1 Evolutionary Synthesis of Bayesian Networks for Optimization

Heinz Mühlenbein, Thilo Mahnig

We shortly review our theoretical analysis of genetic algorithms and provide some new results. The theory has lead to the design of three different algorithms, all based on probability distributions instead of recombination of strings. In order to be numerically tractable, the probability distribution has to be factored into a small number of factors. Each factor should depend on a small number of variables only. For certain applications the factorization can be explicitly determined. In general it has to be determined from the search points used for the optimization. Computing the factorization from the data leads to learning Bayesian networks. The problem of finding a minimal structure which explains the data is discussed in detail. It is shown that the Bayesian Information Criterion is a good score for this problem. The algorithms are extended to probabilistic prototype trees used for synthesizing programs.

1.1 Introduction

Simulating evolution as seen in nature has been identified as one of the key computing paradigms for the next decade. Today evolutionary algorithms have been successfully used in a number of applications. These include discrete and continuous optimization problems, synthesis of neural networks, synthesis of computer programs from examples (also called genetic programming) and even evolvable hardware. But in all application areas problems have been encountered where evolutionary algorithms performed badly. Therefore a mathematical theory of evolutionary algorithms is urgently needed. Theoretical research has evolved from two opposed end; from the theoretical approach there are theories emerging that are getting closer to practice; from the applied side ad hoc theories have arisen that often lack theoretical justification.

Evolutionary algorithms for optimization are the easiest for theoretical analysis. Here results from classical population genetics and statistics can be used. In our theoretical analysis we have approximated the dynamics of genetic algorithms by probability distributions. Subsequently this analysis has lead to a new algorithm called the *Univariate Marginal Distribution Algorithm* UMDA. It uses search distributions instead of the usual recombination/crossover of genetic strings [9].

The outline of the paper is as follows. First the theory to analyze genetic algorithms and UMDA is introduced. Then UMDA is extended to the *Factorized*

Distribution Algorithm FDA. It uses a general factorization of the probability distribution. FDA optimizes problems where UMDA and genetic algorithms perform badly [8]. FDA needs a valid factorization as input. For some problems this cannot be done. The factorization has to be determined from the data. This problem is solved by mapping probability distributions to Bayesian networks. To find a good Bayesian network structure explaining the data is called *learning*. The corresponding algorithm *Learning Factorized Distribution Algorithm* LFDA is discussed with several numerical examples.

We then investigate if the theory can be applied to other problems in evolutionary computation. The first example is synthesis of programs by a probabilistic tree. The second example is the synthesis of neural trees.

The paper summarizes popular approaches for synthesizing networks or programs. It analyzes the approaches with a common theoretical framework. In addition it formulates open research issues.

1.2 From Recombination to Distributions

For notational simplicity we restrict the discussion to binary variables $x_i \in \{0,1\}$. We use the following conventions. Capital letters X_i denote variables, small letters x_i assignments. Let $\mathbf{x} = (x_1, \dots, x_n)$ denote a binary vector. Let a function $f : \boldsymbol{X} \to R^{\geq 0}$ be given. We consider the optimization problem $\mathbf{x}_{opt} = \text{argmax} f(\mathbf{x})$.

Definition Let $p(\mathbf{x}, t)$ denote the probability of $\mathbf{x}$ in the population at generation t. Then $p(x_i, t) = \sum_{\mathbf{x}, X_i = x_i} p(\mathbf{x}, t)$ defines the univariate marginal distributions.

Mühlenbein [9] has shown that any genetic algorithm can be approximated by an algorithm using univariate marginal distributions only.

UMDA

- **STEP 0:** Set $t \Leftarrow 1$. Generate $N \gg 0$ points randomly.
- **STEP 1:** Select $M \leq N$ points according to a selection method. Compute the marginal frequencies $p_i^s(x_i, t)$ of the selected set.
- **STEP 2:** Generate N new points according to the distribution $p(\mathbf{x}, t+1) = \prod_{i=1}^{n} p_i^s(x_i, t)$. Set $t \Leftarrow t + 1$.
- **STEP 3:** If termination criteria are not met, go to STEP 1.

The simple genetic algorithm uses fitness proportionate selection [3]. In this case the probabilities of the selected points are given by $p^s(\mathbf{x},t) = p(\mathbf{x},t)f(\mathbf{x})/\bar{f}(t)$ where $\bar{f}(t) = \sum_{\mathbf{x}} p(\mathbf{x},t)f(\mathbf{x})$ denotes the average fitness of the population.

For theoretical analysis we consider $\bar{f}(t)$ as depending on $p(x_i,t)$. In order to emphasize this dependency we write

$$W(p(X_1 = 0,t), p(X1 = 1,t), \ldots, p(X_n = 1,t)) := \bar{f}(t) \tag{1.1}$$

For infinite populations the dynamics of UMDA leads to a deterministic difference equations for $p(x_i,t)$ [9].

$$p(x_i, t+1) = p(x_i,t)\frac{\bar{f}_i(t)}{W} \tag{1.2}$$

where $\bar{f}_i(t) = \sum_{\mathbf{x},X_i=x_i} f(\mathbf{x}) \prod_{j\neq i}^n p(x_j,t)$. These equations can be written as

$$p(x_i, t+1) = p(x_i,t)\frac{\frac{\partial W}{\partial p(x_i)}}{W(t)} \tag{1.3}$$

Equation 1.3 shows that UMDA performs a gradient ascent in the landscape given by W. Despite its simplicity UMDA can optimize difficult multi-modal functions. We take as first example the function BigJump. It is defined as follows, with $|\mathbf{x}|_1 = \sum x_i$ equal to the number of 1-bits:

$$\text{BigJump}(n,m,k,\mathbf{x}) := \begin{cases} |\mathbf{x}|_1 & 0 \le |\mathbf{x}|_1 \le n-m \\ 0 & n-m < |\mathbf{x}|_1 < n \\ k\cdot n & |\mathbf{x}|_1 = n \end{cases} \tag{1.4}$$

The bigger m, the wider the valley. k can be increased to give bigger weight to the maximum. For $m = 1$ we obtain the popular OneMax function defined by $OneMax(n) = |x|_1$.

BigJump depends only on the number of bits on. To simplify the analysis we assume that all $p(x_i = 1)$ are identical to a single value denoted as $p(t)$. Then W depends only on one parameter, p. $W(p)$ is shown for $m = 30$ and $k = 20$ in figure 1.1. In contrast to the BigJump function $W(p)$ looks fairly smooth. The open circles are the values of $p(t)$ determined by an UMDA rum, setting $p(t) := 1/n \sum_i p(x_i,t)$. Note how closely the simulation follows the theoretical curve. Furthermore the probability $p(t)$ changes little over time

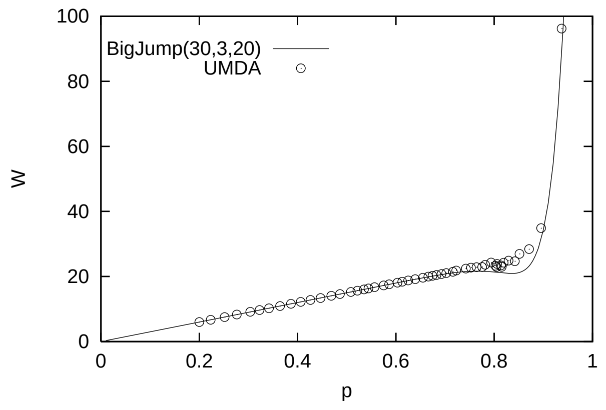

Figure 1.1
BigJump(30,3,20), UMDA, p versus average fitness, Popsize=2000

when W increases only slightly.

This simple example demonstrates in a nutshell the results of our theory. It can be formulated as follows: *Evolutionary algorithms transform the original fitness landscape given by* $f(\mathbf{x})$ *into a fitness landscape defined by* $W(\mathbf{p})$. *This transformation smoothes the rugged fitness landscape* $f(\mathbf{x})$. *There exist difficult fitness landscapes* $f(\mathbf{x})$ *like* $BigJump$ *which are transformed into simple landscapes* $W(p)$. *In these landscapes simple evolutionary algorithms will find the global optimum.*

A still more spectacular example is the Saw landscape. The definition of the function can be extrapolated from figure 1.2. In Saw(n,m,k), n denotes the number of bits and $2m$ the distance from one peak to the next. The highest peak is multiplied by k (with $k \leq 1$), the second highest by k^2, then k^3 and so on. The landscape is very rugged. In order to get from one local optimum to another one, one has to cross a deep valley.

But again the transformed landscape $W(p)$ is fairly smooth. An example is shown in figure 1.3. Whereas $f(x)$ has 5 isolated peaks, $W(p)$ has three plateaus, a local peak and the global peak. Therefore we expect that UMDA should be able to cross the plateaus and terminate at the local peak. This behavior can indeed be observed in figure 1.3. Furthermore, as predicted by equation 1.3 the progress of UMDA slows down on the plateaus.

These two examples show that UMDA can solve difficult multi-modal optimization problems. But there are many optimization problems where UMDA

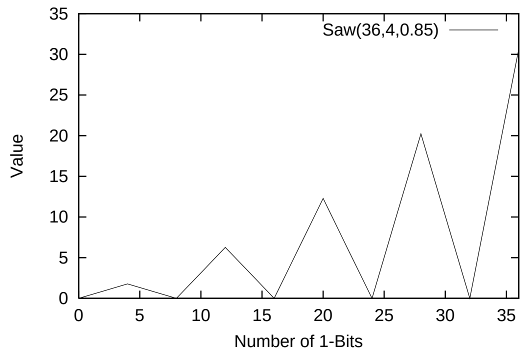

Figure 1.2
Definition of Saw(36,4,0.85)

is mislead. UMDA will converge to local optima, because it cannot explore correlations between the variables. The first example is a deceptive function. We use the definition

$$\text{Decep}(\mathbf{x}, k) := \begin{cases} k - 1 - |\mathbf{x}|_1 & 0 \le |\mathbf{x}|_1 < k \\ k & |\mathbf{x}|_1 = k \end{cases} \tag{1.5}$$

The global maximum is isolated at $x = (1, \ldots 1)$. A deceptive function of order k is a needle in a haystack problem. This is far too difficult to optimize. We simplify the optimization problem by adding l distinct $Decep(k)$-functions to give a fitness function of size $n = l * k$. This function is also deceptive. The local optimum $x = (0, \ldots, 0)$ is surrounded by good fitness values, whereas the global optimum is isolated.

$$\text{Decep}(n, k) = \sum_{i=1,k+1,\ldots}^{n} \text{Decep}\big((x_i, x_{i+1}, \ldots, x_{i+k-1}), k\big) \tag{1.6}$$

In figure 1.4 we show the average fitness $W(p)$ and an actual UMDA run. Starting at $p(0) = 0.5$ UMDA converges to the local optimum $\mathbf{x} = (0, \ldots, 0)$. UMDA will converge to the global optimum if it starts near to the optimum, e.g. $p(0) \ge 0.6$. Also shown is a curve which arises from an algorithm using fourth order marginal distributions. This algorithm converges to the global optimum, even when the initial population is generated randomly. But also for this algorithm $p(t)$ decreases first. The algorithm is discussed in section 1.3.

The next example is a modified BigJump. The peak is now shifted away

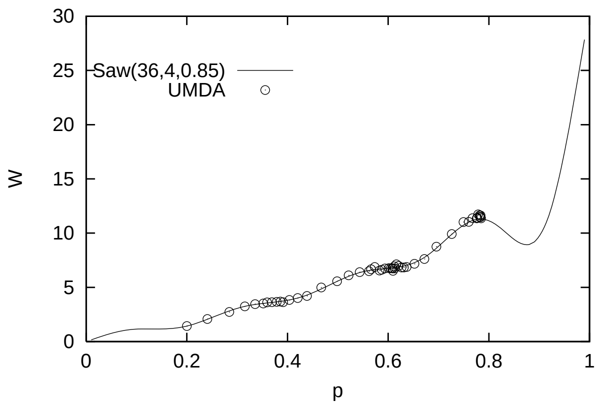

Figure 1.3
Saw(36,4,0.85), UMDA, p versus average fitness, Popsize=2000

from the corner $\mathbf{x} = (1, \ldots, 1)$. It is defined as follows

$$\begin{aligned} \text{S-Peak}(n, m, k) \quad &= \quad \left(\sum_{i=1}^{n} x_i \right) + \\ & k \cdot (m+1)\big((1-x_1)\cdots(1-x_m)x_{m+1}\cdots x_n\big) \end{aligned}$$

This function is difficult to optimize, because the global optimum at $(0, \ldots, 0, 1, \ldots, 1)$ is now in competition with a local optimum at $x = (1, \ldots, 1)$. We discuss in more detail the function S-Peak(30,3). For the presentation we assume that the first three univariate marginal distributions are equal, e.g. $p(x_1) = p(x_2) = p(x_3) := a$ and the remaining ones are equal to b. This means that W depends on two parameters. In order to generate the global optimum a has to approach 0 and b has to approach 1. In figure 1.5 iso-lines of $W(a, b)$ are shown as well as trajectories for UMDA runs starting at different initial positions. The trajectories are almost vertical to the iso-lines, as predicted by the gradient ascent.
In all runs the initial b was set to 0.5, a varied from 0.1 to 0.3. For $0 < a(0) < 0.26$ UMDA will converge to the global optimum, for $a \geq 0.26$ UMDA will converge to the local optimum.

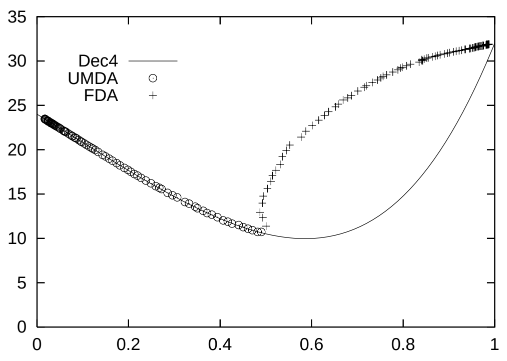

Figure 1.4
Average fitness $W(p)$ for UMDA and FDA for Decep(36,4)

Selection

Proportionate selection allows a mathematical analysis. The equations for the univariate marginal distributions can be explicitly given. Proportionate selection is therefore also popular in population genetics. It is considered to be a model for *natural selection*.

But proportionate selection has a drawback in practice. It strongly depends on the fitness values. When the population approaches an optimum, selection gets weaker and weaker, because the fitness values are similar. Therefore breeders of livestock have used other selection methods. For large populations they mainly apply *truncation selection*. It works as follows. A truncation threshold τ is fixed. Then the τN best individuals are selected as parents for the next generation. These parents are then randomly mated.

We use mainly truncation selection in our algorithms. Another popular scheme is *tournament selection*. In tournament selection of size k, k individuals are randomly chosen. The best individual is taken as parent. Unfortunately the mathematics for both selection methods is more difficult. Analytical results for tournament selection have been obtained in [9].

We model binary tournament selection as a game. Two individuals with genotype $\mathbf{x}$ and $\mathbf{y}$ "play" against each other. The one with the larger fitness gets a payoff of 2. If the fitness values are equal, both will win half of the games. This gives a payoff of 1. The game is defined by a *payoff matrix* with

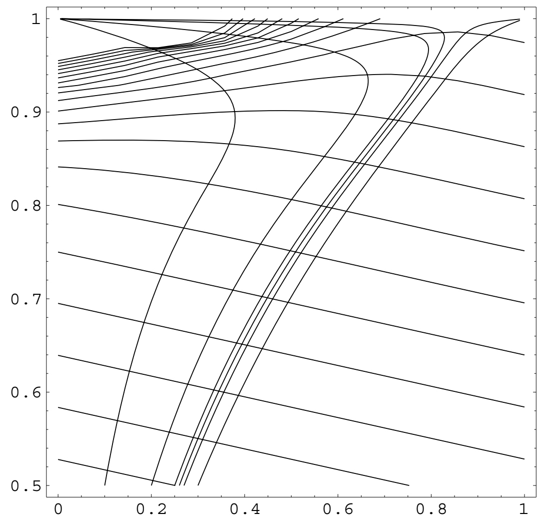

Figure 1.5
Trajectories of UMDA for S-Peak(30,3)

coefficients

$$a_{xy} = \begin{cases} 2 & f(\mathbf{x}) > f(\mathbf{y}) \\ 1 & f(\mathbf{x}) = f(\mathbf{y}) \\ 0 & f(\mathbf{x}) < f(\mathbf{y}) \end{cases}$$

With some effort one can show that

$$\sum_{\mathbf{x}} \sum_{\mathbf{y}} p(\mathbf{x},t) a_{xy} p(\mathbf{y},t) = 1 \tag{1.7}$$

After a round of tournaments the genotype frequencies are given by

$$p^s(\mathbf{x},t+1) = p(\mathbf{x},t) \sum_{\mathbf{y}} a_{xy} p(\mathbf{y},t). \tag{1.8}$$

If we set

$$b(\mathbf{x},t) = \sum_{\mathbf{y}} a_{xy} p(\mathbf{y},t),$$

then the above equation is similar to proportionate selection using the function $b(\mathbf{x},t)$. But b depends on the genotype frequencies. Furthermore the average $\bar{b}(t) = \sum p(\mathbf{x},t) b(\mathbf{x},t)$ remains constant, $\bar{b}(t) \equiv 1$.

The difference equations for the univariate marginal frequencies can be derived in the same manner as for proportionate selection. They are given by

$$p(x_i, t+1) = p(x_i, t) \cdot \bar{B}_i(t) \tag{1.9}$$

$$\bar{B}_i(t) = \sum_{\mathbf{x}|x_i=1} b(\mathbf{x}, t) \prod_{\substack{j=1 \\ j \neq i}}^{n} p_j(x_j, t) - 1 \tag{1.10}$$

Tournament selection uses only the order relation of the fitness values. The fitness values themselves do not change the outcome of a tournament. Therefore the evolution of the univariate marginal frequencies depends on the order relation only. Tournament selection does not perform gradient ascent on the average fitness of the population.

The different selection schemes are shown in figure 1.6 for OneMax(128).

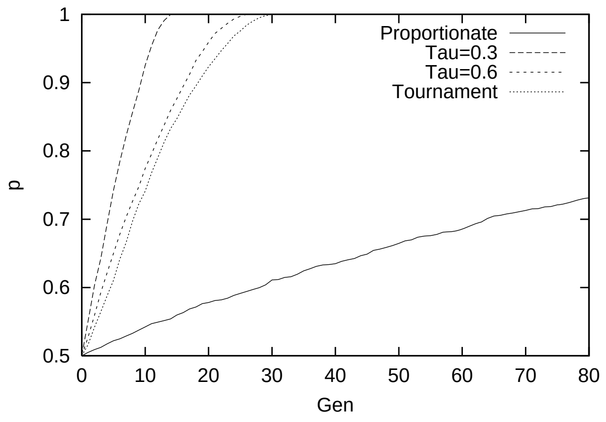

Figure 1.6
Comparison of selection methods for OneMax(128)

For OneMax proportionate selection is very weak. It takes the population a long time to approach the optimum. In contrast, truncation selection and tournament selection lead to a much faster convergence. p increases almost linearly until near the optimum. Truncation selection with $\tau = 0.6$ behaves very similarly to tournament selection.

A more complicated problem is the multi-modal function Saw$(36, 4, 0.85)$ displayed in figure 1.7. From figure 1.3 we recall that there are two plateaus at $p = 0.4$ and $p = 0.6$ and a local optimum at about $p = 0.78$. This structure is reflected both for truncation selection and proportionate selection. But truncation selection converges much faster. Both selection methods end on the local optimum $p = 0.78$.

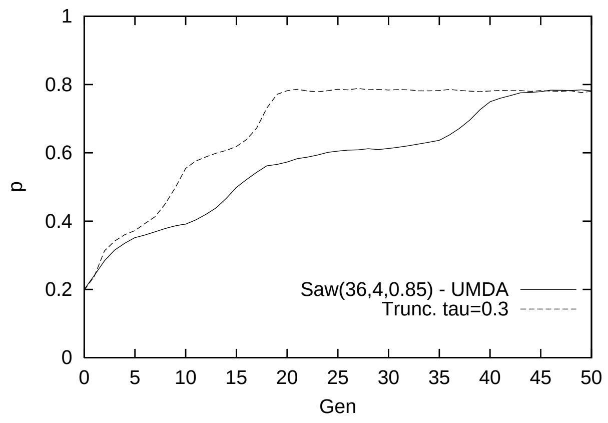

Figure 1.7
Comparison of selection methods for Saw(36,4,0.85)

The main conclusion of the theory remains valid for all selection schemes. *Evolution is mainly driven by the fitness landscape defined by* $W(p)$. For proportionate selection this can be theoretically shown. For the other selection methods an approximate analysis gives the same result. The analysis is based on the *response to selection* equation

$$W(p(t+1)) - W(p(t)) \approx b \cdot I_\tau(\sigma(W(p(t)))) \tag{1.11}$$

The response $W(p(t+1)) - W(p(t))$ is determined by the fitness distribution, not the fitness values. In a first approximation it is assumed that the fitness distribution is normally distributed and can be characterized by the standard deviation $\sigma(W(p(t)))$. $b(t)$ is called realized heritability and I_τ the selection intensity corresponding to truncation threshold τ.

This equation and the selection problem is discussed in detail in [9]. It has been used to determine an analytical solution for OneMax.

UMDA solves many difficult multi-modal functions. But it can easily be deceived by a simple function like the deceptive function. The deception problem can be solved by using exact factorizations of the probability. This is discussed next.

1.3 FDA – The Factorized Distribution Algorithm

For the mathematical analysis we will assume Boltzmann selection. Boltzmann selection can be seen as proportionate selection applied to the transformed function $F(\mathbf{x}) = \exp(\beta f(\mathbf{x}))$.

Definition: *For Boltzmann selection the distribution after selection is given by*

$$p^s(\mathbf{x}, t) = p(x,t)\frac{e^{\beta f(x)}}{W_\beta} \tag{1.12}$$

where $\beta > 0$ is a parameter, also called the inverse temperature, and $W_\beta = \sum p(\mathbf{x},t)e^{\beta f(x)}$ is the weighted average of the population.

For Boltzmann distributions we have proven a factorization theorem for the distribution $p(\mathbf{x}, t)$ and convergence for an algorithm using this factorization [10]. The proof is simple, because if $p(x,t)$ is a Boltzmann distribution with factor β_1 and Boltzmann selection is done with factor β_2, then $p(\mathbf{x}, t+1) = p(\mathbf{x}, t)$ is a Boltzmann distribution with factor $\beta = \beta_1 + \beta_2$.

THEOREM 1.1: Let $p(\mathbf{x}, 0)$ be randomly distributed. Let $\beta_1, \ldots, \beta_{t-1}$ be the schedule of the inverse temperature for Boltzmann selection. Then the distribution is given by

$$p(\mathbf{x}, t) = \frac{e^{\beta f(\mathbf{x})}}{Z_\beta} \tag{1.13}$$

where $\beta = \sum_{i=1}^{t-1} \beta_i$. Z_β is the partition function $Z_\beta = \sum_{\mathbf{x}} e^{\beta f(\mathbf{x})}$.

Equation 1.13 is a complete analytical solution of the dynamics. But it cannot be used for an algorithm. $p(\mathbf{x}, t)$ consists of $2^n - 1$ variables. Therefore the amount of computation is exponential. But there are many cases where the distribution can be factored into conditional marginal distributions each depending only on a small number of parameters. We recall the definition of conditional probability.

Definition *The conditional probability* $p(\mathbf{x}|\mathbf{y})$ *is defined as*

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{y})} \tag{1.14}$$

From this definition the following theorem easily follows.

THEOREM 1.2 BAYESIAN FACTORIZATION: Each probability can be factored into

$$p(\mathbf{x}) = p(x_1)\prod_{i=2}^{n} p(x_i|pa_i) \tag{1.15}$$

Proof: By definition of conditional probabilities we have

$$p(\mathbf{x}) = p(x_1)\prod_{i=2}^{n} p(x_i|x_1,\cdots,x_{i-1}) \tag{1.16}$$

Let $pa_i \subset \{x_1,\cdots,x_{i-1}\}$. If x_i and $\{x_1,\cdots,x_{i-1}\}\setminus pa_i$ are conditionally independent given pa_i, we can simplify $p(x_i|x_1,\cdots,x_{i-1}) = p(x_i|pa_i)$. ■

PA_i are called the parents of variable X_i. This factorization defines a directed graph. In the context of graphical models the graph and the conditional probabilities are called a Bayesian network [5, 2]. The factorization is used by the Factorized Distribution Algorithm FDA.

FDA

- **STEP 0:** Set $t \Leftarrow 0$. Generate $N \gg 0$ points randomly.
- **STEP 1:** Selection
- **STEP 2:** Compute the conditional probabilities $p^s(x_i|pa_i,t)$ using the selected points.
- **STEP 3:** Generate a new population according to

$$p(x,t+1) = \prod_{i=1}^{n} p^s(x_i|pa_i,t)$$

- **STEP 4:** If termination criteria is met, FINISH.
- **STEP 5:** Set $t \Leftarrow t+1$. Go to STEP 2.

FDA can be used with an exact or an approximate factorization. It is not restricted to Bayesian factorization. FDA uses *finite samples* of points to estimate the conditional distributions. Convergence of FDA to the optimum will depend on the size of the samples.

If the factorization does not contain conditional marginal distributions, but only marginal distributions, FDA can be theoretically analyzed. The difference equations of the marginal distributions are of the form given in equation 1.3 [7].

The amount of computation of FDA depends on the size of the population (N) and the number of variables used for the factors. There exist many problems where the size of the factors is bounded by k independent from n. In this case FDA is very efficient [8]. But for the function BigJump an exact factorization needs a factor of size n. Then the amount of computation of FDA is exponential in n. We have seen before that for BigJump UMDA will already find the global optimum. Thus an exact factorization is not a necessary condition for convergence. But it is necessary if we want to be sure that the optimum is found.

Approximation of the Distribution

The FDA theory needs an exact factorization. In order to use approximate distributions one can try to formulate the problem as an approximation problem of distributions. Given a target distribution (the Boltzmann distribution) and a family of probability models: What is the best approximation of the target distribution and how can the best approximation be found?

We restrict the discussion to distributions defined by the product of univariate marginal distributions.

Approximation Problem: Given a Boltzmann distribution

$$p(\mathbf{x}, t) = \frac{e^{\beta f(\mathbf{x})}}{Z_\beta} \tag{1.17}$$

with $\beta > 0$. Given approximations $\tilde{p}(\mathbf{x}, t) = \prod \tilde{p}(x_i, t)$: Which set of $\tilde{p}(x_i, t)$ minimizes some distance to the Boltzmann distribution?

A popular measure for estimating the approximation error between distributions is the *entropy distance* proposed by Kullback and Leibler [6].

Definition: The Kullback Leibler divergence between two distributions is defined by

$$KL(\tilde{p}, p) = \sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \cdot \log \frac{\tilde{p}(\mathbf{x})}{p(x)} \tag{1.18}$$

The KL measure has the following properties

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

The entropy distance is not symmetric, i.e. $KL(\tilde{p}, p) \neq KL(p, \tilde{p})$. The entropy distance depends on how close p and $\tilde{p}$ are to the boundary. To see this, note that the term $(x + \epsilon)\log((x + \epsilon)/x)$ grows to infinity when x goes to 0. Thus the entropy distance can be arbitrarily large while p and $\tilde{p}$ differ by ϵ. More generally, the entropy distance is extremely sensitive to small deviations close to the boundary of the probability simplex.

The above argument indicates that convergence in terms of entropy distance is harder than convergence in some L^p norm. This raises the question why we should use this measure. A discussion is outside the scope of this paper.

The best approximation minimizes $KL(\tilde{p}, p)$. The computation of the best approximation is very difficult. The topic is still under study. In practice a different approach is now popular. The approach is based on the theory of Bayesian networks.

1.4 LFDA - Learning a Bayesian Factorization

Computing the structure of a Bayesian network from data is called learning. Learning gives an answer to the question: *Given a population of selected points $M(t)$, what is a good Bayesian factorization fitting the data?* The most difficult part of the problem is to define a quality measure also called scoring measure.

A Bayesian network with more arcs fits the data better than one with less arcs. Therefore a scoring metric should give the best score to the minimal Bayesian network which fits the data. It is outside the scope of this paper to discuss this problem in more detail. The interested reader is referred to the two papers by Heckerman and Friedman et al. in [5].

For Bayesian networks two quality measures are most frequently used - the *Bayes Dirichlet* (BDe) score and the *Minimal Description Length* (MDL) score. We concentrate on the MDL principle. This principle is motivated by universal coding. Suppose we are given a set D of instances, which we would like to store. Naturally, we would like to conserve space and save a compressed version of D. One way of compressing the data is to find a suitable model for D that the encoder can use to produce a compact version of D. In order to recover D we must also store the model used by the encoder to compress D. Thus the total description length is defined as the sum of the length of the compressed version of D and the length of the description of the model. The MDL principle postulates that the optimal model is the one that minimizes the total description length.

In the context of learning Bayesian networks, the model is a network B describing a probability distribution p over the instances appearing in the data. Several authors have approximately computed the MDL score. Let $M = |D|$ denote the size of the data set. Then MDL is approximately given by

$$MDL(B,D) = -\text{ld}(P(B)) + M \cdot H(B,D) + \frac{1}{2} PA \cdot \text{ld}(M) \tag{1.19}$$

with $\text{ld}(x) := \log_2(x)$. $P(B)$ denotes the prior probability of network B, $PA = \sum_i 2^{|pa_i|}$ gives the total number of probabilities to compute. $H(B,D)$ is defined by

$$H(B,D) = -\sum_{i=1}^{n} \sum_{pa_i} \sum_{x_i} \frac{m(x_i,pa_i)}{M} \text{ld} \frac{m(x_i,pa_i)}{m(pa_i)} \tag{1.20}$$

where $m(x_i,pa_i)$ denotes the number of occurrences of x_i given configuration pa_i. $m(pa_i) = \sum_{x_i} m(x_i,pa_i)$. If $pa_i = \emptyset$, then $m(x_i,\emptyset)$ is set to the number of occurrences of x_i in D.

The formula has an interpretation which can be easily understood. If no prior information is available, $P(B)$ is identical for all possible networks. For minimizing, this term can be left out. $0.5PA \cdot \text{ld}(M)$ is the length required to code the parameter of the model with precision $1/M$. Normally one would need $PA \cdot \text{ld}(M)$ bits to encode the parameters. However, the central limit theorem says that these frequencies are roughly normally distributed with a variance of $M^{-1/2}$. Hence, the higher $0.5\text{ld}(M)$ bits are not very useful and can be left out. $-M \cdot H(B,D)$ has two interpretations. First, it is identical to the logarithm of the maximum likelihood ($\text{ld}(L(B|D))$). Thus we arrive at the following principle:

Choose the model which maximizes ld$(L(B|D)) - \frac{1}{2} PA \cdot$ *ld*(M).

The second interpretation arises from the observation that H(B,D) is the conditional entropy of the network structure B, defined by PA_i, and the data D. The above principle is appealing, because it has no parameter to be tuned. But the formula has been derived under many simplifications. In practice, one needs more control about the quality vs. complexity tradeoff. Therefore we use a weight factor α. Our measure to be maximized is called BIC.

$$BIC(B,D,\alpha) = -M \cdot H(B,D) - \alpha PA \cdot \text{ld}(M) \tag{1.21}$$

This measure with $\alpha = 0.5$ has been first derived by Schwarz [13] as *Bayesian Information Criterion.*

To compute a network B^* which maximizes BIC requires a search through the space of all Bayesian networks. Such a search is more expensive than to search for the optima of the function. Therefore the following greedy algorithm has been used. k_{max} is the maximum number of incoming edges allowed.

$$\mathbf{BN}(\alpha, \mathbf{k_{max}})$$

- **STEP 0:** Start with an arc-less network.
- **STEP 1:** Add the arc (x_i, x_j) which gives the maximum increase of BIC(α) if $|PA_j| \leq k_{max}$ and adding the arc does not introduce a cycle.
- **STEP 2:** Stop if no arc is found.

Checking whether an arc would introduce a cycle can be easily done by maintaining for each node a list of parents and ancestors, i.e. parents of parents etc. Then $(x_i \rightarrow x_j)$ introduces a cycle if x_j is ancestor of x_i.

The BOA algorithm of Pelikan [11] uses the BDe score. This measure has the following drawback. It is more sensitive to coincidental correlations implied by the data than the MDL measure. As a consequence, the BDe measure will prefer network structures with more arcs over simpler networks [1]. The BIC measure with $\alpha = 1$ has also been proposed by Harik [4]. But Harik allows only factorizations without conditional distributions. This distribution is only correct for separable functions.

Given the BIC score we have several options to extend FDA to LFDA which learns a factorization. Due to limitations of space we can only show results of an algorithm which computes a Bayesian network at each generation using algorithm $BN(0.5, k_{max})$. FDA and LFDA should behave fairly similar,

Table 1.1
Numerical results for different algorithms, LFDA with $BN(\alpha, 8)$

Function	n	α	N	τ	Succ.%	SDev
OneMax	30	UMDA	30	0.3	75	4.3
	30	0.25	100	0.3	2	1.4
	30	0.5	100	0.3	38	4.9
	30	0.75	100	0.3	80	4.0
	30	0.25	200	0.3	71	4.5
BigJump(30,3,1)	30	UMDA	200	0.3	100	0.0
	30	0.25	200	0.3	58	4.9
	30	0.5	200	0.3	96	2.0
	30	0.75	200	0.3	100	0.0
	30	0.25	400	0.3	100	0.0
Saw(32,2,0.5)	32	UMDA	50	0.5	71	4.5
	32	UMDA	200	0.5	100	0.0
	32	0.25	200	0.5	41	2.2
	32	0.5	200	0.5	83	1.7
	32	0.75	200	0.5	96	0.9
	32	0.25	400	0.5	84	3.7
Deceptive-4	32	UMDA	800	0.3	0	0.0
	32	FDA	100	0.3	81	3.9
	32	0.25	800	0.3	92	2.7
	32	0.5	800	0.3	72	4.5
	32	0.75	800	0.3	12	3.2
S-Peak(15,3,15)	15	UMDA	1000	0.5	75	4.3
	15	0.25	1000	0.5	77	4.2
	15	0.5	1000	0.5	70	4.6
	15	0.75	1000	0.5	76	4.3

if LFDA computes factorizations which are in probability terms very similar to the FDA factorization. FDA uses the same factorization for all generations, whereas LFDA computes a new factorization at each step which depends on the given data M.

We have applied LFDA to many problems [8]. The results are encouraging. Here we only discuss the functions introduced in Section 1.2. We recall that UMDA finds the optimum of BigJump, Saw and small instances of S-Peak. UMDA uses univariate marginal distributions only. Therefore its Bayesian network has no arcs.

Table 1.1 summarizes the results. For LFDA we used three different values of α, namely $\alpha = 0.25, 0.5, 0.75$. The smaller α, the less penalty for the size of the structure. Let us discuss the results in more detail. $\alpha = 0.25$ gives by far the best results when a network with many arcs is needed. This is the case for Deceptive-4. Here a Bayesian network with three parents is optimal. $\alpha = 0.25$ performs bad on problems where a network with no arcs defines a good search distribution. For the linear function OneMax $BIC(0.25)$ has only a success rate of 2%. The success rate can be improved if a larger population size N is used. The reason is as follows. $BIC(0.25)$ allows denser networks. But if a small population is used, spurious correlations may arise. These correlations have a negative impact for the search distribution. The problem can be solved by using a larger population. Increasing the value from $N = 100$ to $N = 200$ increases the success rate from 2% to 71% for OneMax.

For Bigjump, Saw and S-Peak a Bayesian network with no arcs is able to generate the optimum. An exact factorization requires a factor with n parameters. We used the heuristic BN with $k_{max} = 8$. Therefore the exact factorization cannot be found. In all these cases $\alpha = 0.75$ gives the best results. $BIC(0.75)$ enforces smaller networks. But $BIC(0.75)$ performs very bad on Deceptive-4. Taking all results together $BIC(0.5)$ gives good results. This numerical results supports the theoretical estimate.

The numerical result indicates that control of the weight factor α can substantially reduce the amount of computation. For Bayesian network we have not yet experimented with control strategies. We have intensively studied the problem in the context of neural networks [15]. The method will be discussed in section 1.6.

Network Equivalence and Optimality

Many Bayesian networks represent the same probability. These networks are called *equivalent*. Let us discuss a simple example.

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_2,x_1)p(x_4|x_3) \tag{1.22}$$

The independence structure is shown in figure 1.8.

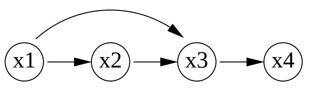

Figure 1.8
Structure of a Bayesian network

It can be proven that the following factorizations describe the same probability distribution.

$$p(x) = p(x_2)p(x_3|x_2)p(x_1|x_2,x_3)p(x_4|x_3) \tag{1.23}$$

$$p(x) = p(x_3)p(x_4|x_3)p(x_2|x_3)p(x_1|x_2,x_3) \tag{1.24}$$

$$p(x) = p(x_4)p(x_3|x_4)p(x_2|x_3)p(x_1|x_2, x_3) \tag{1.25}$$

Furthermore all Bayesian factorizations which contain one of the above factorizations also generate the true distribution. We say that B_1 *is contained in* B_2 *if all arcs of* B_1 *are also arcs in* B_2.

THEOREM 1.3: Let B^* be a Bayesian network which generates the data and has the minimal number of arcs. The MDL score, as well as the BDe score has the following characteristics.

- As $M \to \infty$ the true structure B^* maximizes the scores.
- As $M \to \infty$ the maximal scoring structures are equivalent.

Proof: The proof is technically difficult, we just give an outline. Suppose network B has an arc not contained in B^*, then for $M \to \infty$

$$-M \cdot H(B, D) \quad \to \quad -M \cdot H(B^*, D) - e \cdot M$$

where e depends on B. Therefore

$$BIC(B^*, D, \alpha) - BIC(B, D, \alpha) \to e \cdot M - \alpha \cdot log(M) \cdot (dim(B^*) - dim(B))$$

Because $M/log(M) \to \infty$ the score of B^* is much larger than the score of B. This proves the first part of the theorem. Now let us assume that B^* has an arc not contained in B. As $M \to \infty$ we have $-M \cdot H(B, D) \to -M \cdot H(B^*, D)$. Therefore

$$BIC(B, D, \alpha) - BIC(B^*, D, \alpha) \to \alpha \cdot log(M) \cdot (dim(B^*) - dim(B)) \tag{1.26}$$

But if $dim(B) > dim(B^*)$ then $BIC(B, D, \alpha)$ cannot be maximal. ∎
The above theorem is true for all values of $\alpha > 0$. For the proof only the term $log(M)$ is needed. We can now ask if our search algorithm $BN(\alpha, k)$ is able to find an optimal structure, given enough data. Unfortunately the following result shows that this cannot be proven in general [1].

Let a probability distribution be given. Then the problem of finding a Bayesian network with a minimum number of arcs is NP-hard for $k > 1$. *If only trees are allowed, there exists a polynomial algorithm (in the number of variables).*

Bayesian networks

In this paper we have used Bayesian networks in a new context: to find good search distributions for an optimization problem. Normally Bayesian networks are used in a different context: to provide a formalism for reasoning about partial beliefs under conditions of uncertainty. Therefore Bayesian networks are also called *belief networks*. The nodes represent random variables. The arcs signify the existence of direct causal influences between the variables. Ultimately the belief network represents a probability distribution over X having the Bayesian product form.
The following problems are defined over belief networks with fixed structure:

Belief updating: Given a set of observations, compute the posterior probability of each proposition.

Missing values: Given some observed variables, find a maximum probability assignment to the rest of the variables.

Creating hypothesis: Given some evidence, find an assignment to a subset of hypothesis variables that maximizes their probability.

In general these tasks are NP-hard. But they permit a polynomial propagation algorithm for tree networks. The above tasks will be discussed in the context of neural networks in section 1.6. Next we will investigate genetic programming.

1.5 Synthesis of Programs

Salustewicz and Schmidhuber [12] have proposed a method for the synthesis of programs bases on probabilities. This method is very similar to the methods discussed in this paper. Therefore we summarize the major concepts of the method. In their method programs are generated according to a probabilistic prototype tree (PPT).

A program contains instructions chosen from a set of functions $F = \{f_1, f_2, \ldots, f_k\}$ and a terminal set $T = \{t_1, t_2, \ldots t_l\}$. Functions have one or more arguments, terminals have no argument. The terminal set consists of input variables x_i and a generic random constant $R \in [0;1)$ which will be changed during the run.

Programs are encoded in m-ary trees with m being the maximum number of function arguments. Each non-leaf node encodes a function from F and each leaf node a terminal from T. The number of subtrees that each node has corresponds to the number of arguments for its function. The Probabilistic Prototype Tree (PPT) is generally a complete m-ary tree. At each node $N_{d,w}$ it contains a probability vector $p_{d,w}$ and a random constant $R_{d,w}$. d denotes the depth of the node and w defines the node's horizontal position. PPT is parsed from top to bottom and left to right. The probability vector $p_{d,w}$ has $i = l + k$ elements. Each element $p_{d,w}(I)$ denotes the probability of choosing instruction $I \in F \cup T$ at $N_{d,w}$.

A program PROG is generated from PPT in the following way. An instruction I_{prog} at node $N_{d,w}$ is selected with probability $p_{d,w}(I_{prog})$. If $I_{prog} \in F$, a subtree is created for each argument of I_{prog}. If $I_{prog} \in T$ the selected input variable or a constant is put at the program tree. Therefore a program is generated with probability

$$p(PROG) = \prod_{d,w} p_{d,w}(I_{prog}) \tag{1.27}$$

Thus the algorithm uses univariate marginal distributions. I_{prog} is not a binary variable, but has i distinct values. Therefore our theory developed in the first sections can be applied. We have yet to define a fitness. Normally the program gets only examples $(\mathbf{x}_j, \mathbf{y}_j)$. A measure of the goodness of a program is given by

$$score(PROG) = \sum_j |\mathbf{y}_j - PROG(\mathbf{x}_j)| \tag{1.28}$$

The quadratic score popular with neural networks can be used instead.

The presented model is very similar to UMDA. The only extension is the usage of a probabilistic tree instead of a probabilistic string. Each probabilistic variable has $l + k$ elements. Using only univariate marginal distributions is obviously a very crude approximation for the problem domain considered. An extension to conditional marginal distributions seems necessary. In a first approximation each node at layer $k + 1$ can be conditioned on its parent node at layer k. For tree networks very efficient algorithms exist which determine the optimal structure. This proposal needs further study.

1.6 Neural Trees

Bayesian networks are founded on probability theory. The problem of determining the optimal parameters of a Bayesian network given the data is solved by probability theory. The optimal parameter are just the empirical frequencies of the conditional marginal distributions. There also exist powerful heuristics like $BN(\alpha)$ which change the structure of a Bayesian network to better fit the data.

For general neural networks like the famous multilayer perceptron the situation is much more difficult. The determination of the optimal parameters (called weights) is difficult. Special algorithms have been developed for this problem. In addition there exist no heuristics backed up by theory which adapt the structure.

In principle Bayesian networks can also be used in the domain of neural networks: to learn a function $\mathbf{x} \rightarrow \mathbf{y} = f(\mathbf{x})$. The mapping $(\mathbf{x}, \mathbf{y})$ can be represented by a conditional probability distribution $p(\mathbf{y}|\mathbf{x})$. But during the training $\mathbf{y}$ will depend on all variables of $\mathbf{x}$ or of the variables of a hidden layer. This means that the number of parents used for Bayesian factorization is n. Thus the amount of computation is of order $O(2^n)$ making this approach obsolete.

Currently two paths are followed to attack this problem. The first path restricts the class of neural networks. There exist a subclass of neural networks called *sigmoid belief networks* which allow a probabilistic interpretation. The interconnection structure of the belief networks has to be restricted to trees in order that the algorithms invented for Bayesian networks can be applied.

We have experimented with neural trees. Here the neurons are complex, but the structure is restricted to trees. Because the neurons are complex, we do not yet have a probabilistic interpretation of neural trees. Therefore we are faced with two problems: to determine the optimal weights for a structure and to find the optimal structure.

Let $\mathcal{NT}(d, b)$ denote the set of all possible trees of maximum depth d and maximum number of branches for each node b. The nonterminal nodes represent (formal) neurons and the neuron type is an element of the basis function set $\mathcal{F} = \{\text{neuron types}\}$. Each terminal node is labeled with an element from the terminal set $\mathcal{T} = \{x_1, x_2, \ldots, x_n\}$, where x_i is the ith component of the external input $\mathbf{x}$. Each link (i, j) represents a directed connection from node i to node j and is associated with a value $w_{i,j}$, called synaptic weight. The members of $\mathcal{NT}(d, b)$ are referred to as neural trees. The root node is also called the output unit and the terminal nodes are called input units. The rest are hidden units. The layer of a node is defined as the longest path length among the paths to the terminal nodes of its subtree.

Different neuron types compute different net inputs. For the evolution of higher-order networks, we consider two types of units. The sigma units compute the sum of weighted inputs from the lower layer:

$$net_i = \sum_j w_{ij} y_j \tag{1.29}$$

where y_j are the inputs to the ith neuron. The pi units compute the product of weighted inputs from the lower layer:

$$net_i = \prod_j w_{ij} y_j \tag{1.30}$$

where y_j are the inputs to i. The output of the neurons is computed either by the threshold response function

$$y_i = \sigma(net_i) = \begin{cases} 1 & : \quad net_i \geq 0 \\ -1 & : \quad net_i < 0 \end{cases} \tag{1.31}$$

or the sigmoid transfer function

$$y_i = f(net_i) = \frac{1}{1 + e^{-net_i}} \tag{1.32}$$

where net_i is the net input to the unit.

A higher-order network with m output units can be represented by a a list of m neural trees. That is, the genotype A_i of ith individual in our evolutionary framework consists of m sigma-pi neural trees:

$$A_i = (A_{i,1}, A_{i,2}, \ldots, A_{i,m}) \quad \forall k \in \{1, \ldots, m\},\ A_{i,k} \in \mathcal{NT}(d, b) \tag{1.33}$$

These neural networks are called *sigma-pi neural trees*. The neural tree representation does not restrict the functionality since any feed-forward network can be represented with a forest of neural trees. The connections between input units to arbitrary units in the network is also possible since input units can appear more than once in the neural tree representation. The output of one unit can be used as input to more than one unit in the upper layers by using them more than once.

Searching for the Best Structure

The structural adaptations alter the topology, size, depth, and shape of neural networks: the first three are changed by crossover and the last by mutations.

As with other genetic programming methods, the crossover operation is in essence performed by exchanging subtrees of parent individuals. Because of the uniformity of the neural tree representation, no syntactic restriction is necessary in choosing the crossover points. Instead of producing two offspring, we create only one, which allows a guided crossover by subtree evaluation. From two parent individuals A and B, the offspring C is generated by the following procedure:

1. Select a subtree a of A at random or one that has poor local fitness value.
2. Select a subtree b of B at random or one that has good local fitness value.
3. Produce an offspring C by copying A and replacing a with b.

The local fitness is measured by a combination of the error and size of the subtrees. Other criteria for subtree evaluation proposed in the literature include the error of the subtree, error difference, frequency of subtrees, use of the average fitness of population, correlation-based selection, combination of frequency and error difference [15]. Mutation and adding of library functions is also discussed there.

1.7 MDL-Based Fitness Function

The goal is to find a neural tree or model A whose evaluation $f_A(\mathbf{x})$ best approximates the unknown relation $\mathbf{y} = f(\mathbf{x})$ given an input $\mathbf{x}$. The goodness of the program for the dataset D is measured by the quadratic error

$$E(A|D) \quad = \quad \frac{1}{N}\sum_{c=1}^{N}\left(\mathbf{y}_c - f_A(\mathbf{x}_c)\right)^2, \tag{1.34}$$

where N is the number of training examples.

We can now apply the MDL principle as before. The data consists of examples $(\mathbf{x}, \mathbf{y})$. In general, however, the distribution of the examples is unknown and the exact formula for the fitness function is impossible to obtain. Therefore we propose to approximate MDL by

$$MDL(A, D) = E(A|D) + \alpha C(A), \tag{1.35}$$

where the parameter α controls the trade-off between complexity $C(A)$ and fitting error $E(D|A)$ of the neural tree.

Now we describe a heuristic that controls α. Care must be taken in applying the MDL principle to neural trees so that redundant structures should be pruned as much as possible, but at the same time premature convergence should be avoided. Avoiding the loss of diversity is especially important in the early stages, while strong pruning is desirable to get parsimonious solutions and improve generalization performance in the final stage.

To balance the parsimony with accuracy dynamically, we fix the error factor at each generation and change the complexity factor adaptively with respect to the error. Each individual of our population consists of a neural tree. Let $E_i(t)$ and $C_i(t)$ denote the error and complexity of ith individual at generation t. The complexity of neural trees can be measured as the sum of the units, weights, and layers. The fitness of an individual i at generation t is defined as follows:

$$F_i(t) \quad = \quad E_i(t) + \alpha(t) C_i(t), \tag{1.36}$$

where $0 \leq E_i(t) \leq 1 \text{ and } C_i(t) > 0$ is assumed.

We control $\alpha(t)$ as follows

$$\alpha(t) = \begin{cases} \frac{1}{N^2} \frac{E_{best}(t-1)}{\hat{C}_{best}(t)} & \text{if } E_{best}(t-1) > \epsilon \\ \frac{1}{N^2} \frac{1}{E_{best}(t-1) \cdot \hat{C}_{best}(t)} & \text{otherwise,} \end{cases} \tag{1.37}$$

where N is the size of training set. User-defined constant ϵ specifies the maximum training error allowed for the final solution.

Note that $\alpha(t)$ depends on $E_{best}(t-1)$ and $\hat{C}_{best}(t)$. $E_{best}(t-1)$ is the error value of the program which had the smallest (best) fitness value at generation $t-1$. $\hat{C}_{best}(t)$ is the size of the best program at generation t estimated at generation $t-1$. It is computed as follows

$$\hat{C}_{best}(t) = C_{best}(t-1) + \Delta C_{sum}(t-1) \tag{1.38}$$

where ΔC_{sum} is a moving average that keeps track of the difference in the complexity between the best individual of one generation and the best individual of the next:

$$\Delta C_{sum}(t) = 0.5\big(C_{best}(t) - C_{best}(t-1) + \Delta C_{sum}(t-1)\big) \tag{1.39}$$

with $\Delta C_{sum}(0) = 0$ (see [14] for more details). $\hat{C}_{best}(t)$ is used for the normalization of the complexity factor. In essence, two adaptation phases are distinguished. When $E_{best}(t-1) > \epsilon$, $\alpha(t)$ decreases as the training error falls since $E_{best}(t-1) \leq 1$ is multiplied. This encourages fast error reduction at the early stages of evolution. For $E_{best}(t-1) \leq \epsilon$, in contrast, as $E_{best}(t)$ approaches 0 the relative importance of complexity increases due to the division by a small value $E_{best}(t-1) \ll 1$. This encourages stronger complexity reduction at the final stages to obtain parsimonious solutions.

Numerical results of this method can be found in [15].

1.8 Conclusion

Our theory of evolutionary algorithms has shown that their dynamic behavior can best be explained by the fitness distribution defined by $p(\mathbf{x})$ and not by the fitness landscape $f(\mathbf{x})$. Many evolutionary algorithms are doing gradient ascent on the landscape defined by the average fitness $W(p)$. We have shown that genetic algorithms can be approximated by evolutionary algorithms using probability distributions. The theory leads to a synthesis problem: finding a good factorization for the search distribution. This problem lies in the center of probability theory. For Bayesian networks numerical efficient algorithms have been developed. We have discussed the algorithm LFDA in detail, which computes a Bayesian network by minimizing the Bayesian Information Criterion.

We have outlined how the theory might be used to synthesize programs from probabilistic prototype trees. The advantage of this approach is its backup by a solid theory. In order to be numerical efficient, additional research is necessary.

Synthesis of complex neural networks is still outside this theory. Here two solutions seem possible: either Bayesian networks are extended to the application domain of neural networks or the structure of neural networks is restricted so that it can be handled in a Bayesian framework.

References

[1]R.R. Bouckaert. Properties of bayesian network learning algorithms. In R. Lopez de Mantaras and D. Poole, editors, *Proc. Tenth Conference on Uncertainty in Artificial Intelligence*, pages 102–109, San Francisco, 1994. Morgan Kaufmann.

[2]B.J. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, Cambrigde, 1998.

[3]D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, 1989.

[4]G. Harik. Linkage learning via probabilistic modeling in the ecga. Technical Report IlliGal 99010, University of Illinois, Urbana-Champaign, 1999.

[5]M.I. Jordan. *Learning in Graphical Models*. MIT Press, Cambrigde, 1999.

[6]S. Kullback and Leibler. On information and suffiency. *Annals of Mathematical Statistics*, 22:76–86, 1951.

[7]H. Mühleinbein and Th. Mahnig. Convergence theory and applications of the factorized distribution algorithm. *Journal of Computing and Information Technology*, 7:19–32, 1999.

[8]H. Mühleinbein and Th. Mahnig. Fda – a scalable evolutionary algorithm for the optimization

of additively decomposed functions. *Evolutionary Computation*, 1999. to be published.

[9]H. Mühlenbein. The equation for the response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997.

[10]H. Mühlenbein, Th. Mahnig, and A. Rodriguez Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5:215–247, 1999.

[11]M. Pelikan, D.E. Goldberg, and E. Cantu-Paz. Boa: The bayesian optimization algorithm. Technical Report IlliGal 99003, University of Illinois, Urbana-Champaign, 1999.

[12]R. Salustewicz and J.H. Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5:123–143, 1997.

[13]G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 7:461–464, 1978.

[14]Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3:17–38, 1995.

[15]Byoung-Tak Zhang, Peter Ohm, and Heinz Mühlenbein. Evolutionary induction of sparse neural trees. *Evolutionary Computation*, 5:213–236, 1997.