the curves $I(1,3|2)$ and $G(1,2,3)$ coincide

Table 8. FirstPolytree5 function ($\mathbf{x} = (x_5, x_4, x_3, x_2, x_1)$)

$\mathbf{x}$	$f^5_{Poly}(\mathbf{x})$	$\mathbf{x}$	$f^5_{Poly}(\mathbf{x})$	$\mathbf{x}$	$f^5_{Poly}(\mathbf{x})$	$\mathbf{x}$	$f^5_{Poly}(\mathbf{x})$
00000	–1.141	01000	–0.753	10000	–3.527	11000	–6.664
00001	1.334	01001	1.723	10001	–1.051	11001	4.189
00010	–5.353	01010	–4.964	10010	7.738	11010	–10.876
00011	–1.700	01011	–1.311	10011	–4.085	11011	–7.223
00100	0.063	01100	1.454	10100	1.002	11100	–1.133
00101	–0.815	01101	0.576	10101	0.124	11101	–2.011
00110	–0.952	01110	0.439	10110	–0.013	11110	–2.148
00111	–0.652	01111	0.739	10111	0.286	11111	–1.849

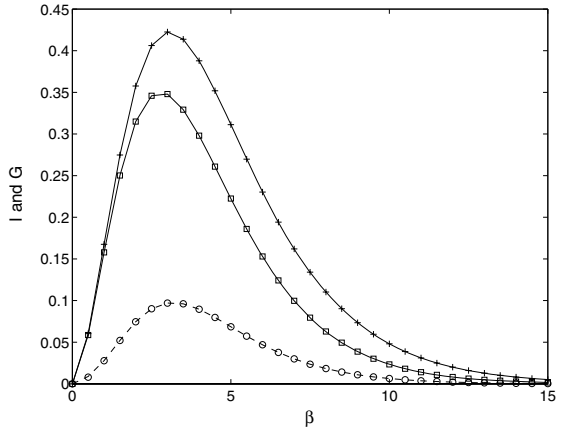

Fig. 5. Boltzmann information curves for the FirstPolytree3 function: (*plus*) $I(2,3|1)$, (*square*) $I(1,2|3)$, (*solid line*) and (*circle*) $G(1,2,3)$. Note that the last two curves coincide at the chosen scale

at the chosen scale. This means that the marginal curve $I(1,3)$ is close to zero. The actual values are below 10^{-3}, which amounts to independence for sample sizes below 2000 configurations. The other two marginal dependencies are quite strong for $\beta = 2$ (the value used in the design of the function). As far as $G(1,2,3)$ is always positive we conclude that for any selection pressure we have more evidence to decide against conditional independence than against marginal independence. Note that the conditional interval $[\beta^c_{min}, \beta^c_{max}]$ for $I(1,3|2)$ is completely included in the other two conditional intervals for any sample size. Note that in contrast with the Deceptive3, in this function the value β^G_z is not inside the interval $[\beta^c_{min}, \beta^c_{max}]$.

Figure 6 presents the conditional and marginal Boltzmann curves for the OneEdge function. For all β, the values $I(1,3)$ and $I(1,3|2)$ are very close; their difference, $G(1,2,3)$, is less than 10^{-8} and negative. The curves $I(1,2|3)$ and $I(2,3|1)$ are below 10^{-9}, which implies independence.

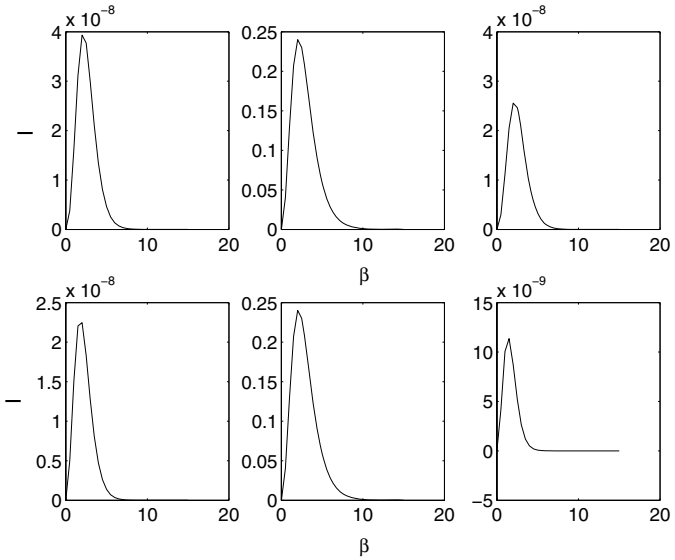

Fig. 6. Boltzmann information curves for the OneEdge function. The second row, from *left* to *right*, contains the conditional curves $I\,(\,1,2|\,3)$, $I\,(\,1,3|\,2)$ and $I\,(\,2,3|\,1)$. The upper row contains the corresponding marginal curves

In what follows, we use the BFDA to investigate two separable functions of 30 and 60 variables. The functions are formed by concatenating either the function $f^3_{OneEdge}$ or the function f^3_{Poly}.

By just looking at Tables 1, 6 and 7 it is difficult to draw any conclusion about what is the best factorization and which is the more difficult function for the BFDA. Following the theorem 2 the choice would be the complete model, which was shown to be the best factorization for the Deceptive3. However, the simulations of this section show that this is not the case for the other functions.

Table 9 presents the results of running the BFDA with the population size set to 120 for the FirstPolytree3. This time the factorization 12-32 is the clear winner. The convergence is almost twice as fast and its success rate is twice as high, in the factorization 12-32, as in the complete model. Similarly, the number of function evaluations is much bigger if the complete factorization is used. Therefore, we conclude

Table 9. BFDA runs with the FirstPolytree3. Setting: $N = 120$

	12-32				12-13-32			
n	$\%S$	G_c	β_{min}	β_{max}	$\%S$	G_c	β_{min}	β_{max}
30	95	13.32	0.197	4.927	55	22.29	0.139	7.954
60	85	13.49	0.196	5.526	38	21.95	0.139	8.715

that the assumption making the variables 1 and 3 marginally dependent is wrong. This is what we expected from our design decisions.

Regarding the Boltzmann curves the important observation is that the runs occur within the most inner interval $[\beta^c_{min}, \beta^c_{max}]$. Moreover, the better the conditions for the optimization are, the smaller the value of β_{max}. For example, for a fixed model, the smallest problem converges with the smallest β_{max}. Alternatively, if the size of the problem is fixed, then the best model has a smaller β_{max}. The same is observed in the simulations with the OneEdge. Table 10 presents the results.

Table 10. BFDA runs with the OneEdge. Setting: $N = 120$

	12-32			12-13-32			13		
n	$\%S$	G_c	β_{max}	$\%S$	G_c	β_{max}	$\%S$	G_c	β_{max}
30	94	11.64	4.724	98	9.36	3.917	100	9.25	3.867
60	39	20.82	7.026	75	17.067	6.151	98	17.03	6.045

For the OneEdge function three models are investigated. The model 13 – the one that is used in the design of the function – is the best. For example, compare the success rate of the complete model and the best model for 60 variables. Note that the convergence time is the same. In the model 12-32 the variables 1 and 3 are independent, which explains its poor performance.

We also have investigated the functions with the FDA. Besides the separable problem, in the simulations an overlapped additive function have been included. The overlapped case is constructed as follows: the last variable of a sub-set is also the first variable of the next sub-set in the additive decomposition. We use the letter O to denote this case. For example, contrast O-12-32 with 12-32.

Tables 11 and 12 present the numerical results. The factorizations 12-32 and O-12-32 are the best for the functions f^3_{Poly}. Similarly, the models 13 and O-13 perform better for the function $f^3_{OneEdge}$. Both the separable and the overlapped complete models do not scale well. For example, compare the success rates for the overlapped case of the OneEdge function.

Table 11. FDA runs with the FirstPolytree3

	N	$\%S$	G_c	$\%S$	G_c	$\%S$	G_c
		$n = 30$		$n = 60$		$n = 90$	
12-13-32	120	92	5.39	42	9.10	6	12.33
12-32	120	93	5.36	67	9.46	27	13.04
		$n = 31$		$n = 61$		$n = 91$	
O-12-13-32	200	83	6.81	25	11.04	4	14.00
O-12-32	200	94	6.59	63	11.28	20	14.90

Table 12. FDA and the OneEdge function

	N	$\%S$	G_c	$\%S$	G_c	$\%S$	G_c
		$n = 30$		$n = 60$		$n = 90$	
12-13-32	60	54	5.12	6	8.33	0	–
13	60	95	4.53	65	8.18	28	10.71
		$n = 31$		$n = 61$		$n = 91$	
O-12-13-32	100	71	5.57	20	10.00	2	12.50
O-13	100	100	5.20	81	9.03	57	12.52

We summarize the results as follows. The behaviour of the polytree functions investigated in this section, agrees with our design expectations. On the other hand, a clear correspondence between what happened in the simulations and the Boltzmann curves was observed. We take this as a sort of validation of both the usefulness of the analysis and design method introduced in this chapter.

4.3 Designing Random Class of Functions

In the previous section, we followed the common practice of concatenating low order functions to form larger additively decomposable functions. However, it would be useful if we could design a complete additive function with a given structure without resorting to the trick of concatenating small sub-functions. Moreover, it would be even more useful to design random class of functions, instead of isolated functions. To accomplish this task our method has to be extended.

In this section, we restrict ourselves to the design of the random class of binary polytree functions. This will provide the reader with general ideas and guidelines that might be helpful to undertake other design efforts.

Sampling the Structural Constraints

The first step is the generation of a random polytree graph. As was explained in Sect. 4.2, it is the structural constraint.

There exist simple methods for generating random graphs. Any of these algorithms together with a rejection sampling technique to reject graphs with directed cycles and undirected cycles, will do the job. At this stage the method outputs the graph, its junction tree and two lists, L_1 and L_2. If a pair (i, j) belongs to the first list, both i and j are parents of the same node and therefore, $I(X_i, X_j) < I_t$. On the other hand, the second list contains a pair (i, j), if and only if, j is the parent of i. In this case, $I(X_i, X_j) > I_t$. For each pair (i, j) in the lists, we sample a bivariate marginal distribution $p(x_i, x_j)$, that obeys the corresponding mutual information constraint. This non-trivial task is discussed in what follows.

Sampling Bivariate Marginals Under Independence

The problem is related to the evaluation of the exact sampling distributions of the cell counts in applied multivariate analysis [35]. Therefore, we set $n_i = Np(x_i)$, $n_j = Np(x_j)$ and $n_{ij} = Np(x_i, x_j)$, where N is the sample size.

Let assume Poisson, multinomial or independent multinomial sampling. Under the null hypothesis of independence, the conditional distribution of n_{ij} given the observed marginal counts n_i and n_j is the central hyper-geometric distribution, which is known exactly. The random scalar variable N_{ij} is given by

$$N_{ij} \sim \frac{\binom{n_i}{n_{ij}}\binom{N-n_i}{n_j-n_{ij}}}{\binom{N}{n_j}} \tag{17}$$

Let n_i and n_j be given. Then, for any pair (i, j) in the list L_1 we generate the bivariate marginal $p(x_i, x_j)$ by sampling n_{ij} from (17), and then substituting $p_{ij} = n_{ij}/N$, p_i and p_j in (16).

It is worth noting, that the method can be extended to deal with variables of cardinality greater than two [35].

Sampling Correlated Bivariate Marginals

For the computation of the marginals associated to the list L_2, the solution comes from the exact non-null distribution theory [1].

Let assume multinomial sampling and let θ be the odds ratio [35]. Conditional on n_i and n_j, the distribution of n_{ij} depend only on θ, and is given by

$$N_{ij} \sim \frac{\binom{n_i}{n_{ij}}\binom{N-n_i}{n_j-n_{ij}}\theta^{n_{ij}}}{\sum_{u=m}^{M}\binom{n_i}{u}\binom{N-n_i}{n_j-u}\theta^{u}} \tag{18}$$

where $m = max(0, n_i + n_j - n)$ and $M = min(n_i, n_j)$.

As far as the constraints are specified using the mutual information, one could try a reparameterization of (18). However, we use directly the odds ratio, which obeys $0 \leq \theta < \infty$. Values of θ farther from 1.0 in a given directions represent higher values of mutual information. Moreover, if $\theta_1 = 1/\theta_2$, then both θ_1 and θ_2 represent the same level of dependence.

Let n_i and n_j be given. For any pair (i, j) in the list L_2, we compute θ according to the mutual information $I(X_i, X_j)$. Then, n_{ij} is sampled from (18) and $p(x_i, x_j)$ is obtained from (16).

Once all the bivariate marginals have been computed we are ready to build the maximum-entropy junction tree. Afterwards, we obtain an instance of the random class by substituting in (13) the distribution and the given β.

How to Test EDA Algorithms

The procedure introduced in the previous sections allows us to define a large class of functions: the class of random Boltzmann polytree functions (RBPF). We denote the class by $RBPF\,(n,\,K,\,\beta)$, where K is the maximum number of parents in the polytree. Note that Z is not included as a parameter because it is chosen automatically in such a way to make the function non-negative for any configuration $\boldsymbol{x}$.

Testing evolutionary algorithms have been recognized as a major problem in current EDA research [23]. We believe that the approach presented in this chapter will improve the ability of the research community to test and compare EDA algorithms. Moreover, the design of random classes of Boltzmann functions should help to understand the complex mechanisms involved in EDA optimization, because now we have an explicit control of the dependencies presented in the functions. We are confident that others random classes can be designed using similar ideas to the ones presented in this chapter.

Within our framework, any optimization algorithm should be tested in samples of carefully designed random classes of functions. In other words, instead of using a single function and running the algorithm 100 times, we prefer to use once 100 different functions sampled from the same random class.

5 Learning Low Cost Max-Entropy Distributions

A critical problem of learning search distributions in EDAs is the sample complexity, which is related with the number of functions evaluations. One important challenge of an evolutionary algorithm is the reduction of the number of evaluations, while the effectiveness and efficiency of the search is preserved. In this section we will use the concept of entropy to achieve this goal. Our idea is simple: the true search distribution is substituted by an approximation, which can be reliably computed with less population size.

The following definitions will help to clarify our ideas.

Algorithm 4 Maximum-entropy EDA

Step 1	Set $t \leftarrow 1$. Generate $N \gg 0$ points randomly.
Step 2	Select M points according to a selection method.
Step 3	Find a suitable R and learn a $\mathbb{R}_{p^s(x),R}$ from the selected set.
Step 4	Compute the maximum entropy distribution $p_R^s\,(X)$.
Step 5	Sample N new points according to the distribution $p\,(x, t+1) = p_R^s(x_1, \ldots, x_n)$
Step 6	Set $t \leftarrow t+1$. If termination criteria are not met, go to step 2.

Definition 5 *Let $p(X_1, \ldots, X_n)$ be the factorization of the selected set. We say that $p(X_1, \ldots, X_n)$ is a true search distribution if it was computed from a data set, whose size allows reliable estimates of the factors' probabilities.*

Definition 6 *Let $\boldsymbol{X}$ be a random vector of dimension n and $R = \{r_1, \ldots, r_m\}$ be a set of index-sets. A restriction of a joint distribution $p(\boldsymbol{x})$ is a set of marginal distributions*

$$\mathbb{R}_{p(\mathbf{X}),R} = \{p(X_{r_1}), \ldots, p(X_{r_m})\}$$

of $p(\boldsymbol{x})$, such that the following holds:

1. $\forall i, 1 \leq i \leq m$, $r_i \subset \{1, \ldots, n\}$ *and* $r_i \neq \emptyset$
2. $\forall i, j, 1 \leq i, j \leq m$, $r_i \nsubseteq r_j$

Definition 7 *Let $\mathbb{R}_{p(\mathbf{X}),R}$ be a restriction of $p(\boldsymbol{x})$, then $p_R(\boldsymbol{x})$ is defined as the maximum-entropy distribution that fulfills the constraints $\mathbb{R}_{p(\mathbf{X}),R}$.*

Using the above definitions, we introduce an EDA that uses the MEP (see algorithm 4). We have called it maximum-entropy EDA (meEDA).

Step 2 is a critical point of the meEDA algorithm because the algorithm has to choose a suitable restriction set. It is an open problem how to identify good restrictions of the search distributions. For example, besides the primary goal of getting a sampling distribution with less cost than the true distribution, there could be other reasons that determine a good choice of the restriction set. On the other hand, an efficient procedure for the computation of the maximum-entropy distribution exists only if the structure of the restriction set satisfies certain constraints. The next section presents an algorithm EDA where the maximum-entropy distribution can be computed efficiently.

5.1 Extending PADA2 with Maximum-Entropy

The polytree functions designed in Sect. 4 have a common property: their search distributions are single-connected. In this section we modify PADA2 – an algorithm specially designed to deal with single connected Bayesian networks – to transform it into a meEDA.

The polytree approximation distribution algorithm (PADA) [33,34] was designed to deal with the whole class of single-connected Bayesian networks; also called the polytree class. It uses first, second and third order marginals to recover polytrees from data. In this work we will use PADA2 [33] – variant of PADA, which learns only first and second order marginals distributions. PADA2 is inspired by an algorithm proposed by Rebane and Pearl [30]. We shortly outline the basic ideas behind the algorithm.

A polytree with n variables has a maximum of $n - 1$ arcs, otherwise it would not be single connected. PADA2 chooses the edges that have the largest values of the magnitude $H(X) + H(Y) - H(X, Y)$, which is also called mutual information [3]. The selection of the edges is done by a greedy maximum weight spanning

tree algorithm. These edges form the so-called skeleton (the underlying undirected graph).

After the construction of the skeleton is done, a procedure tries to orient the edges by using the following scheme: if $X - Z - Y \in skeleton$, then whenever $H(X) + H(Y) = H(X, Y)$ holds statistically it orients the edges to Z. In this case it is said that Z is a head to head connection. The edges that were not oriented after the above test are directed at random without introducing new head to head connections.

Both during learning and sampling, EDAs that learn general Bayesian networks need a population size, which is exponential in the number of parents. This is important to get reliable estimates of the conditional probabilities. However, although PADA2 only learns first and second order marginals, it has to deal with the same exponential problem in the sampling step, i.e. what is gained in learning is lost in the sampling.

To transform PADA2 into mePADA2 we must define the polytree's restriction set, i.e. all bivariate marginals that belong to the skeleton and the bivariate marginals defined for each pair parent-child. Note that this restriction set was used as a parametric constraint in Sect. 4.2. The next step consists in computing the higher order marginals as the maximum-entropy distributions that obey the given second order marginals. Consistency is guaranteed by propagating across the junction tree associated to the polytree as was explained in Sect. 2.4.

Now we present some numerical results to support the theoretical claims. We use two separable ADF functions, which are based on the Deceptive3 and FirstPolytree5. Although the structure of the Deceptive3 function is not single-connected, PADA2 tries to build the better single-connected approximation it can. It is remarkable that the method still produces very good results. We recall that the basic claim of our research is that the maximum-entropy distribution, which can be computed with a smaller population size than the true search distribution, is suitable for sampling. Moreover, sometimes it gives better results than the true distribution.

The algorithms are run until a maximum of 20 generations with a truncation selection of 0.3 and without elitism. Each experiment is repeated 100 times. The problem sizes were set to 21 variables for the Deceptive3 and 20 variables for the FirstPolytree5.

As can be seen from Table 13 the improvement of mePADA2 is enormous as compared to PADA2. For the f^5_{Poly}, the superiority of mePADA is more evident; not only it scales much better than PADA2, but the convergence time is drastically reduced. It is also remarkable that the number of generations until success always stays the same or even improves. It has also stabilized as can be seen from the decrease in the standard deviation.

The idea of improving the performance of EDAs by constructing maximum-entropy approximations of the search distributions was first introduced in [29]. Later it was further developed in [23] for multi-connected networks.

Table 13. PADA2 vs. mePADA2 with f_{dec3} and f^5_{Poly}

	f_{dec3}				f^5_{Poly}			
	PADA2		mePADA2		PADA2		mePADA2	
N	$\%S$	G_c	$\%S$	G_c	$\%S$	G_c	$\%S$	G_c
200	0	–	2	8.5 ± 0.7	25	10.1 ± 2.1	59	5.1 ± 1.1
600	8	9.7 ± 1.5	69	7.4 ± 1.1	50	10.4 ± 2.6	100	3.9 ± 0.7
800	10	8.7 ± 3.2	90	7.0 ± 1.2	54	10.6 ± 2.3	100	3.7 ± 0.6
5000	92	7.2 ± 1.2	100	5.8 ± 0.9	55	10.8 ± 1.5	100	2.9 ± 0.4

6 Entropy and Mutation

The last section of this chapter relates the concept of entropy toa powerful operator of evolutionary algorithms: mutation.

The mutation operator did not receive much attention during the early years of research in EDAs. It was believed to play no important role due the dramatic improvement in search efficiency achieved by EDAs, with regard to GAs. People profoundly believed that the success of EDAs is determined by the amount of knowledge it has about the search distributions, i.e. the best informed models were considered – and still are considered – the best models. Within this way of thinking there was little space for mutations. However, after some years of hard work, researchers have come to the conclusion that mutation is also a powerful operator within EDAs. Therefore, new and original developments are needed in the field to deal with this issue.

To begin with, we must draw the reader attention to the fundamental shift in the interpretation of mutation: EDAs have to approach mutation from a distribution perspective, in contrast with the genotype perspective of GAs. While a GA mutates single individuals, an EDA must mutate distributions. We have developed an approach to fulfill this requirement.

6.1 How do we Measure the Effect of the Mutation?

A major problem with the mutation operator in evolutionary algorithms, is the lack of a comprehensible, uniform and standard mechanism for measuring its impact in the evolution. There are almost as many mutation operators as problems, and only few of them are problem-independent. The common way of assessing the amount of mutation considers the probability or frequency of application of the operator, i.e. there are no measurements units for mutation. The obvious drawback of this approach is that it is difficult to compare the impact of different operators or the effect of the same operator in different situations.

Our approach to mutation solves the above-mentioned problems. It has a distribution perspective, is problem-independent, has measurements units, and its impact in different scenarios can be easily compared. It is based on the concept of entropy.

The relation between entropy and mutation is quite intuitive: when a random variable is mutated a certain degree of randomness is added to it. In others words,

mutation increases the level of uncertainty we have about the exact value of a random variable. Therefore, it seems reasonable to measure the amount of mutation applied to a variable as the increase of its entropy. This connection was first made in [33].

Linking the concepts of mutation and entropy has some important advantages:

- Entropy is a well understood information-theoretic concept, which encapsulates the notion of randomness and uncertainty.
- It connects the mutation to other fundamental concepts like mutual information and relative entropy.
- Mutation also gets from entropy measurements units: bits or nats, instead of using the popular, but less clear notion of probability of application of the mutation operator.

6.2 From Bit-flip to Entropic Mutation

In this section, we shortly discuss two important mutation schemes that precede our proposal. One was introduced in GAs, and the other was recently introduced in EDAs. The observation of the entropic variations produced by these schemes was a major motivation for our work.

Bit-flip Mutation

The classical GA mutation operator for binary problems is a bit-flip (BF) operation that is applied to each gene with a certain given probability μ [8]. The next lemma relates BF-mutation with the univariate probabilities.

Lemma 4 *For binary variables, BF mutation changes the probability according to*

$$p_f - p_i = \mu (1 - 2p_i)$$

where p_f is the probability after mutation and p_i is the probability before mutation.

Proof 2 *Let the probability of a bit flip be μ, and p_i be the probability of a gene being 1 before mutation. As these events are independent, we can write for the probability p_f of the gene being 1 after mutation*

$$p_f = p_i (1 - \mu) + (1 - p_i) \mu = p_i (1 - 2\mu) + \mu \tag{19}$$

and from this we get

$$p_f - p_i = \mu (1 - 2p_i) \quad \square \tag{20}$$

If we compute the entropy of a variable before and after the BF-mutation, $H(p_i)$ and $H(p_f)$ respectively, then we can measure the increase of entropy produced by this operation

$$\delta H = H(p_f) - H(p_i)$$

Figure 7 shows δH curves for six different values of the probability of mutation μ. Note that δH is nonlinear for small values of the initial entropy, $H(p_i)$, and small μ. However, for large values of $H(p_i)$ the curves approach a linear function. Moreover, for large μ the curves approach lines. The limit case, $\mu = 0.5$, defines a random walk: for any p_i the probability after mutation is 0.5.

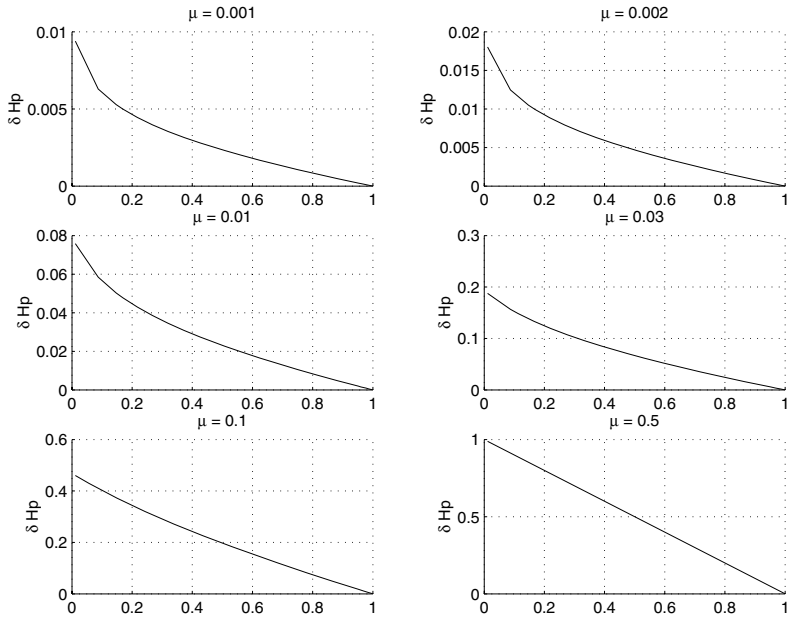

Fig. 7. Entropic curves δH vs H for bit-flip mutation

Prior Mutation

Prior mutation was introduced in [20]. It uses the concept of Bayesian prior, which assumes that the probability of an event has an a priori known distribution. Usually, for binomial variables, the family of Dirichlet distributions plays the role of prior distributions.

In an EDA with prior mutation, the univariate probabilities are not approximated by the maximum likelihood estimates m/N (m is the number of 1 in N cases). Instead the approximation $(m + r) \,/\, (N + 2r)$ is used, where r is the hyper-parameter of the Dirichlet distribution. Prior mutation is linked to bit-flip mutation. The following theorem was proved in [20].

Theorem 5 *For binary variables, a Bayesian prior with parameter r corresponds to mutation rate $\mu = r/\,(N + 2r)$*

Therefore, for the univariate case bit-flip mutation amounts to prior mutation, and as a consequence, they have the same entropic curves.

6.3 Entropic Mutation

The linear properties of both the bit-flip and prior entropic curves, have suggested that we consider a mutation scheme where δH changes linearly. As a result we have come out with a novel mutation scheme that we have called linear entropic mutation (LEM). In this chapter, we just outline the general ideas.

The Univariate Case

In this section, we discuss the entropic mutation of a binary scalar random variable.

Definition 8 *Let X be a random scalar variable with entropy $H(X)$. We say that to the variable X has been applied the univariate entropic mutation $\delta H(X)$, if after mutation the entropy of the variable is given by*

$$H_m(X) = H(X) + \delta H(X)$$

This is a general definition, which can be applied as well to discrete and continuous random vector variables. Besides the univariate mutation, we have defined the conditional and the joint entropic mutations. However, these cases are beyond the scope of this work.

Definition 9 (Full mutation) *Let X be a binary random scalar variable with entropy $H(X)$. We say that $\delta H(X)$ is a full (or complete) mutation of the variable X if*

$$\delta H(X) = 1 - H(X)$$

Full mutation amounts to bit-flip mutation with $\mu = 0.5$. In this case, a variable gets an increase of entropy equal to what it needs to reach its maximum entropy. This kind of mutation has little use in an optimization context. At this point it is natural to ask ourselves when and how much the entropy of a given variable should be changed. A simple answer based on common sense says that one would like to change a variable if it has low entropy. Indeed, it does not make any sense to mutate a variable with probability $p = 0.5$ ($H(p) = 1$).

Figure 8 shows the line of full mutation as a function of the initial entropy, together with two others linear functions of H. The slopes of the lines are the mutation intensities, α. The following definition formalizes this idea.

Definition 10 *Let X be a random scalar variable with entropy $H(X)$. We say that to the variable X has been applied the linear entropic mutation $\delta H(X)$ with parameter α if after mutation it has entropy $H_\alpha(X)$ and the following holds*

$$\delta H(X) = (1 - H(X))\,\alpha \;\Leftrightarrow\; H_\alpha(X) = (1-\alpha)\,H(X) + \alpha \tag{21}$$

Note in Fig. 8, that α is the ordinate for $H(X) = 0$. So, it is bounded by $\alpha = 1$ (full mutation) and $\alpha = 0$ (no mutation).

The mutation intensity α controls the strength of the mutation, i.e. how much the entropy of a variable is changed. In an optimization scenario α might change across time; thus, the general form of the mutation intensity is $\alpha(t)$.

The computation of a LEM-mutation of $p(X)$ is accomplished in two steps. Firstly, $H_\alpha(X)$ is computed according to (21), and then the new probability distribution $p_\alpha(X)$ is obtained from $H_\alpha(X)$. However, as the entropy of binary variables is symmetric – each entropy value is mapped to exactly two probability values – we introduce the following definition to resolve the ambiguity.

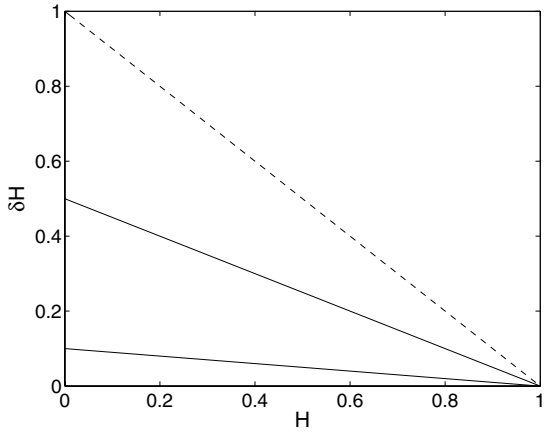

Fig. 8. LEM-mutation of a random binary variable. From *top* to *bottom* the slopes (α) are equal to 1, 0.5 and 0.1

Definition 11 *(Inverse function of $H(X)$). Let $H^{(-1)} : [0,1] \times [0,1] \rightarrow [0,1]$ be a function such that for any real numbers p and q, with $0 \leq p, q \leq 1$,*

$$p = H^{(-1)}\left(H\left(p\right), q\right) \Rightarrow (2p-1)(2q-1) \geq 0$$

Definition 11 says that for a given pair $\langle h, q \rangle$ (with $h = H(p)$), the function $H^{(-1)}(h, q)$ returns a probability p, such that both p and q lie together in the interval $[0, 0.5)$ or in $[0.5, 1]$. This definition is useful because for any p, p_α lie in the same half of $[0, 1]$ as p. Finally we can write the expression for p_α as follows:

$$p_\alpha = H^{(-1)}\left(\left(1-\alpha\right) H\left(X\right) + \alpha, p\right) \tag{22}$$

A Note on the Multivariate Case

The multivariate LEM is more difficult than the univariate case, even for binary variables. Here we just give a necessary condition. Other results for multidimensional distributions will be published elsewhere soon.

Definition 12 *Let $p(x_1, x_2, \ldots, x_n)$ and $p_\alpha(x_1, x_2, \ldots, x_n)$ denote a binary joint probability mass and its LEM-mutation with mutation intensity α. If $H(X)$ and $H_\alpha(X)$ are their respective entropy values, then the following holds:*

$$\delta H\left(X\right) = \left(n - H\left(X\right)\right)\alpha \quad and \quad H_\alpha\left(X\right) = \left(1-\alpha\right) H\left(X\right) + n\alpha \tag{23}$$

Table 14 shows the set of joint probability distributions $p_\alpha(x_1, x_2, \ldots, x_n)$ that were used to compute the family of functions $f_{dec3}(\alpha)$ in Sect. 3.2. Note in the second column that the entropy values obey the relation (23), where $H(X)$ is the entropy of the first row and $n = 3$. However, computing $p_\alpha(x_1, x_2, \ldots, x_n)$ from

Table 14. LEM mutation and the family $f_{dec3}(\alpha)$.

		$x_3x_2x_1$							
α	H_α	000	001	010	011	100	101	110	111
0.00	1.74	0.204	0.071	0.071	0.000	0.071	0.000	0.000	0.582
0.05	1.80	0.202	0.075	0.074	0.002	0.073	0.001	0.001	0.572
0.20	1.99	0.197	0.084	0.082	0.012	0.078	0.006	0.006	0.536
0.40	2.24	0.189	0.095	0.091	0.028	0.086	0.017	0.015	0.479
0.50	2.37	0.184	0.100	0.096	0.038	0.090	0.024	0.022	0.446

$p(x_1, x_2, \ldots, x_n)$ and α is not a trivial task and is beyond the scope of this chapter. Here we just present a special case where we easily can show a distribution that fulfill (23). The following theorem gives the details.

Theorem 6 *Let $p(\boldsymbol{x})$ be the joint probability mass of a set of independent random variables $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$. If*

$$p_\alpha(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} H^{(-1)}\left((1-\alpha) H(X_i) + \alpha, p_i\right) \tag{24}$$

then

$$H(p_\alpha(X)) = (1-\alpha) H(X) + n\alpha \tag{25}$$

Proof 3 *The lemma follows from theorem 3 and the linearity of the LEM-mutation. We rewrite the right term of (25)*

$$\begin{aligned}(1-\alpha) H(X) + n\alpha &= (1-\alpha) \sum_{i=1}^{n} H(X_i) + n\alpha \\ &= \sum_{i=1}^{n} \left((1-\alpha) H(X_i) + \alpha\right) \\ &= \sum_{i=1}^{n} H_\alpha(X_i)\end{aligned}$$

From (22) and (24) follows that $p_\alpha(x_1, x_2, \ldots, x_n)$ is the distribution of independence with univariate probabilities $p_\alpha(x_i)$. Therefore, the left term of (25) is given by

$$H_\alpha(X) = \sum_{i=1}^{n} H_\alpha(X_i)$$

This completes the proof. □

Closely related to the above theorem is the following general result.

Theorem 7 *Let $p(\boldsymbol{x})$ be any joint probability mass of a set of random variables $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$, then*

$$H_\alpha(\boldsymbol{X}) \leq \sum_{i=1}^{n} H(X_i)_\alpha$$

with equality if and only if the variables are independent.

Proof 4 *The proof follows immediately from theorem 3 and the linearity of LEM.*

6.4 Testing the UMDA with LEM

On the basis of theorem 6 we can add LEM-mutation to the UMDA [22], which is a FDA with full factorization. The mutation operation is inserted before the sampling step, i.e. the distribution of the selected set is mutated.

Mutation is a powerful mechanism that does not only makes the optimization algorithm more robust and effective, but also might reduce its population size requirements. The search using mutation takes more time and less population size than without it. With regard to the number of function evaluations these are conflicting factors. We just illustrate this issue with an example.

We run the UMDA with the OneMax function, which outputs the number of variables set to one in its input. The UMDA solves this function (with high probability) if the population size is close to the problem size [22]. For the experiment we have chosen a population size that is half the problem size ($N = 30$, $n = 60$), which implies a dramatic reduction of the success rate. Figure 9 shows the success rate and the number of function evaluations as a function of α. Note that for $\alpha = 0$ (no mutation), the success rate is $\approx 18\%$ (out from 100 runs). However, for $\alpha \in [0.06,\ 0.2]$ the success rate is above 90%.

Note that for $\alpha \in [0.08,\ 0.12]$, the number of functions evaluations reaches the minimum. This value is less than the minimum population size ($N \approx 55$) that is needed to have a success rate above 90% without mutation. This value is shown as a threshold dot line in the figure. We conclude that the gain due to the population size is not eliminated by the increment in the convergence time.

In summary, with small populations and low or high mutation rates the algorithm performs badly. However, there exists a window $[\alpha_{min}, \alpha_{max}]$ where the success rate is high, that might contain another window where the algorithm reaches the minimum possible number of functions evaluations.

7 Conclusions

This chapter has highlighted several important issues regarding the relation between the concept of entropy and EDAs.

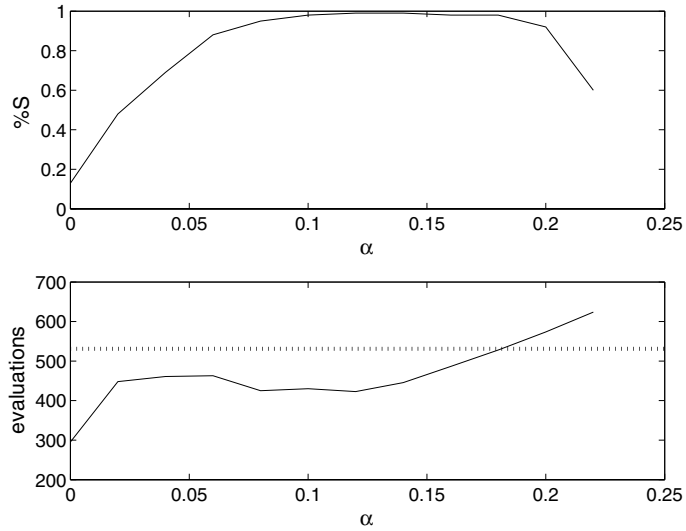

Fig. 9. UMDA and the Onemax function. Success rate and number of function evaluations vs. the mutation intensity α. Setting $N = 30$, $n = 60$

We have introduced a tool to investigate the levels of interactions of the variables under Boltzmann selection: the Boltzmann mutual information curves. It constitutes the corner stone of a method for analysing the complexity of functions for EDAs.

Closely related to the analysis method, is our approach to the design of single and random classes of benchmark functions. We are confident that the use of random classes of Boltzmann functions improves our ability to test EDA algorithms in a more scientific way giving to the benchmark approach a sound theoretical basis. The point is that our method offers an explicit control of the dependencies presented in the functions.

We have used the maximum entropy principle as a key element of the design method and also to build low cost approximations of search distributions that obey a given collection of constraints. We believe that the building of low cost distributions may have tremendous impact on real-world applications of EDAs, so it deserves the special attention of the research community.

Finally, a short introduction to a new scheme of mutation, which is based on the concept of entropy was presented. The linear entropic mutation is a natural operator for EDAs because it mutates distributions instead of single individuals. From a theoretical point of view it opens new exciting directions of research toward a better understanding of the complex dynamics describing the golden equilibrium between exploration and exploitation.

Acknowledgments

We would like to thank Rolando Biscay, Roberto Santana, Omar Ochoa, the anonymous referees and the editors, for their useful comments, fruitful discussions, and support. We specially thank Heinz Mühlenbein for his oustanding contribution to our research work.

References

1. A. Agresti. *Categorical Data Analysis*. John Wiley and Sons, 1990.
2. I. Beinlich, H. R. Chavez, and G. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Artificial Intelligence in Medical Care*, pp. 247–256, 1989.
3. T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Jonh Wiley and Sons, New York, 1991.
4. I. Csiszar. I-Divergence geometry of probability distributions and minimization problems. *Annals of Probability*, 3:146–158, 1975.
5. L. M. de Campos. Independency relationship and learning algorithms for singly connected networks. *Experimental and Theoretical Artificial Intelligence*, 10:511–549, 1998.
6. K. Deb, J. Horn, and D. E. Goldberg. Multimodal deceptive function. *Complex Systems*, 7:131–153, 1993.
7. D. E. Goldberg. Simple genetic algorithms and the minimal deceptive problem. In Lawerance Davis, editor, *Genetic Algorithms and Simulated Annealing*, pp. 74–88. Pitman, 1987.
8. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
9. D. E. Goldberg, K. Deb, and J. Horns. Massive multimodality, deception, and genetic algorithms. *Lecture Notes in Computer Sciences, Parallel Problem Solving from Nature PPSN II*, pp. 37–46, 1992.
10. C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *Journal of Approximate Reasoning*, 15(3):225–263, 1996.
11. C. T. Ireland and S. Kullback. Contingency tables with given marginals. *Biometrika*, 55:179–188, 1968.
12. E. T. Jaynes. Information theory and statistical mechanics. *Physics Review*, 6:620–643, 1957.
13. E. T. Jaynes. Where do we stand on maximum entropy? In R. D. Levine and M. Tribus, editors, *The Maximum Entropy Formalism*. MIT Press, 1978.
14. F.V. Jensen and F. Jensen. Optimal junction trees. In *10th Conference on Uncertainty in Artificial Intelligence*, pp. 360–366, Seattle, 1994.
15. R. Jiroušek and S. Přeučil. On the effective implementation of the iterative proportional fitting procedure. *Computational Statistics and Data Analysis*, 19:177–189, 1995.
16. S. L. Lauritzen. *Graphical Models*. Oxford Press, 1996.
17. F. Leisch, A. Weingessel, and K. Hornik. On the Generation of Correlated Artificial Binary Data. Technical Report 13, Viena University of Economics and Bussines Administration, Viena, 1998.
18. P. M. Lewis. Approximating probability distributions to reduce storage requirements. *Information and Control*, 2:214–225, 1959.

19. T. Mahnig and H. Mühlenbein. Comparing the adaptive Boltzmann selection schedule SDS to truncation selection. In *Third International Symposium on Adaptive Systems ISAS 2001, Evolutionary Computation and Probabilistic Graphical Models*, pp. 121–128, La Habana, 2001.
20. T. Mahnig and H. Mühlenbein. Optimal mutation rate using Bayesian priors for estimation of distribution algorithms. *Lecture Notes in Computer Sciences*, 2264:33–48, 2001.
21. C. H. Meyer. *Korrektes Schließen bei Unvollständiger Information*. PhD thesis, Fernuniversität Hagen, 1998. In German.
22. H. Mühlenbein. The equation for the response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1998.
23. H. Mühlenbein and R. Höns. The estimation of distributions and the maximum entropy principle. *Evolutionary Computation*, 2004. To appear.
24. H. Mühlenbein and T. Mahnig. Evolutionary optimization and the estimation of search distributions. *Journal of Approximate Reasoning*, 31(3):157–192, 2002.
25. H. Mühlenbein, T. Mahnig, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):213–247, 1999.
26. H. Mühlenbein and G. Paas. From recombination of genes to the estimation of distributions I. Binary parameters. *Lecture Notes in Computer Sciences, Parallel Problem Solving from Nature PPSN IV*, 1141:178–187, 1996.
27. D. Nilsson. An efficient algorithm for finding the M most probable configuration in Bayesian networks. *Statistics and Computing*, 2:159–173, 1998.
28. D. Nilsson. The computation of moments of decomposable functions in probabilistics expert systems. In *Third International Symposium on Adaptive Systems ISAS 2001, Evolutionary Computation and Probabilistic Graphical Models*, pp. 116–120, La Habana, 2001.
29. A. Ochoa, R. Höns, M. Soto, and H. Müehlenbein. A maximum entropy approach to sampling in EDA – the single connected case. *Lecture Notes in Computer Sciences, 8th Iberoamerican Congress on Pattern Recognition CIARP 2003*, 2905:683–690, 2003.
30. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
31. J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2000.
32. C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.
33. M. Soto. *Un Estudio sobre los Algoritmos Evolutivos Basados en Redes Bayesianas Simplemente Conectadas y su Costo de Evaluación*. PhD thesis, Instituto de Cibernética, Matemática y Física, La Habana, 2003. In Spanish.
34. M. Soto and A. Ochoa. A factorized distribution algorithm based on polytrees. In *Congress on Evolutionary Computation CEC 2000*, pp. 232–237, California, 2000.
35. J. Whittaker. *Graphical Models in Applied Multivariate Statistics*. John Wiley and Sons, 1989.