On Representation and Genetic Operators in Evolutionary Algorithms

Stefan Droste[*]
LS Informatik II
Universität Dortmund
44221 Dortmund
Germany

Dirk Wiesmann[†]
LS Informatik XI
Universität Dortmund
44221 Dortmund
Germany

Abstract

The application of evolutionary algorithms (EAs) requires as a basic design decision the choice of a suitable representation of the variable space and appropriate genetic operators. In practice mainly problemspecific representations with specific genetic operators and miscellaneous extensions can be observed. In this connection it attracts attention that hardly any formal requirements on the genetic operators are stated.

In this article we first formalize the representation problem and then propose a package of requirements to guide the design of genetic operators. By the definition of distance measures on the geno- and phenotype space it is possible to integrate problem-specific knowledge into the genetic operators. As an example we show how this package of requirements can be used to design a genetic programming (GP) system for finding Boolean functions.

1 Introduction

The application of evolutionary algorithms (EAs) requires as a basic design decision the choice of a suitable representation of the variable space and appropriate genetic operators. The choice of a suitable representation for a given problem and of the appropriate genetic operators requires a basic understanding of the mechanism that makes EAs work. Unfortunately up to now there are only hypotheses on the functioning of EAs. Historically standard representations like binary strings or real-valued vectors with their corresponding genetic operators emerged, which stand in a strong relation to the relevant hypotheses on the working mechanisms of the corresponding EAs.

For real-world optimizing tasks, a trivial translation into standard representations is not always given. One possibility is to fall back on a standard representation by using a suitable mapping between geno- and phenotype space, though the choice of a suitable mapping is not without problems. For instance the coding functions are often non-linear and sometimes not even continuous. Therefore in practice another proceeding established. The literature gives multiple examples of problem-specific representations with special genetic operators, repair mechanisms, and various extensions. The algorithms presented have often very little in common with the canonical forms of EAs, e.g. genetic algorithms (GAs) (see [Hol75]) or evolution strategies (ESs) (see [Rec94, Sch95]). In particular it attracts attention that hardly any formal requirements on the genetic operators are stated.

[*]droste@ls2.informatik.uni-dortmund.de
[†]wiesmann@ls11.informatik.uni-dortmund.de

In this article a set of requirements will be presented towards which the design of representation and operators can be orientated. Through the definition of distance measures on the geno- and phenotype space problem-specific knowledge can be integrated into the genetic operators. The article is two-fold. In the first part the representation problem will be formalized and the set of requirements will be motivated and discussed.

In the second part we show how the requirements can be used to design a genetic programming (GP) system for finding Boolean functions. The designed GP system shows in comparison to other GP systems a remarkable increase in efficiency. For instance it was possible to solve the 12-parity problem with GP for the first time.

2 The Representation Problem

In order to clarify some important notations, we take at first a brief look at the biological model. Afterwards the representation problem of EAs will be formalized.

2.1 The Biological Model

The building plan of living beings is linearly encoded in the deoxyribonucleic acid (DNA). The alphabet of the genetic information in which the plan is written down consists of four components (called nucleotide bases). The entirety of genetic make-up is called genotype. The creation of an organism (phenotype) from its genetic building plan (genotype) is a very complex process. A description of this process is beyond the scope of this article, so we only give a very short overview (see also [Bäc96]).

The carrier of the phenotype are the proteins. Proteins are long chains of amino acids. The finite alphabet of amino acids consists of twenty (natural) acids. Certain chemical interactions between the amino acids cause a three-dimensional folding of the chain. The linear sequence of the amino acid chain is encoded in the following way. Three subsequent nucleotide bases (called codons) in the DNA strand encode a certain amino acid at a time. Since there are $4^3 = 64$ different symbols to encode the 20 natural amino acids, the genetic code is redundant. The biosynthesis of proteins starts with the transcription of the DNA information into a RNA form (messenger ribonucleic acid, m-RNA). The m-RNA transfers the genetic information to the ribosomes, where it is used to synthesize the corresponding protein. The ribosome which consists of ribonucleic acid and proteins performs the mapping from triplets of nucleotide bases to amino acids (translation). The sequence of codons (about 100 to 1000) that encodes a protein is called gene.

The impact of genes on phenotypical features of an organism can not be described by a simple one-to-one correspondence between genes and features. Several genes can have an influence on one phenotypical characteristic (polygeny). On the other hand it is also possible that one gene has an influence on several phenotypical features (pleiotropy). Moreover there can exist sequences of codons (called introns) which do not have any genetic information for the phenotype at all. It is important to state that information is only passed from genotype to phenotype:

$$\text{DNA} \rightarrow \text{RNS} \rightarrow \mathit{Protein}.$$

During the reproduction process the genotype is influenced by mutation and recombination. Each single individual with its own phenotypical characteristics is in an environment with limited resources subject to selection. The better the individual is adapted to the environment the greater is the chance, that the individual lives for a longer while and will be able to generate offsprings and transmit parts of his genotype to his descendants. The indeterministic nature of reproduction leads to a constant generation of new genetic material and so to always varying offsprings. This very short summary of the biological genotype and its translation into phenotype shows already that it is too complicated to copy the biological model exactly.

2.2 Representation and genetic operators in EAs

Within EAs one distinguishes between the genotype space, here the representation of the search space, and the phenotype space, the original variable space (see Figure 1).

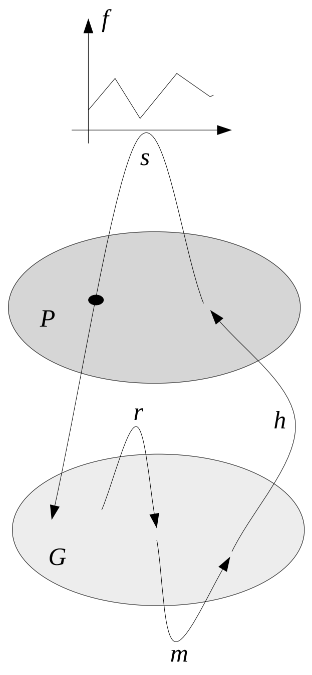

Figure 1: The connection between genotype $\mathcal{G}$ and phenotype $\mathcal{P}$. See text for more information.

The connection between the genotype space $\mathcal{G}$ and the phenotype space $\mathcal{P}$ becomes clearer, when the course of an EA is formalized in a suitable way (the notation follows [Bäc96]).

In the context of EAs it is sufficient to store an individual in form of its genotype, as the geno-phenotype mapping $h : \mathcal{G}^\lambda \to \mathcal{P}^\lambda$ is deterministic and environmental influences are not taken into consideration (yet). That is why we can denote a population at generation t by $Pop(t) \in \mathcal{G}^\mu$. Here μ denotes the number of parents and λ the number of offsprings in a generation. The mapping $h : \mathcal{G}^\lambda \to \mathcal{P}^\lambda$ can be composed simply out of the reduced mapping $h' : \mathcal{G} \to \mathcal{P}$, which represents the geno-phenotype mapping for single individuals:

$$h(g) = (h'(g_1), \ldots, h'(g_\lambda)), \quad g = (g_1, \ldots, g_\lambda) \in \mathcal{G}^\lambda. \tag{1}$$

So the mapping $h' : \mathcal{G} \to \mathcal{P}$ stands for the process of protein biosynthesis (translation) and the development of the phenotype (ontogenesis). Mutation and recombination are working on the genotype space $\mathcal{G}$ only. The recombination operator $r : \mathcal{G}^\mu \times \Omega_r \to \mathcal{G}^\lambda$ generates the genetic building plan for λ offsprings out of the parent population by mixing the parental genetic information. The consideration of the probabilistic influence is done by the probability space (Ω_r, P_r). After recombination the mutation of the λ offsprings by the operator $m : \mathcal{G}^\lambda \times \Omega_m \rightarrow \mathcal{G}^\lambda$ follows. Here (Ω_m, P_m) is the underlying probability space.

The selection creates the next population $Pop(t+1) \in \mathcal{G}^\mu$ from the offsprings of generation $Pop(t)$. The assessment of an individuum is based directly or indirectly on the objective function $f : \mathcal{P} \rightarrow W$ (e.g. $W = \mathbb{R}$). The objective function assesses only the phenotypic expression of the genotype. The selection operator

$$s : \mathcal{G}^\lambda \times \mathcal{P}^\lambda \times \Omega_s \rightarrow \mathcal{G}^\mu \tag{2}$$

makes this connection clear (with probability space (Ω_s, P_s)). So an evolutionary algorithm can be described by the iterated use of the genetic operators and the selection operator. With the auxiliary function

$$\begin{aligned} h^* \quad &: \quad \mathcal{G}^\lambda \rightarrow \mathcal{G}^\lambda \times \mathcal{P}^\lambda, & (3) \\ h^*(g) \quad &:= \quad (g, h(g)) \quad \text{for } g \in \mathcal{G}^\lambda & (4) \end{aligned}$$

the equation

$$Pop(t+1) = s(h^*(m(r(Pop(t), \omega_r), \omega_m)), \omega_s), \tag{5}$$

is valid with

$$\omega_r \in \Omega_r, \quad \omega_m \in \Omega_m, \text{ and } \omega_s \in \Omega_s. \tag{6}$$

Canonical forms of EAs use a binary representation (GAs) or real-valued vectors (ESs) with corresponding mutation and recombination operators. As an example let us take a look at continuous parameter optimization problems. The function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ has to be minimized. For a canonical GA we have the setting:

$$\mathcal{G} = \mathbb{B}^l, \quad \mathcal{P} = \mathbb{R}^n, \quad h' = \Upsilon. \tag{7}$$

Here $\Upsilon : \mathbb{B}^l \rightarrow \prod_{i=1}^{n} [u_i, v_i]$ is a decoding function to decode a binary string into the corresponding rational value. For this the use of a simple base two representation or a Gray code interpretation is widespread.

In comparison for ESs the setting

$$\mathcal{G} = \mathbb{R}^n, \quad \mathcal{P} = \mathbb{R}^n, \quad h' = I \tag{8}$$

is valid. Here I is the identity. So genotype space and phenotype space are identical. In [Rec94] further remarks on the differentiation between genotype space and phenotype space can be found.

From (5) the strong connections and dependencies between genotype space $\mathcal{G}$, phenotype space $\mathcal{P}$, mapping h, recombination r, and mutation m can be seen. How are $\mathcal{G}$, r, m, and h to be chosen, if a certain variable space $\mathcal{P}$ and an objective function $f : \mathcal{P} \rightarrow W$ are given?

One possibility is to choose a standard genotype space ($\mathcal{G} = \mathbb{B}^l$ for GAs or $\mathcal{G} = \mathbb{R}^n$ for ESs) and the corresponding genetic operators of the canonical EAs. The task is then to design a suitable mapping $h' : \mathcal{G} \rightarrow \mathcal{P}$. Since in the past the basic research concentrated upon the standard representations, it is possible in this case to use existing theoretical results (see [Rud97]), e.g. about convergence properties, and recommendations, e.g. for a suitable setting of the parameters of an EA. But the choice of a suitable mapping $h' : \mathcal{G} \rightarrow \mathcal{P}$ is not without problems. An example for this is the continuous parameter optimization with GAs (see [Bäc96]). The used standard GA with coding function ($h' = \Upsilon$) shows in comparison with an ES, which can be used directly for continuous parameter optimization ($h' = I$), clear problems in the convergence speed and reliability (see [BS93, Bäc96]).

One possible explanation is, that the coding function reduces the causality between genotype and object function value. One has to take into account that the coding function (the mapping) is often nonlinear and sometimes even not continuous. We will deal with the term causality in section 3.2 in more detail. In [SKvS97] an approach is presented to formalize the term causality regarding the mapping.

For many real-world problems a canonical mapping in standard representation is not given (e.g. mixed-integer optimization [BS95, SS96], structure optimization [GHM$^+$96]). Take as an example the synthesis of chemical plants. How should the different objects like pumps, heat-exchangers, process units, and the connections between them be represented in this case? A further problem can be the loss of information about the context of "the genotypical atoms" (genes) implied by the coding. Groß et al. [GHM$^+$96] are comparing e.g. a standard representation and a graph representation for heat exchanger networks. The operators of the graph representation are able to use information about the connections between the streams and the heat exchangers. It is difficult to extract this kind of information out of a standard representation.

Nevertheless the use of standard representations was recommended by basic research for a long time. In particular in the area of GAs it was argued, that the binary representation is superior to other representations because of its similarity to the biological model [Hol75]. But this statement is seen more and more critical. This problem will be discussed in section 3. In practice another proceeding can be observed. The literature gives numerous examples of problemspecific representations with specific operators and miscellaneous extensions. These evolutionary algorithms often have very little in common with the canonical forms of EAs.

The choice of $\mathcal{G}$ and the design of r, m, and h' follow a more pragmatic point of view (see e.g. [Mic96, Dav90]). The working mechanisms are not questioned enough and useful requirements on the operators are hardly stated (an exception is the approach of Radcliffe [Rad94, RS95]). Often it seems that it depends only on how the operators roughly behave (recombination: there is an exchange of information; mutation: there are probabilistic modifications). In practice the lack of theoretical results leads to questionable ad hoc solutions that are not thought through to the end. So there are no results about convergence and efficiency available for this kind of EAs.

3 Building Block vs Gradient Diffusion

In order to choose a suitable representation with suitable genetic operators, a basic understanding of the mechanisms of EAs is essential. What are the principles of EAs, how and why are they finding optima? Unfortunately there are only hypotheses on the working mechanisms of EAs available until now. The two best known hypotheses are the building block hypothesis and the hypothesis of gradient diffusion. In the following section we will judge these two hypotheses.

3.1 GAs and the Schema Theorem

The so called schema theorem [Hol75, Gol89] forms the basis of a widespread interpretation of the functioning of GAs. A schema is a subset of strings which are equal at certain string positions. It can be defined as a string over the alphabet $\{0, 1, *\}$. The $*$ serves as a wild card and matches both 0 and 1. So every position in a string that does not equal $*$ is a fixed position. The number of fixed positions is the order of the schema. The defining length of a schema is given by the maximum distance of fixed positions. For a GA with proportional selection, higher reproduction rates for better individuals, one point crossover with fixed crossover probability, and small, fixed mutation rates the schema theorem shows the exponential increase of schemata with above average fitness within the population.

On the assumption that the population contains the globally optimal solution at some time, the solution will only then persist definitely in the population, when neither mutation nor recombination is used. The schema theorem only considers the disruptive effect of mutation. The probability that a schema will be destroyed by mutation is exponentially decreasing with the order of the schema. Whether recombination is destructive or not depends on the position of the crossover point and on the parents. The larger the defining length of a schema, the higher is the probability that the schema will be destroyed by recombination.

From this point of view schemata with a small defining length, small order and above average fitness are of particular interest. The schemata of this kind are often denoted as building blocks (an exact and homogeneous definition of building blocks is still missing). From the representation of the schema theorem the statement is often derived, that building blocks will have an exponential increase in the following generations. In this form the schema theorem plays a decisive role in determining the interpretation on the working method of GAs. Short schemata with small order and above average fitness will be put together by recombination to strings with potentially higher fitness. This interpretation is often denoted as *building block hypothesis*. Simply speaking the recombination operator assembles the "good parts" of the genotype of the involved individuals to a new individual, whose fitness goes beyond the fitness of its parents in the ideal case. The schema theorem is valid for other than binary alphabets, too. The strong preference of binary alphabets arises from the observation, that a binary alphabet offers the maximum number of schemata under all discrete alphabets. It is often referred to the similarities between the binary representation and the biological model, too. The manipulation of bit strings in GAs seems not very unsimilar to the recombination and mutation of chromosoms (in contrast to discrete recombination and mutation of real-valued vectors in ESs).

But the schema theorem and its conclusions are often criticized. The discussion on the schema theorem should not be repeated at this place. Instead we give a listing of the most frequently used subjects of criticism and the corresponding bibliographical references:

- The schema theorem says nothing about the convergence and so it seems not to be appropriate to explain the functioning of GAs as function optimizers [Bey95, Bey97, Rud97].
- Mainly the harmful effects of mutation and recombination are considered [Sch95].
- In practice only populations of finite size are available [Bäc96].
- Proportional selection is not of interest for function optimization [Rad97].
- The building block hypothesis seems to assume the separability of the objective function [Rad97].
- The argument of implicit parallelism and the preference of binary alphabets is not valid [Ant89, Bey97, FG97, WM97].

3.2 EAs=gradient strategies?

In biology mutations are appearing very rarely nowadays. The changes of the genetic material are caused by several factors. The simplest form of mutation is a replication error switching a specific base pair in the original DNA strand to the other possible base pair in the copy. For the human genom Gottschalk gives a probability of $6 \cdot 10^{-6}$ up to $8 \cdot 10^{-6}$ for the occurrence of a spontaneous mutation of a specific gene [Got89].

Moreover the genetic code seems to be build up in a way that mutations can only cause very small changes of the genotype. Because of their chemical bonding properties the amino acids can be divided into different groups. Is an amino acid replaced by a amino acid out of the same group, the change in the folding of the accessory proteins will be smaller than by amino acids with strongly varying bonding properties. The genetic code is build up in a way that a change of a nucleotide base within a code word will more frequently lead to a amino acid out of the same group such as the original code word [Rec73].

In Section 2.1 we have already pointed out the redundancy of the genetic code. Is a code word changed by mutation the new code word does not necessarily code another amino acid. In that case mutation has no direct effect on the phenotype.

This indicates that evolution gropes its way with mainly small steps in a smooth fitness landscape. Only small changes in the genotype space happen, which lead due to a suitable encoding and a special translation process to only small changes in the phenotype space. Large movements in the phenotype space caused by mutations are possible but happen very rarely. The transition from an individual to a descendant with a very similar phenotypical appearance will generally result in a small change in the fitness.

Rechenberg [Rec94] pointed out that one can see here the principle of strong causality (similar causes have similar effects). He sees in the existence of the strong causality (between genotype space and fitness) the fundamental basis for the functioning of evolution[1]:

> "Strong causality is the guiding principle for all human action. Strong causality is the basis of evolution also."
>
> "Without strong causality the optimizer gets lost in the immensity of the high-dimensional space"

The term "smooth fitness landscape" must also be seen in the context of strong causality. The high-dimensional fitness landscape has to be made in such a way that small changes in position result only in small changes of the height.

To Rechenberg evolution strategies (ESs) with their small (groping) steps in a world (fitness landscape) of strong causality are doing nothing else than exploring the gradient. The population is then simply running down the gradient found. The strength of ESs is established by the fact that the gradient will not be followed exactly, rather a gradient diffusion is taking place (caused by setting the new search points without any direction). Because of that the danger of convergence to the next local optima is reduced. Accordingly GAs act in a very fuzzy way by following the gradient in the $\mathbb{R}^n$ (see Section 4).

But also this approach to explain the functioning of EAs as a gradient diffusion in a world of strong causality is not without questions. On the one hand ESs have shown their ability to work with good performance on even very noisy object functions [Bäc96, BH94]. And GAs are still working without considering the guiding principle of strong causality directly (see Section 4 and [SKvS97]). On the other hand the connection between genotype space and phenotype space is not always as obvious as indicated in section 3.2. In biology there is not always a strong causality between genotype and phenotype with respect to fitness. From comparative molecular genetics there are some indications that there need not be a strong connection between genetic and morphological evolution. The genotype can be object of strong

[1]Translated from the german original by the authors.

changes without observing consequences on the phenotype. But also the other case, a strong change of the phenotype caused by a small genetic change can be observed. Not least in the last 30 years the research to nonlinear dynamics has shown that the world view based on strong causality only is incomplete.

4 Recombination and Mutation in EAs

We have already seen that the biological evolution cannot be transferred and imitated directly because of its complexity. Moreover the biological information processing has not been examined to such an extend so far to derive knowledge directly from biology to guide the design of the genetic operators and the geno-phenotype mapping. Therefore, EAs are a strong abstraction of a few mechanisms of the biological evolution considered to be pivotal. But there are varying interpretations of these mechanisms. There are only hypotheses of the working mechanisms of EAs.

In the following we will first take a closer look at recombination/crossover and mutation in GAs and ESs. Here we do not look at the implementation of the operators (we assume that this is known) as described many times [BFM97, Bäc96, Fog95, Sch95, Gol89], but we have a more general discussion on the concepts and the (possible/intended) mechanisms of the operators. Afterwards we propose requirements towards which the design of the genetic operators can be orientated.

4.1 Recombination

In EAs recombination is used to generate a new individual (descendant) out of two or more individuals (parents). The genotype of the descendant is put together out of the parental genotypes according to specific rules.

GAs often use a bit string representation as genotype. The crossover operator (e.g. one-point crossover, multi-point crossover) generates the offspring by mixing parts of the parental genotypes. This procedure does not seem to be unsimilar to the recombination in biology. The idea that useful segments (parts of the genotype) of different parents will be combined to an offspring with better genetic information (bit combination) by means of crossover suggests itself. We will not go into the open discussions round the building block hypothesis (see Section 3.1). But an example from [Sch95] (see p. 155ff) should be helpful to illustrate the differences to recombination used in ESs. The crossover in GAs can lead to an erratic behavior in the phenotype space. The recombined offspring of two parents that are closely related in the phenotype space may largely deviate from both parental positions there. Table 1 shows a situation with two parents producing two offsprings by means of two point crossover. The bit string has length eight and the two phenotypic variables in the range $[0, 15]^2$ are encoded in the standard Boolean form. In real life such extreme deviations of an offspring from its parents are very rare.

	genotype		phenotype	
parent 1	0111	1100	7	12
parent 2	**1000**	**1011**	8	11
offspring 1	0**000**	**10**00	0	8
offspring 2	**1**111	11**11**	15	15

Table 1: The effect of two point crossover within a GA on the phenotype.

Within ESs the effect of recombination is limited. For the two-dimensional case Figure 2 ([Bäc96], p. 75) shows schematically the possible results of different re-

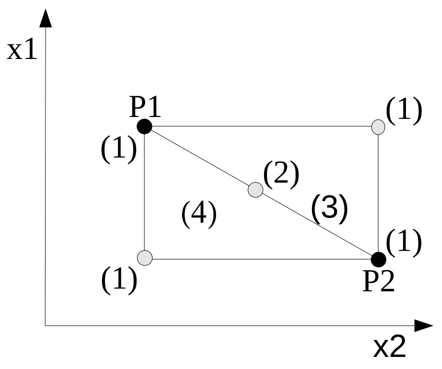

Figure 2: 2-dimensional schema of different recombination mechanisms.

combination mechanisms within ESs. Here P_1 and P_2 indicate the position of the two parental object variables. Since the mapping $h : \mathcal{G} \rightarrow \mathcal{P}$ is the identity, this example is valid for both the phenotype space $\mathcal{P}$ and the genotype space $\mathcal{G}$. Note that no offspring generated from P_1 and P_2 by means of recombination can leave the n-dimensional hyperbody formed by the parent population. The offsprings must always have a certain phenotypic similarity with their parents. Discrete recombination can only reach the corners (indicated (1) in the figure) of the rectangle. Intermediate recombination generates an offspring at the center (2) of the rectangles' diagonal lines. Generalized intermediate recombination can reach points located anywhere on the diagonal line (3). Finally panmictic generalized intermediate recombination allows offsprings located at an arbitrary point somewhere within the rectangle (4) (here the parents are still restricted to P_1 and P_2).

The working mechanism of crossover within GAs can be induced by the building block hypothesis. But what may be the benefit, when the offspring has a phenotypic similarity with its parents? From theoretical research concerning (μ, λ)- and $(\mu/\mu, \lambda)$-ES Beyer [Bey94] tries to derive basic principles of ESs, especially of mutation and recombination.

The examination of the normalized progress rate $\varphi^* := \varphi \frac{n}{R}$ for the sphere model shows that the progress (positive or negative) is composed out of a progress gain and a progress loss (*Evolutionary Progress Principle (EPP)*). The progress rate $\varphi := E\{\Delta R\}$ is the expected distance-to-optimum change ΔR observed from generation g to generation $g + 1$. So mutation has a useful part and a harmful part. Results from the $(\mu/\mu_I, \lambda)$-theory show, that intermediate recombination reduces the harmful part (progress loss) by a factor of $\frac{1}{\mu}$. As the basic principle of intermediate recombination the so-called *Genetic Repair (GR)* [Bey94] is identified. For dominant recombination (global discrete recombination) the principle reads: "*Mutation Induced Species by Recombination (MISR)*".

The application of dominant recombination and of mutation to generate an offspring population can be replaced by a special surrogate mutation of the center of mass ("wild-type") of the parent population. Besides Beyer concludes an implicit GR for $(\mu/\mu_D, \lambda)$-strategies (see [Bey94]). There are clues that the EPP, MISR and GR principles could be transferred to EAs.

4.2 Mutation

Originally mutation was only of small importance within GAs. Mainly mutation served to reintroduce correct bits which were lost by crossover into finite populations. But recent studies have shown, that the importance of mutation should not be underrated. It is often helpful to begin with a large mutation rate which is decreasing over the course of evolution. The effects of mutation on the phenotype are not of special interest in the field of GAs. We have already indicated in Section 2.2 that it is not advisable to ignore the connections between phenotype and genotype. Different mappings can have a varying influence on the performance of a GA. For instance the Gray code which has the property that neighboring values differ in one bit position only has some advantages over standard base two coding [CS88, Bäc96]. The normal binary code probably suffers from Hamming cliffs. Depending on its position a single bit flip (smallest possible mutation concerning the genotype) can cause a small but also a very large change on the level of phenotype[2] [Bäc96, Sch95]. But really problematic is the situation when a small change of the phenotype can only be achieved by a great change of the genotype, so that an EA has to wait for a long time if it prefers small changes.

Within ESs mutation is a very important operator. The mutation operator must obey some rules:

- The mutation has no bias, i.e. without selective pressure resp. under a constant objective function mutation makes a pure *random walk*.
- Smaller mutations occur more often than large ones.

Since within ESs for real-valued optimization problems the genotype space coincides with the phenotype space the identity ($h = I$) can be used as a mapping. Analogously GAs have the advantage that no mapping is needed when trying to solve problems which have a natural binary encoding like combinatorial problems. So in both cases the extent of a mutation is related to both the genotype space and the phenotype space. Since in both spaces the same distance measure (metric) is used, it is additionally valid:

- Small changes of the genotype lead to only small changes in the phenotype (regarding the Euclidean distance). Being more precisely the changes are identical in both spaces.

5 Requirements on Recombination and Mutation Operators: A Proposal

We assume that an optimization task is given, so that the function

$$f : \mathcal{P} \to \mathbb{R} \tag{9}$$

has to be optimized. The space $\mathcal{P}$ is already built in a way, that the objective function f can be evaluated efficiently. In general there are two approaches to tackle this task with EAs.

1. A standard EA is used (e.g. a GA). As genotype space e.g. $\mathcal{G} = \mathbb{B}^l$ will be chosen and a mapping $h : \mathcal{G} \to \mathcal{P}$ will be defined. After that the canonical operators can be applied.

[2]Even for a Gray code a single bit flip may cause large changes to the corresponding integer values [Bäc96].

2. The representation is designed as close as possible to the characteristics of the phenotype space. The genotype space then needs special genetic operators which are adjusted to the genotype space.

The first approach suffers from using a mapping which usually induces additional nonlinearities together with the genetic operators, so that the change of the genotype allows no statement about the change in fitness. With the second approach one tries to avoid the problem by choosing a representation which is very similar to the phenotype space and designing operators which allow some insight into the working mechanisms and using pre-knowledge. Since no formal requirements on the genetic operators let alone design instructions are available, a bad proliferation in designing problem-specific EAs can be observed (see Section 2.2).

In general EAs are used for parameter optimization. In this connection it is important to look not only at the biological model in order to identify new principles which can possibly enhance the performance of EAs. But it is also helpful to consider this problem from a more mathematical point of view, because the balance between biological inspiration and mathematical foundation gets lost to the disadvantage of the latter. For an EA as a optimization algorithm the following questions are of importance:

- Can each permissible solution be reached from an arbitrary initial population?
- Are the operators (mutation, recombination) generating an additional bias which superimposes the real selection process?

Unfortunately very often nobody goes to the trouble of answering these questions when designing a problem-specific EA. Jones and Forrest (see [JF95]) introduced the so-called *fitness distance correlation*, which measures the relation between fitness and distance to next optimum for an objective function in order to predict the behaviour of EAs for this function. But as their definition of distance is not directly related to the used genetic operators, their method, although its idea is related to ours, shows no concrete guidelines how to design genetic operators.

In the following we will present some requirements on genetic operators and the mapping. Our aim is here to support a smooth motion of the search process through the fitness landscape. So the requirements are more orientated towards the gradient diffusion hypothesis than towards the building block hypothesis. But neither should this be a final judgment, nor it is said that the gradient diffusion describes the evolutionary search completely. Though there are some pointers that representations and operators supporting a smooth motion through the fitness landscape have an advantage over that kind showing an erratic behavior [Bäc96, SKvS97]. Such an erratic behavior may have the advantage of reducing the danger of premature convergence to a suboptimal solution. However further advantages are hard to find especially when looking at the search in high dimensional space.

It is important to mention once again that causality has to exist between the genotype space and the objective function (the fitness function). The genetic operators are only working on the genotype space and selection considers only the fitness of the individuals. With this in mind to find an optimal representation for a problem is at least as difficult as to find the global optimal solution. The representation is then only suitable for the problem at hand, at most for a specific problem domain. Similar to the *No-Free-Lunch Theorem* (see [WM97]) Radcliffe and Surry have shown that all representations are equivalent when their behavior is considered on average over all possible objective functions [RS95].

We are assuming that we have certain information about the phenotype space concerning a specific problem domain. This pre-knowledge will be used to form a distance measure. This distance measure shows the similarity between two phenotypes regarding a certain mark of quality, i.e. two elements of the phenotype space with small distance should have similar quality. This mark of quality should be strongly connected to the objective function to be optimized. The pre-knowledge should be used as good as possible by formulating the requirements on the operators on the basis of the distance measure. The performance of the EAs will then mainly depend on the suitability of the distance measure, i.e. on the quality of the available pre-knowledge about the problem domain.

But we have already seen that we cannot ask for more [WM97, RS95, FG97]. It is of no use, when the object function is used directly to obtain the distance measure. In this case the optimum must be found before the development of the operators takes place. Moreover the EAs should be designed for a whole problem domain of suitable size and not only for one objective function. Considering this aspect we are assuming that a suitable metric $d_{\mathcal{P}} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbb{R}_0^+$ on the phenotype space $\mathcal{P}$ is chosen. A metric must fulfill the following requirements:

Definition 1 (Metric) *A metric on a set X is a mapping $d : X \times X \rightarrow \mathbb{R}$, such that $\forall \vec{v}, \vec{w}, \vec{x} \in X$:*

$$d(\vec{v}, \vec{w}) = 0 \quad \Leftrightarrow \quad \vec{v} = \vec{w} \tag{10}$$

$$d(\vec{v}, \vec{w}) \leq d(\vec{v}, \vec{x}) + d(\vec{w}, \vec{x}). \tag{11}$$

If d is a metric on X, so (X, d) is called a *metric space*. In a metric space for all $\vec{v}, \vec{w} \in X$ follows:

$$d(\vec{v}, \vec{w}) = d(\vec{w}, \vec{v}) \tag{12}$$

$$d(\vec{v}, \vec{w}) \geq 0 \tag{13}$$

In the following both the genotype space and the phenotype space are assumed to be finite spaces. To extend our proposal on infinite spaces, one can either extend our formalization to infinite spaces by using continous probability spaces or restrict the infinite spaces to finite aprroximations (e.g. by using fixed-length representations of real numbers with finite precision as it is used in every computer).

5.1 Mapping

Assuming that the mapping h is injective the metric $d_{\mathcal{P}}$ induces automatically a metric $d_{\mathcal{G}} : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$ on the genotype space through

$$d_{\mathcal{G}}(g_1, g_2) := d_{\mathcal{P}}(h(g_1), h(g_2)) \qquad \forall g_1, g_2 \in \mathcal{G}. \tag{14}$$

If a genetic operator causes a small change regarding $d_{\mathcal{G}}$, this will result in just a small change in the phenotype space. The reversal is valid on condition that h is bijective. In general the assumption that h is injective does not seem to be useful. If one thinks of neutral mutations or introns it is not impossible that two different genotypes are coding exactly the same phenotype. On the other hand two perfect identical individuals require that the ontogenesis of the individuals has to take place under exactly the same environmental factors. If one rules out such constant environmental factors the assumption that h is injective seems possible again. Of course the mapping does not consider the effect of environmental factors so far.

If h is not injective (so (14) is not valid), the following requirement can be stated on the mapping:

Requirement H 1

$$\begin{array}{rcl} \forall u, v, w \in \mathcal{G} : & & \\ d_{\mathcal{G}}(u,v) & \leq & d_{\mathcal{G}}(u,w) \\ & \Leftrightarrow & \\ d_{\mathcal{P}}(h(u),h(v)) & \leq & d_{\mathcal{P}}(h(u),h(w)) \end{array} \tag{15}$$

If h is bijective, with (14) requirement H 1 is automatically valid.

5.2 Recombination

For the recombination operator it should at first be guaranteed that the offspring has a certain similarity with its parents. In the following we look at the way an individuum is created by recombination. Instead of the general form it suffices to present the reduced recombination operator $r' : \mathcal{G}^2 \times \Omega'_r \to \mathcal{G}$, with a finite probability space $(\Omega_{r'}, P_{r'})$. The recombination operator $r' : \mathcal{G}^2 \times \Omega_{r'} \to \mathcal{G}$ assigns with probability

$$P_{r'}(r'(u,v)) := P_{r'}(\{\omega \in \Omega_{r'} \mid r'(u,v,\omega) = w\}) \tag{16}$$

two parents $u, v \in \mathcal{G}$ an offspring $w \in \mathcal{G}$ (with $\omega \in \Omega_{r'}$).

The first requirement on the recombination operator has the following form:

Requirement R 1

$$\begin{array}{l} \forall u, v \in \mathcal{G}, \omega \in \Omega_{r'}, z = r'(u,v,\omega) : \\ \max(d_{\mathcal{G}}(u,z), d_{\mathcal{G}}(v,z)) \leq d_{\mathcal{G}}(u,v) \end{array} \tag{17}$$

So the genotype of the offspring is not allowed to be further away from his parents (regarding $d_{\mathcal{G}}$) than the parents from each other.

Moreover the recombination operator should not generate an additional bias independent of the parents. This will be illustrated by Figure 2. Using a discrete recombination an offspring can reach the corners of the rectangle indicated (1). But if one of these points has now a higher probability to be reached by recombination than the rest of the points, the recombination operator induces a direction independent of the parents. So a recombination operator should fulfill additionally:

Requirement R 2

$$\begin{array}{l} \forall u, v \in \mathcal{G}, \forall \alpha \geq 0 : \\ P_{r'}(d_{\mathcal{G}}(u, r'(u,v)) = \alpha) = P_{r'}(d_{\mathcal{G}}(v, r'(u,v)) = \alpha) \end{array} \tag{18}$$

For recombination operators using more than two parents the requirements can be formulated in an analogous way.

On account of pre-knowledge, experiences or implementation details (see example of use) the requirements R 1 and R 2 can be extended or specialized. Intermediate recombination, which is often used in ESs, e.g. limits the space of offsprings available by panmictic generalized intermediate recombination to a special area.

5.3 Mutation

The mutation operator $m : \mathcal{G}^\lambda \times \Omega_m \to \mathcal{G}^\lambda$ can basically work on all λ individuals at the same time. Since the mutation of an individual is independent from the other individuals it is sufficient to look at the reduced mutation operator $m' : \mathcal{G} \times \Omega_{m'} \rightarrow \mathcal{G}$ with the finite probability space $(\Omega_{m'}, P_{m'})$. With probability

$$P_{m'}(m'(u) = v) := P_{m'}(\{\omega \in \Omega_{m'} \mid m'(u, \omega) = v\}) \tag{19}$$

the mutation operator m' generates out of an element $u \in \mathcal{G}$ a new element $v \in \mathcal{G}$.

The first requirement should guarantee that from each point $g \in \mathcal{G}$ each other point $u \in \mathcal{G}$ can be reached in one mutation step:

Requirement M 1

$$\forall g, u \in \mathcal{G} : \qquad P_{m'}(m'(g) = u) > 0. \tag{20}$$

This requirement can be weakened by replacing the mutation with a finite sequence of mutations (where $m' \circ m'(g, \omega) = m'(m'(g, \omega))$) [3]:

$$\begin{aligned} &\forall g, u \in \mathcal{G}, \quad \exists a \in \mathbb{N} \text{ with } a < \infty : \\ &P_{m'}(\underbrace{m' \circ \ldots \circ m'}_{a}(g) = u) > 0. \end{aligned} \tag{21}$$

Moreover small mutations (measured with $d_{\mathcal{G}}$) must occur more often than larger mutations:

Requirement M 2

$$\begin{aligned} &\forall a, b \in \mathbb{N}_0^+, \quad \forall g \in \mathcal{G} : \\ &a < b \quad \Rightarrow \quad P_{m'}(d_{\mathcal{G}}(g, m'(g)) = a) > P_{m'}(d_{\mathcal{G}}(g, m'(g)) = b) \end{aligned} \tag{22}$$

For a given $g \in \mathcal{G}$ all genotypes $u \in \mathcal{G}$ which have the same distance to g should have the same probability to be reached by mutation of g:

Requirement M 3

$$\begin{aligned} &\forall g, u, v \in \mathcal{G} : \\ &d_{\mathcal{G}}(g, u) = d_{\mathcal{G}}(g, v) \quad \Rightarrow \quad P_{m'}(m'(g) = u) = P_{m'}(m'(g) = v). \end{aligned} \tag{23}$$

This requirement guarantees that the mutation operator does not induce a bias (on its own). Since at the beginning of search there is no information on the position of the global optimum the mutation should not give the population a certain direction. By working with a dynamic adaptation of step-size (e.g. self-adaptation of $n > 1$ step-sizes) requirement M 3 can be violated. By learning an internal model of the topology the preference of a certain search direction can be useful. But whenever there is no suitable information about the topology of the search space available anymore, it should be guaranteed that a state can be reached in which M 3 is fulfilled.

5.4 Metric Based Evolutionary Algorithm (MBEA)

Since all requirements are based on the definition of a suitable metric on genotype and phenotype space, an EA which fulfills the requirements H 1, R 1, R 2, M 1, M 2 and M 3 for a given metric and mapping is called *Metric Based Evolutionary Algorithm* (MBEA).

[3] In this case it must be possible that elements with lower fitness can be preferred over elements with higher fitness.

6 Metric Based Genetic Programming

If one looks at all the requirements that define a MBEA, it is not obvious how a MBEA can be build. Therefore, we will give an example of an *Metric Based Genetic Programming (MBGP)*-System in this section to show how crossover and mutation operators conforming with our requirements can be build. The paradigm of genetic programming (GP) (see [Koz92, Koz94] for a detailed introduction to GP) was used, as most existing GP-systems use crossover and mutation operators, whose results in the phenotype space are very unclear most of the times.

Look, for instance, at the GP-system presented in [Koz92], which has become a standard for many other GP-systems afterwards: the elements of the phenotype-space are programs, which are represented by S-expressions, i.e., tree-structures with functions at the inner nodes and terminals at the leaves. Both, function and terminal set (which has to include the set of input variables), are predefined and an S-expression is evaluated by bottom-up evaluation of the tree structure. In most GP-systems the recombination of two S-expressions chooses one node in every parent and exchanges the subtrees starting at these nodes. Hence, the functional behaviour of the resulting S-expression is dependent on the function and terminal set, the chosen nodes, and the form of the S-expressions, which are above the chosen nodes in the parental S-expressions.

Even if the two parental S-expressions have a rather similar functional behaviour, the functional behaviour of the child can vary drastically from that of the parents and there is no general method known for predicting it. Therefore, looking at our requirements, most GP-systems do not fulfill properties R 1 and H 1, as a child can differ more from the parents than the parents differ from each other in the phenotype-space. If we assume that the fitness-landscape on the phenotype-set is rather smooth, it probably would be advantageous if the GP-system searches "between" two good parents, as one can assume that other good elements of the phenotype space will exist there. This holds, because a smooth fitness landscape implies that in the neighborhood of points with similar fitness other points with approximately the same fitness lie.

In our MBGP-system no S-expressions are used. Instead a data structure is used which is better appropriated for the problem being attacked and which allows us to implement mutation and recombination operators that conform with the requirements defining an MBEA.

The task of our MBGP-system is to solve the following often used benchmark-problem: find a representation of an unknown function, which is given by its complete training set, i.e., for every input the correct output is given. Although this task has no practical applications, if a representation of a completely specified function can be easily build, it is often used to compare the quality of search methods in the space of computer programs (see for instance [Koz92, Koz94, GR97, Che97]). In our case, the unknown function g is a Boolean function with n inputs and one output, i.e. $g : \{0,1\}^n \to \{0,1\}$. The training set T simply is the set $\{(x, g(x)) \mid x \in \{0,1\}^n\}$.

Every element $f : \{0,1\}^n \to \{0,1\}$ of the phenotype set will get the fitness $F(f) := |\{x \in \{0,1\}^n \mid f(x) = g(x)\}|$, i.e., the number of inputs, where f has the same output as the unknown function g. Hence, the unknown function g is the only maximum of the fitness function F. The task of our MBGP-system is to find a representation of this maximum.

As we have stated above, the data stucture used for representation plays a major role for the success and efficiency of every search method. In our MBGP-system we will not use S-expressions for representation, but a special data structure for Boolean functions, called *Ordered Binary Decision Diagrams (OBDD)* (see [Bry86]). OBDDs allow polynomially-sized representation in n of many practically important Boolean functions (because there are 2^{2^n} Boolean functions on n input variables no data structure can represent every Boolean function in polynomially bounded space) and many operations on OBDDs can be done in polynomially bounded time with regard to the size of the involved OBDDs. Some of these efficient operations are used in the implementation of our mutation and recombination operators.

Before our MBGP-system is described in detail, the next subsections will give a short introduction to OBDDs and an overview of existing GP-systems using OBDDs.

6.1 Ordered Binary Decision Diagrams

Ordered Binary Decision Diagrams (OBDDs) are a data structure for the efficient representation of Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$ (see [Bry86]). Given a permutation π of the input variables $x_1, \ldots, x_n$ an OBDD is a directed acyclic graph with one source and two sinks, fulfilling the following properties:

- every inner node is labeled by one of the input variables $x_1, \ldots, x_n$ and has two outgoing edges, leading to the 0- and 1-successor, resp.,
- there are exactly two nodes without outgoing edges, the so-called 0- and 1-sink, resp.,
- every edge between two inner nodes must obey the permutation π, i.e., if there is an edge from a node v to a node u, the label of v has to be smaller than the label of u with regard to π.

How does a given OBDD correspond to a Boolean function, i.e., how can we evaluate the function f represented by an OBDD when an input $a \in \{0,1\}^n$ is given? To do this, just repeat the following recursive procedure starting at the source until a sink is reached: if the label of the actual node is x_i go to the a_i-successor. The value $f(a)$ is 0, if the 0-sink is reached, and 1, if the 1-sink is reached. As the label of the nodes must increase (with regard to π) when going to a successor-node, this procedure is finished after at most n steps. This is one advantage of OBDDs in comparison to S-expressions: they can be evaluated in time $O(n)$, while an S-expression S needs time $\Omega(|S|)$, which is worse for all non-trivial S-expressions containing at least all input-variables as terminal symbols. Figure 3 shows an OBDD, where the inner nodes are represented by circles and the sinks by squares (to get a better overview three sinks are shown, although only two sinks are stored in memory).

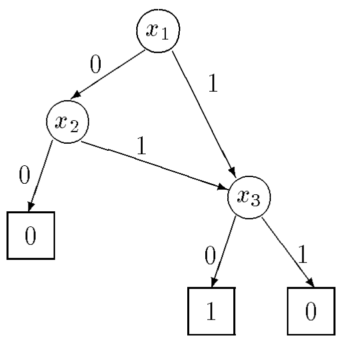

Figure 3: OBDD representing the function $f(x_1, x_2, x_3) = x_1\overline{x}_3 \vee \overline{x}_1 x_2 \overline{x}_3$, where $\pi = (x_1, x_2, x_3)$.

The variable ordering π has an essential influence on the size of an OBDD. There are functions that have OBDDs of linear or exponential size depending on the variable ordering π. As even the existence of a polynomial time approximation scheme for finding the optimal variable ordering implies $P = NP$ (see [Sie98]), it is very unlikely that any efficient scheme for finding a variable ordering being nearly optimal exists.

But why are OBDDs nevertheless the state-of-the-art data structure for Boolean functions, if they have this disadvantage? First of all, many practically important functions can be represented with OBDDs of polynomial size. The second point is the existence of efficient algorithms for many important operations. Of special interest for our MBGP-system are algorithms for counting the number of one-inputs and for the synthesis of two OBDDs (see [Bry86]).

The first problem consists of determining $|\{x \in \{0,1\}^n \mid f(x) = 1\}|$, when an OBDD D representing f is given. This problem can be solved in linear time in the size of D (see [Bry86]). The synthesis of two OBDDs D_1 and D_2 (representing f_1 and f_2) is the construction of an OBDD for the function $f_1 \otimes f_2$, where $(f_1 \otimes f_2)(x) := f_1(x) \otimes f_2(x)$ and $\otimes : \{0,1\}^2 \rightarrow \{0,1\}$ is the synthesis-function. An algorithm for the synthesis is known, which needs time $O(|D_1||D_2| \log(|D_1||D_2|))$ (see [SW93]). Hence, if both OBDDs have polynomial size, both algorithms work in polynomial time.

Another big advantage of OBDDs is the existence of *reduced* OBDDs. For a given function f and variable ordering π the reduced OBDD representing f is the unique OBDD from the set of all OBDDs representing f, having the minimal number of nodes (see [Bry86]). A given OBDD D_1 can be reduced in time $O(|D_1|)$ and using a hash-table and bottom-up construction one can implicitly build only reduced OBDDs, which is very space-efficient and used in our MBGP-system.

6.2 Existing GP-Systems using OBDDs

This subsection is intended to give a short overview of the different GP-systems that use OBDDs for the representation of programs. As we are especially interested in the effects of crossover and mutation, we will emphasize this point in the discussion of the GP-systems.

The advantages of OBDDs for GP were used by Yanagiya for the first time (see [Yan95]). In his work OBDDs are the form of representation and the task of his GP-system is to find a representation of a Boolean function given by a complete training set. To determine the fitness of an OBDD D, he uses the follwing method: in the beginning of the run the OBDD D^* of the unknown function is build, which is possible, as the training set includes all inputs. For fitness evaluation an EXOR-synthesis of D and D^* is performed. The number of inputs leading to the output 1 is counted afterwards. This number is exactly the number of inputs, where D and D^* disagree. Using the algorithms mentioned in section 6.1, the evaluation therefore needs only time $O(|D||D^*| \log(|D||D^*|))$ in comparison to time $\Omega(2^n \cdot |S|)$, when using an S-expression S.

This time decrease made his GP-system the first, that found the 20-multiplexer. But one has to be aware that this method cannot be used in practical applications, where the output of the unknown function is not known for all inputs. Although this method of evaluation is used in our MBGP-system for time efficiency, too, we do not compare our runtime to that of other algorithms. Furthermore, it is important, that this method of evaluation has no influence on the search process itself.

The recombination described in [Yan95] is used in our MBGP-system, too, but without the implicitly used mutation. Because of this implicit mutation, a child can disagree with its parents even on inputs where both parents agree. Although this is done with low probability, it can change the functional behaviour of the child to a rather high degree.

Furthermore, the OBDDs in [Yan95] can only have a maximum breadth, i.e., when the maximum number of nodes with the same label is reached, new nodes with this label, which are to be constructed, are not build anymore. Instead the edges leading to this unbuild node are redirected to existing nodes with another label. Naturally, this makes the effect of recombination very hard to predict. Mutation is not used in this system, if we do not look at the implicit mutation during recombination.

The GP-system described in [SHIK96] uses OBDD-like structures for representation to find a representation of an unknown function given by a complete training set. The recombination and mutation operators work similar to the operators described in [Koz92, Koz94] for S-expressions, i.e., they swap substructures of the parental OBDDs or create new substructures. As there are no measures taken to guarantee that the resulting structures are still valid OBDDs, selection pressure is used to make the results OBDD-like by giving OBDD-like structures a higher fitness. Using this method the parity-function can be found in [SHIK96] for up to six variables. No investigations about the behaviour of mutation and recombination are made and are likely to be difficult, because the genetic operators are not made especially for OBDDs.

Another GP-system using OBDDs, whose genetic operators work similar to the standard operators described in [Koz92, Koz94], is presented in [Dro97]. Here all individulals, i.e., Boolean functions, are represented by reduced OBDDs. But the task of this GP-system is to approximate an unknown Boolean function, which is given only by an incomplete set of training examples. Recombination and mutation in [Dro97] swap substructures of the parental OBDDs and create random substructures in an OBDD, resp. Although special measures are taken to guarantee that the resulting structures are still valid OBDDs, the effect of recombination and mutation on the functional behaviour is unclear.

The main advantage of OBDDs in the system described in [Dro97] is the possibility to build OBDDs, being consistent with the training set. In the initial generation this is done explicitly. As in all further generations OBDDs, which are not consistent with the training examples, are replaced by one of their parents, only consistent OBDDs are used. This allows it to concentrate only on the size of the OBDDs to determine their fitness, i.e., the GP-system tries to find a consistent OBDD with minimal size. This strategy results in rather good generalizing Boolean functions (i.e., they are similar to the unknown function on many of the inputs not in the training set). In [Dro98] the effect of minimzing the size of a program on its generalization properties is investigated under a more theoretical viewpoint.

All in all, the GP-systems with OBDDs, which are used so far, use genetic operators without investigating their effects on the search process. No requirements on the behaviour of the genetic operators are made: in most cases the authors are satisfied, if crossover and mutation exchange information and create random information, respectively.

6.3 A concrete MBEA

We have already seen that the requirements for a MBEA are not fulfilled by most existing GP-systems working with S-expressions or OBDDs. In the following we will present a GP-system in detail that fulfills these requirements, thereby showing that they can be fulfilled. We start with a metric on the phenotype space, so that the fitness landscape over the phenotype space is rather smooth, if one uses this metric to define neighborhoods. So the metric reflects our knowledge of the problem we try to solve.

Then a rough outline of the GP-system is given, which resembles a (μ, λ)-ES. This outline is filled by giving detailed descriptions of the recombination and mutation operators. In order to show that this system is a MBEA, we prove that our requirements on the mutation and recombination operators are fulfilled.

6.3.1 Metric

Our GP-system shall try to find an unknown Boolean function $g : \{0,1\}^n \to \{0,1\}$ and we measure the fitness of an individual $f : \{0,1\}^n \to \{0,1\}$ by counting the number of inputs, where f and the unknown function g agree. Hence, an appropriate metric $d_{\mathcal{P}}$ is the function, that counts the number of inputs, where the two measured functions disagree:

$$d_{\mathcal{P}}(f,h) := |\{x \in \{0,1\}^n \mid f(x) \neq h(x)\}|.$$

This is a metric, which can be easily seen, when realizing that it is equivalent to the Hamming distance on the space of all bitstrings of length 2^n. Because two functions f and h can differ in their fitnesses $F(f)$ und $F(h)$ by $d_{\mathcal{P}}(f,h)$ at most, the fitness landscape is smooth, i.e. neighbouring functions have similar fitness.

On the genotype space, i.e. the space of concrete representations, we use the analogous function $d_{\mathcal{G}}$ defined by

$$d_{\mathcal{G}}(u,v) := d_{\mathcal{P}}(h(u), h(v)),$$

where $h : \mathcal{G} \to \mathcal{P}$ is the mapping between the genotype and phenotype space. As we use reduced OBDDs to represent our individuals, the mapping h is bijective: a reduced OBDD represents exactly one Boolean function and every Boolean function is represented by exactly one reduced OBDD (assuming, that the variable ordering is fixed). Hence, $d_{\mathcal{G}}$ is a metric, too. In the next subsection a rough outline of the used GP-system is given.

6.3.2 A rough outline of the GP-system

In the following, we assume that $MAXGEN \in \mathbb{N}$ (the maximum number of generations), $\mu \in \mathbb{N}$ (the size of a parental generation) and $\lambda \in \mathbb{N}$ (the number of children) are predefined by the user. Then, the used MBGP-system looks as follows:

Algorithm A 1 *The MBGP-system.*

```
1. Set $t := 0$.
2. Initialize $P_0$ by choosing $\mu$ Boolean functions with equal probability from $\{f \mid f : \{0,1\}^n \to \{0,1\}\}$ and including the reduced OBDDs representing these functions.
3. While $t \leq MAXGEN$:
   (a) For $i \in \{1, \ldots, \lambda\}$ do:
       i. Choose two reduced OBDDs $D_1$ and $D_2$ with equal probability from $P_t$.
       ii. Recombine $D_1$ and $D_2$ and call the resulting reduced OBDD $D'$.
       iii. Mutate $D'$ and call the resulting reduced OBDD $D^*$.
       iv. Include $D^*$ in $P'_{t+1}$.
   (b) Choose the $\mu$ OBDDs in $P_{t+1}$ with the highest fitness to form $P_{t+1}$.
   (c) Set $t := t + 1$.
4. Output the OBDD with the highest fitness that was found.
```

The MBGP-system resembles a (μ, λ)-ES with respect to the following points:

- The size of a parent population is μ.
- The number of children per generation is λ.
- The μ best children form the next parent generation.
- The children are generated by choosing two parents with equal probability, recombining them and mutating the result.

In the next two subsections, we will fill this rough outline with content by describing recombination and mutation in detail.

6.3.3 Mutation

To be an MBEA the mutation operator must fulfill the requirements (M 1), (M 2) and (M 3). In the following we will present a concrete implementation fulfilling all three requirements.

The basic idea of the mutation of an OBDD D consists of the following steps:

1. Choose a number $M \in \{0, \dots, 2^n\}$.
2. Choose M inputs, i.e. elements of $\{0,1\}^n$.
3. Change the output of D for the M chosen inputs.

The number M is chosen randomly using the following probability distribution, where $\alpha \in \{0,1\}$ is a predefined constant (non-costant or even adapting mutation rates will hopefully further improve the system and are left for future research):

$$P(M = k) = \begin{cases} \alpha + (1 \Leftrightarrow \alpha)^{2^n+1} & \text{, falls } k = 0 \\ \alpha \cdot (1 \Leftrightarrow \alpha)^k & \text{, falls } k \in \{1, \dots, 2^n\} \end{cases} .$$

This is a slight modification of the geometrical distribution with parameter α (where $P(M = k) = \alpha \cdot (1 \Leftrightarrow \alpha)^k$) in order to guarantee that M is at most 2^n. Using an equally distributed random variable $X \in [0,1]$, the modified geometrical distribution is created by the following algorithm in expected time $O(1/\alpha)$ (another algorithm to do this can be found in [Rud94]):

Algorithm A 2 *Choose the number M of inputs, that are to be modified.*

1. *Set $M := 0$.*
2. *Choose an equally distributed random variable $X \in [0,1]$.*
3. *If $X > \alpha$ and $M \leq 2^n$, set $M := M + 1$ and go to step 2.*
4. *If $M = 2^n + 1$, set $M := 0$.*
5. *Output M.*

After choosing M with algorithm (A 2), we have to choose the M inputs (i.e. elements of $\{0,1\}^n$), where the output of the OBDD has to be changed. For this reason, we identify the input space $\{0,1\}^n$ with $\{0, \dots, 2^n \Leftrightarrow 1\}$ by interpreting a bit string as binary code. In order to fulfill the requirements, we have to guarantee that we choose M different inputs with equal probability from all subsets of $\{0, \dots, 2^n \Leftrightarrow 1\}$ of size M. This is done by using algorithm (A 3):

Algorithm A 3 *Choose one of the subsets of $\{0, \dots, 2^n \Leftrightarrow 1\}$ with M elements.*

```
1. For all $j \in \{1, \ldots, M\}$:
   (a) Choose $i_j \in \{0, \ldots, 2^n \Leftrightarrow j\}$ with equal probability.
   (b) Set $l := 1$.
   (c) While $i_l \leq i_j$ and $l < j$:
       i. Set $i_j := i_j + 1$.
       ii. Set $l := l + 1$.
   (d) While $l < j$:
       i. Set $t := i_l$.
       ii. Set $i_l := i_j$.
       iii. Set $i_j := t$.
       iv. Set $l := l + 1$.
2. Output $\{i_1, \ldots, i_M\}$.
```

The loop in step 1.(c) maps the element i_j to the i_j-th element of $\{1, \ldots, 2^n\} \setminus \{i_1, \ldots, i_{j-1}\}$, if we assume that all previous elements (i.e. $i_1, \ldots, i_{j-1}$) are sorted in increasing order. But this can easily be shown by induction, as in step 1.(c) the correct position l of i_j is found and all following elements are shifted by one position in step 1.(d). Hence, algorithm (A 3) chooses a subset with M elements with equal probability from all subsets of $\{0, \ldots, 2^n \Leftrightarrow 1\}$ with M elements. The runtime of (A 3) is $O(M^2)$, i.e., its expected runtime is $O(1/\alpha^2)$.

Now the OBDD D has to be mutated by changing its outputs for the inputs $i_1, \ldots, i_M$. This can be done by the following algorithm:

Algorithm A 4 *Change output of D for inputs $i_1, \ldots, i_M$:*

```
1. Choose $M$ by applying algorithm (A 2).
2. Choose $\{i_1, \ldots, i_M\}$ by applying algorithm (A 3).
3. Build a reduced OBDD $D^*$, that outputs 1, if and only if the input is an
   element of $\{i_1, \ldots, i_M\}$.
4. Output the result of the EXOR-synthesis of $D$ and $D^*$.
```

The recuced OBDD D^* in step 3 can be build in time $O(n \cdot M^2)$ and has $O(n \cdot M)$ nodes. Hence, the EXOR-synthesis in step 4 can be done in time $O(|D| \cdot n \cdot M \cdot \log(|D| \cdot n \cdot M))$, i.e. its expected runtime is $O(|D| \cdot n \cdot \log(|D| \cdot n/\alpha)/\alpha)$. So the expected runtime of the mutation of a reduced OBDD D is polynomial in the size of D, if α is constant.

Theorem 1 *The mutation operator described in algorithm (A 4) fulfills the requirements (M 1), (M 2) and (M 3).*

Proof: Requirement (M 1) states, that mutating an arbitrary reduced OBDD D can result in any other reduced OBDD D'. This follows from the fact, that in Algorithm (A 2) any number $M \in \{0, \ldots, 2^n\}$ has positive probability of being chosen and that in Algorithm (A 3) any subset of size M can be chosen with positive probability. Hence, every other function can result from mutation, i.e., every other reduced OBDD.

Requirement (M 2) is fulfilled, as Algorithm (A 2) chooses for M with higher probability $k \in \mathbb{N}_0$ than $k + 1$ and (A 3) chooses every subset with M elements with the same probability. Hence, small changes (with respect to $d_{\mathcal{G}}$) are more likely than greater ones.

The last requirement (M 3) states, that two OBDDs D_1 and D_2 have the same probability of resulting from a reduced OBDD D by mutation, if $d_{\mathcal{G}}(D, D_1) = d_{\mathcal{G}}(D, D_2)$. This again follows from the fact that (A 3) chooses the subset with M elements with equal probability from all subsets with M elements. Hence, the mutation operator fulfills all three requirements. $\square$

6.3.4 Recombination

The recombination operator $r' : \mathcal{G}' \times \mathcal{G}' \times \Omega_{r'} \rightarrow \mathcal{G}$ creates a new reduced OBDD out of two parental OBDDs under random influence (represented by $\Omega_{r'}$). To fulfill requirement (R 1) the new OBDD is not allowed to have a greater distance to one of its parents than the distance between the parents. Requirement (R 2) is some kind of symmetry requirement: with the same probability the result of the recombination has distance d from the first parent and the second parent, respectively.

The source of the resulting OBDD, when recombining two reduced OBDDs D_1 and D_2 results from calling the following recursive algorithm, that gets the sources v_1 and v_2 of D_1 and D_2, resp., as arguments (where $v \rightarrow 0$ and $v \rightarrow 1$ is the zero- and one-successor of a node v, resp., and sinks are assumed to have label x_{n+1} and the variable ordering is w.l.o.g. $x_1, \ldots, x_n$):

Algorithm A 5 *Rec*(v_1, v_2)*:*

```
1. If $v_1 = v_2$, return $v_1$.
2. If $v_1$ and $v_2$ are both sinks, return one of them with equal probability.
3. Let $x_i$ be the label of $v_1$ and $x_j$ the label of $v_2$:
    (a) If $i < j$, return node $v'$ with label $x_i$, $v' \rightarrow 0 := Rec(v_1 \rightarrow 0, v_2)$, and
        $v' \rightarrow 1 := Rec(v_1 \rightarrow 1, v_2)$.
    (b) If $i > j$, return node $v'$ with label $x_j$, $v' \rightarrow 0 := Rec(v_1, v_2 \rightarrow 0)$, and
        $v' \rightarrow 1 := Rec(v_1, v_2 \rightarrow 1)$.
    (c) If $i = j$, return node $v'$ with label $x_i$, $v' \rightarrow 0 := Rec(v_1 \rightarrow 0, v_2 \rightarrow 0)$,
        and $v' \rightarrow 1 := Rec(v_1 \rightarrow 1, v_2 \rightarrow 1)$.
```

So the recombination operator works as follows: starting from the sources, the OBDDs are run through in a parallel manner. If one node has a smaller label than the other node, the zero- and one-successor of this node are visited before the successors of the other node. If both nodes have the same label, the successors of both nodes are visited. If sinks are reached in both OBDDs, one of the sinks is returned with equal probability.

The result of a call of *Rec*(v_1, v_2) for a pair (v_1, v_2) of nodes is not stored to guarantee that the random decisions are independent of each other. Because of this strategy, the running time of the recombination cannot be upper bounded polynomially in the OBDD sizes. Indeed, one can easily think of worst case examples, where the recombination needs exponential running time in n even though the OBDDs have polynomial size in n (for instance, when the reduced OBDD for the parity function is recombined with its complement). But all experiments indicate that the average running time is rather small, which is caused by the increasing similarity of the OBDDs in the population.

Although the recombination operator is similar to the synthesis-algorithm with the Boolean operator $\otimes$, where $\otimes(x_1, x_2)$ returns one of its arguments with equal probability, the result can be very different from the $\otimes$-synthesis of two OBDDs. For instance, the $\otimes$-synthesis of the one-sink and the zero-sink can result in every function, while the recombination results in one of the sinks with equal probability.

To prove that the requirements (R 1) and (R 2) are fulfilled we need more knowledge about the behaviour of the recombination operator: how does the probability distribution on the space of all reduced OBDDs look like, that represents the possible results of the recombination of two given OBDDs?

To answer this question we present the function represented by an OBDD in a different way. For a given OBDD an input defines the set of nodes, that are visited while evaluating the OBDD for this input. The nodes in this set form a *path* from the source to one of the sinks. As every variable is tested on this path at most once, we can identify a path with a vector $a \in \{0, 1, *\}$ with the following interpretation:

- If $a_i = *$, then the variable x_i is not tested on the path.
- If $a_i = 0$, then the variable x_i is tested and the zero-successor was chosen.
- If $a_1 = 1$, then the variable x_i is tested and the one-successor was chosen.

It can be easily seen, that for a given OBDD there is one-to-one equivalence between a path and such a vector. Hence, we will call such a vector a path, too.

Let $B(a)$ (which is called a *block*) the set of inputs, so that the evaluation procedure for these inputs visits the nodes on the path a. It can be written as

$$B(a) = \{x \in \{0, 1\} \mid \forall i \in \{1, \ldots, n\} : a_i \neq * \Rightarrow x_i = a_i\}.$$

It is clear that all inputs belonging to one block have the same output according to the given OBDD. So if $s_D(a) \in \{0, 1\}$ is the label of the sink, that is reached when following the path a in the OBDD D, all elements of $B(a)$ are mapped by D on $s_D(a)$. So D can be represented as the set of all blocks with the according sink:

$$M(D) := \{(B(a), s_D(a)) \mid \text{a is a path in } D \text{ leading to the } s_D(a)\text{-sink}\}.$$

Using this representation of an OBDD, we can explain the effect of the recombination of two OBDDs D_1 and D_2 more easily: remember, that $Rek(v_1, v_2)$ returns a sink, if and only if v_1 and v_2 are sinks. Let a^1 and a^2 be the paths on which v_1 and v_2 were reached in D_1 and D_2, resp. Then a^1 and a^2 have to be compatible, i.e.

$$\forall i \in \{1, \ldots, n\} : (a_i^1 \neq * \vee a_i^2 \neq *) \Rightarrow a_i^1 = a_i^2.$$

The sink that is returned by the recombination operator lies at the end of the *union* $a \in \{0, 1, *\}^n$ of a^1 and a^2, defined by:

$$\forall i \in \{1, \ldots, n\} : a_i := \begin{cases} a_i^1 & \text{, if } a_i^1 \neq * \\ a_i^2 & \text{, if } a_i^2 \neq * \end{cases}.$$

The block $B(a)$ is the intersection of $B(a^1)$ and $B(a^2)$. The sink is chosen with equal probability, so the child-OBDD D' has the following representation:

$$M(D') = \{(B^1 \cap B^2, \otimes(s^1, s^2)) \mid (B^1, s^1) \in M(D^1), (B^2, s^2) \in M(D^2)\}.$$

This form of representation makes it clear that if both parents D_1 and D_2 have the same output for one input, the child-OBDD D has this output for the input, too. If the parents disagree, the output of the child is 0 or 1 with probability 1/2 each. But it is important that these random decisions are not independent for every bit, but dependent on the OBDD-structure. Using this characterisation, we can prove that the requirements (R 1) and (R 2) are fulfilled:

Theorem 2 *The recombination operator described in algorithm (A 5) fulfills the requirements (R 1) and (R 2).*

Proof: For (R 1) we have to show that the distance between D' and D_1 and between D' and D_2 is not greater than the distance between D_1 and D_2. If d is $d_{\mathcal{G}}(D_1, D_2)$, then the OBDDs disagree in their output for exactly d inputs. As the child-OBDD can disagree with one of the parents only, if the parents disagree, it follows, that both $d_{\mathcal{G}}(D', D_1)$ and $d_{\mathcal{G}}(D', D_2)$ can be at most d.

Furthermore, as D' disagrees with D_1 on those d inputs, where D' agrees with D_2, it follows:

$$d_{\mathcal{G}}(D_1, D') + d_{\mathcal{G}}(D', D_2) = d_{\mathcal{G}}(D_1, D_2).$$

Requirement (R 2) states, that D' has distance d with the same probability from D_1 as from D_2. Look at the series of random decisions (each with probability 1/2), that leads to the child D' with distance d from D_1 (on these d inputs D' agrees with D_2). The complementary series of random decisions during recombination leads to the OBDD that is complementary to the old child-OBDD D' on all inputs where D_1 and D_2 disagree. It has distance d from D_2, as it disagrees with D_2 exactly on those inputs, where the old child-OBDD D' agrees with D_2 and D_1 and D_2 disagree. As both series have the same probability, and this argumentation is valid for all series, requirement (R 2) is fulfilled. □

Hence, it follows:

Corollary 1 *The GP-system described by the algorithms (A 1), (A 2), (A 3), (A 4), and (A 5) is a MBEA.*

7 Experimental results

In the following we will present some experiments to show the usefulness of our requirements. We will see, that our MBGP-system needs much less individuals than other comparable GP-systems while solving the benchmark-problems being attacked. Hence, this shows that our requirements make sense at least for the problem class being attacked.

As our MBGP-system tries to find an OBDD-representation for an unknown Boolean function given by a complete training set, we compare our system to other systems trying to find representations of an unknown Boolean function, when the complete training set is given. We use the *computational effort* (see [Koz92] for a definition) to compare our results to those presented in [Koz92, Koz94, Che97]. In the next subsections we shortly summarize the results of [Koz92, Koz94] and [Che97] and present our results. Then we compare them with respect to the computational effort.

7.1 The GP-system introduced by Koza

In [Koz92] a detailed introduction to GP is given and his choice of representation (S-expressions) and genetic operators (mainly recombination by exchange of sub-S-expressions) has become the standard choice for many GP-systems. In [Koz94] the usefulness of automatically defined functions (ADFs), an extension of the representation with S-expressions, is shown by many experiments and comparisons of the results with and without ADFs. Because of these advantages, ADFs have become standard in many GP-systems, too.

One of the problems being attacked both in [Koz92] and [Koz94] was that of finding a representation of an unknown Boolean function given by its complete training set. Both systems (with and without ADFs) do not make any formal requirements on the genetic operators, so in general it is difficult to predict the functional behaviour of the child when that of the parents is known.

GP without ADFs				
Function	Runs	Successful runs	Pop. size	Max. generation
6-Multiplexer	15	15	4000	50
3-Parity	34	34	16000	50
4-Parity	18	18	16000	50
5-Parity	25	11	16000	50
6-Parity	19	0	16000	50

GP with ADFs				
Function	Runs	Successful runs	Pop. size	Max. generation
3-Parity	33	33	16000	50
4-Parity	18	18	16000	50
5-Parity	19	19	16000	50
6-Parity	21	19	4000	50
7-Parity	29	10	4000	50
8-Parity	4	≥ 1	4000	50
9-Parity	4	≥ 1	4000	50
10-Parity	4	≥ 1	4000	50
11-Parity	4	≥ 1	4000	50

Table 2: Results of the experiments found in [Koz92, Koz94]

In Table 2 the number of runs is compared to the number of successful runs for the GP-systems described in [Koz92, Koz94]. A run is successful, when a S-expression was found that had the correct output for all inputs, before the maximal number of generations was reached. While the maximum number of generations was 50 for all runs, the population size varied, so it is shown in the table, too.

For GP with ADFs the n-Parity experiments for $n \geq 8$ were only repeated four times because their running times were too long. Koza states that for every function at least one of the runs was successful, so the exact number of successful runs is not known to us. But four different runs are not enough to allow any reliable guess of the computational effort. Hence, it was omitted for these functions in our comparison.

7.2 An EP-system

The system described in [Che97] uses no recombination but only different mutation operators having effects of different strength on the individual. Hence, this system is an evolutionary programming (EP) system (see [Fog95]) instead of a GP-system. It was applied to (besides a number of other problems) the problem of finding a representation of an unknown Boolean function, given by its complete training set. The Boolean functions used were:

- the 6-multiplexer,
- the 3-, 4-, 5-, and 6-even-parity functions.

The maximal number of generations was 250 for all 50 runs per function, while the population size was 500. Using these parameters, the experiments presented in [Che97] look as shown in Table 3:

An EP-system				
Function	Runs	Successful runs	Pop. size	Max. generation
6-Multiplexer	50	50	500	250
3-Parity	50	50	500	250
4-Parity	50	42	500	250
5-Parity	50	11	500	250
6-Parity	50	0	500	250

Table 3: Results of the experiments found in [Che97]

7.3 The results of our MBGP-system

We tested our MBGP-system on these different functions:

- the 6- and 11-multiplexer,
- the 3- to 12-parity functions, and
- a randomly chosen Boolean function with 8 inputs.

The randomly chosen Boolean function was chosen independently with equal probability for each of the 50 runs from the set of all Boolean functions with 8 inputs. By using this approach, we tested how well the MBGP-system can find representations of functions which have completely no structure. The maximal number of generations for each run was dependent on the number of inputs in order to give the MBGP-system more time to find a solution.

The parameters μ and λ, which determine the size of the parental and successor generation, were chosen as 15 and 100, i.e. from 15 parental OBDDs 100 child OBDDs were generated. Because the initial generation consists of only 15 OBDDs, the number of individuals up to generation i is $15 + 100 \cdot i$.

The parameter α for the mutation strength was chosen for all experiments as 0.2, so that the expected number of mutating bits is approximately four. The results of the MBGP-system are presented in Table 4.

The MBGP-system				
Function	Runs	Successful runs	Pop. size	Max. generation
6-Multiplexer	50	50	(15,100)	30
11-Multiplexer	50	50	(15,100)	1000
3-Parity	50	50	(15,100)	5
4-Parity	50	50	(15,100)	10
5-Parity	50	50	(15,100)	15
6-Parity	50	50	(15,100)	30
7-Parity	50	50	(15,100)	60
8-Parity	50	50	(15,100)	125
9-Parity	50	50	(15,100)	250
10-Parity	50	50	(15,100)	500
11-Parity	50	50	(15,100)	1000
12-Parity	50	50	(15,100)	2000
8-Random	50	50	(15,100)	125

Table 4: Results of the MBGP-experiments

All runs for all functions were successful. But this fact naturally depends on the maximum number of generations the system is allowed to run, so the percentage of successful runs is no fair measure for comparison. A comparison based on the computational effort, a much fairer measure, is given in the next subsection.

Furthermore, we see that even the 8-Random "function" (as the unknown function was chosen randomly for each of the 50 runs, it is no single function, but rather the function class of all Boolean functions with 8 inputs) was found in every of the 50 runs after at most 125 generations. This indicates, that the MBGP-system works well on all Boolean functions, even on those without any "structure".

To get a better overview about the good performance of the MBGP-system we show in Table 5 the average G_{avg} and the maximum G_{max} of the generation, at which the optimum was found for the first time, over 50 runs.

Function	G_{avg}	G_{max}
6-Multiplexer	12.98	22
11-Multiplxer	607.09	776
3-Parity	2.06	3
4-Parity	4.20	7
5-Parity	9.24	11
6-Parity	16.55	19
7-Parity	29.28	51
8-Parity	61.34	79
9-Parity	134.50	201
10-Parity	301.70	486
11-Parity	617.83	792
12-Parity	1391.40	1958
8-Random	56.64	102

Table 5: The average and maximal generation, when the optimum was found, of the MBGP-experiments.

7.4 Comparison of the results

In general one has to be careful when comparing systems that search for representations in different spaces (in our case, in the space of all reduced OBDDs or in the space of all S-expressions over a special terminal and function set), as the search process can have different complexities in different search spaces. But we can compare our MBGP-system to systems using S-expressions, as an OBDD D can be transformed in time $O(|D|)$ to an S-expression over the function set $F = \{AND, OR, NOT\}$, which is common for most systems.

As the unknown Boolean functions for our comparisons we used two function types:

- The n-parity function $par_n : \{0,1\}^n \to \{0,1\}$, whose result is one, if and only if the number of 1-bits in its input is odd.
- The n-multiplexer function $mux_n : \{0,1\}^n \to \{0,1\}$, where n is $2^k + k$ for $k \in \mathbb{N}$, that is defined by

$$mux_n(a_0, \ldots, a_{k-1}, d_0, \ldots, d_{2^k-1}) := d_i \text{ , where } i = \sum_{i=0}^{k-1} a_i \cdot 2^i .$$

While the n-multiplexer problem is not very difficult for most GP-systems (the 20-multiplexer was found by [Yan95] for the first time), the n-parity function is widely assumed to be very difficult for GP-systems working with S-expressions. The 5-parity function is the biggest parity-function that was found by using standard GP-systems with S-expressions but without ADFs (see [GR97]). Using ADFs, representations of the 3-parity up to the 11-parity function were found (see [Koz94]). In this section, we make no difference between the even-parity function, which is one, if and only if the number of 1-bits in its input is even, and the parity function, because one can easily transform a representation of one function to that of the other function for both OBDDs and S-expressions.

In the following we compare our results with respect to computational effort $E(M,z) := \min_i I(M,i,z)$, a guess for the number of individuals that have to be evaluated to reach the optimum with $z\%$ probability, when the population size is M. Here $I(M,i,z)$ is a guess for the number of individuals that have to be evaluated to get the optimum with $z\%$ probability after at most i generations, when the population size is M. To compute $I(M,i,z)$ the following formulas are used (see [Koz92] for a more detailed description):

$$\begin{aligned}
\text{Hits}(i) &:= \text{Number of runs that found the optimum until generation } i \\
P(M,i) &:= \frac{\text{Hits}(i)}{\text{Total number of runs}} \\
R(M,i,z) &:= \left\lceil \left(\frac{\log(1 \Leftrightarrow z)}{\log(1 \Leftrightarrow P(M,i))} \right) \right\rceil \\
I(M,i,z) &:= (i+1) \cdot M \cdot R(M,i,z)
\end{aligned}$$

$R(M,i,z)$ is the number of runs, that are necessary to get the optimum until generation i with $z\%$ probability, when the population size is M. So, the number of individuals to be evaluated is the number of generations $i+1$ (here it is assumed that the initial generation has the same size as all following generations) times the population size M times the number of runs. For the experiments described in [Koz92, Koz94], the probability z was 0.99.

We use the computational effort for comparison, as it reflects the efficiency of the search process in terms of the number of individuals evaluated. Hence, it is independent from implementation details or the hardware being used, but does not allow any direct conclusions on the running time of the algorithm.

Table 6 shows the results. Because in [Koz94] only four runs were done for the 8-, 9-, 10-, and 11-parity function, the computational effort was not computed for these functions. This is abbreviated in the table by the entry "n.e.d." (not enough data).

This comparison shows that the MBGP-system works very well on the functions being tested, using only a very small fraction of the number of individuals the GP-systems described in [Koz92, Koz94] or the EP-system of [Che97] needs. Because these systems rely completely on recombination resp. mutation, we have made further experiments to investigate the role of recombination and mutation in our MBGP-system, which are described in the next subsection.

7.5 The interplay of recombination and mutation

A GP-system basing only on a recombination operator fulfilling the requirements (R 1) and (R 2) is supposed to let the whole population converge to one single individual, as a child can only lie "between" the parents (requirement (R 1)). Hence, over many generations the maximum distance between two individuals of the population will decrease. This makes sense, if we assume that the fitness landscape

Function	GP without ADFs	GP with ADFs	EP	MBGP
6-Multiplexer	160.000	-	93.000	2.215
11-Multiplexer	-	-	-	77.615
3-Parity	96.000	64.000	28.500	315
4-Parity	384.000	176.000	181.500	715
5-Parity	6.528.000	464.000	2.100.000	1.115
6-Parity	-	1.344.000	-	1.915
7-Parity	-	1.440.000	-	5.115
8-Parity	-	n.e.d.	-	7.915
9-Parity	-	n.e.d.	-	20.115
10-Parity	-	n.e.d.	-	48.615
11-Parity	-	n.e.d.	-	79.215
12-Parity	-	-	-	195.815

Table 6: Comparison of the computational effort (with $z = 0.99\%$)

is smooth, indicating that in the neighborhood of individuals with similar fitness other individuals with this fitness lie.

But a GP-system using only a recombination operator fulfilling (R 1) without any mutation can only lead to a best individual that lies "between" the individuals of the initial generation. But in a search space large enough it is very unlikely that the optimum lies "between" the initial population. So a recombination operator, that fulfills our requirements (R 1) and (R 2), alone will lead to stagnation in the search process. Hence, we can assume that mutation is necessary for reliable convergence to the optimum.

To empirically investigate these theories we modified the MBGP-system to two different versions:

- MBGP-system without mutation by eliminating step 3.(a).iii in Algorithm (A 1) and including D' in P'_{t+1} in step 3.(a).iv.
- MBGP-system without recombination by eliminating step 3.(a).ii in Algorithm (A 1) and mutating D_1 to D^* in step 3.(a).iii.

Both versions and the original MBGP-system were run 50-times for the 8-parity function. The average number of hits of the best individual over the 125 generations is shown in Figure 4.

The results confirm our theory, that recombination alone leads to stagnation, although the initial convergence rate is high. While mutation alone is powerful enough to ensure convergence, its convergence is much slower than the initial speed, when using recombination alone or when using mutation and recombination: the average number of generations, until the optimum was found, was 246.88 (its maximal value was 365) when using mutation only. But using mutation and recombination the optimum was found after 61.34 generations on average (and at most 79 generations). So the MBGP-system with mutation and recombination has high convergence speed and also reliable convergence, when trying to find a representation of the 8-parity function.

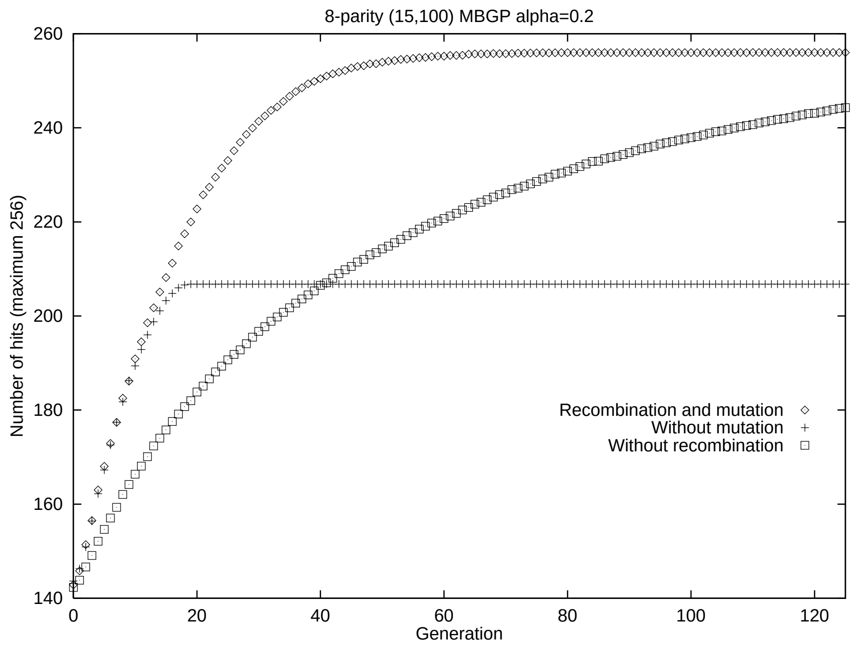

Figure 4: Results for the 8-parity function over 50 runs with (1) mutation and recombination, (2) without mutation, and (3) without recombination.

8 Conclusion

At the beginning of this paper we explained and formalized the problem of representation in evolutionary algorithms. For a proper choice of representation and genetic operators we need knowledge about the problem domain. We assume that this knowledge is expressable by a metric on the genotype space, so similar (according to the metric) elements of the search space have similar fitness. Under these assumptions a number of formal requirements on mutation and recombination are formulated, that define a metric based evolutionary algorithm (MBEA). These requirements shall lead the evolutionary algorithm to a controlled behaviour, where the effects of recombination and mutation are (in some limits) well defined. Assuming that the fitness landscape is smooth, this behaviour should lead to better performance.

To empirically test the performance of an MBEA, we made experiments with an MBGP-system trying to find an unknown Boolean function, given by its complete training set. Emphasizing the role of a proper representation, we used ordered binary decision diagrams (OBDDs) for representation. After introducing and defining OBDDs, we compared existing GP-systems using OBDDs and gave a detailed introduction to our MBGP-system. A number of experiments show that our system needs much less individuals than other GP-systems to find a representation of an unknown Boolean function.

So these experiments show that it can make sense to use formal requirements on the genetic operators, that are in accordance with the knowledge about the problem domain. Furthermore, the choice of a proper representation (with respect to evaluation and genetic operators) is necessary in order to efficiently implement the evolutionary algorithm. An algorithm is only as good as its underlying data structure, i.e. its form of representation.

We are aware that our requirements are only a proposal: other methods of using problem domain knowledge may be more practical and will probably lead to better results. But incorporating knowledge about the problem domain into the algorithm is necessary to have a chance of solving the problem with above-average quality despite the validity of the NFL theorem (see [WM97]). So the MBEA concept is a possibility for formalization of the problem knowledge in evolutionary algorithms.

Of course this concept has to be tested on other problem domains, where it is more difficult to find a representation that allows a metric smoothing the fitness landscape and the construction of genetic operators fulfilling the requirements. Of course, one has to be aware that finding such a metric costs time and does not guarantee successes, so one has to compare the effort to find a better algorithm with the resulting performance gain. In order to circumvent this risk, a theoretical investigation of the properties of a MBEA could lead to more well-based requirements, probably guaranteeing some properties like convergence speed or reliability.

Acknowledgements

Hereby we thank Thomas Bäck, Ulrich Hammel, Thomas Jansen, Günter Rudolph, Hans-Paul Schwefel, and Ingo Wegener for their help while preparing this paper. This research was supported by the Deutsche Forschungsgemeinschaft as part of the collaborative research center "Computational Intelligence" (531).

References

[Ant89] J. Antonisse. A new interpretation of schema notation that overturns the binary encoding constraint. In J.D. Schaffer, editor, *Proc. 3rd Int. Conf. Genetic Algorithms*, pages 86–91. Morgan Kaufmann, 1989.

[Bäc96] Th. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, 1996.

[Bey94] H.-G. Beyer. Towards a theory of "evolution strategies": results for the n-dependent (μ, λ) and the multi-recombinant $(\mu/\mu, \lambda)$-theory. Technical Report of the Systems Analysis Research Group SYS–5/94, University of Dortmund, Department of Computer Science, October 1994.

[Bey95] H.-G. Beyer. How GAs do *NOT* work – understanding GAs without schemata and building blocks. Technical Report of the Systems Analysis Research Group SYS–2/95, University of Dortmund, Department of Computer Science, April 1995.

[Bey97] H.-G. Beyer. An alternative explanation for the manner in which genetic algorithms operate. *BioSystems*, 41:1–15, 1997.

[BFM97] Th. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation.* Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1997.

[BH94] Th. Bäck and U. Hammel. Evolution strategies applied to perturbed objective functions. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano, editors, *Proc. First IEEE Conf. Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 40–45, Orlando FL, June 27–29, 1994. IEEE Press, Piscataway NJ.

[Bry86] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[BS93] Th. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[BS95] Th. Bäck and M. Schütz. Evolution strategies for mixed-integer optimization of optical multilayer systems. In J.R. McDonell, R.G. Reynolds, and D.B. Fogel, editors, *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 33–51, Cambridge, MA, 1995. MIT Press.

[Che97] K. Chellapilla. Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, 1(3):209–216, 1997.

[CS88] R.A. Caruna and J.D. Schaffer. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In J. Laird, editor, *Proceedings of the 5th International Conference on Machine Learning*, pages 153–161, San Mateo, CA, 1988. Morgan Kaufmann Publishers.

[Dav90] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1990.

[Dro97] S. Droste. Efficient genetic programming for finding good generalizing Boolean functions. In J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, H. Iba, and R.L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 82–87, Stanford University, July 13–16 1997. Morgan Kaufmann.

[Dro98] S. Droste. Genetic programming with guaranteed quality. In J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, and R.L. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, University of Wisconsin, Madison, July 22-25 1998. Morgan Kaufmann. (in print).

[FG97] D.B. Fogel and A. Ghozeil. A note on representation and variation operators. *IEEE Transactions on Evolutionary Computation*, 1(2):159–161, 1997.

[Fog95] D.B. Fogel. *Evolutionary Computation:* Toward a New Philosophy of Machine Intelligence. IEEE Press, New York, 1995.

[GHM$^+$96] B. Groß, U. Hammel, P. Maldaner, A. Meyer, P. Roosen, and M. Schütz. Optimization of heat exchanger networks by means of evolution strategies. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV, Int'l Conf. Evolutionary Computation*, pages 1002–1011, Berlin, September 22–26, 1996. Springer, Berlin.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[Got89] W. Gottschalk. *Allgemeine Genetik.* Georg Thieme Verlag, Stuttgart, 1989.

[GR97] C. Gathercole and P. Ross. Tackling the Boolean even n parity problem with genetic programming and limited-error fitness. In J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, H. Iba, and R.L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 119–127, Stanford University, July 13–16 1997. Morgan Kaufmann.

[Hol75] J.H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[JF95] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, 1995.

[Koz92] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[Koz94] J.R. Koza. *Genetic Programming II.* MIT Press, 1994.

[Mic96] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs.* Springer, Berlin, 1996.

[Rad94] N.J. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10(4):339–384, 1994.

[Rad97] N.J. Radcliffe. Schema processing. In Th. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages B2.5:1–10. Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1997.

[Rec73] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, 1973.

[Rec94] I. Rechenberg. *Evolutionsstrategie '94.* Frommann-Holzboog, Stuttgart, 1994.

[RS95] N.J. Radcliffe and P.D. Surry. Fitness variance of formae and performance prediction. In L.D. Whitley and M.D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 51–72, San Francisco, CA, 1995. Morgan Kaufmann.

[Rud94] G. Rudolph. An evolutionary algorithm for integer programming. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature - PPSN III, Int'l Conf. Evolutionary Computation*, pages 139–148, Jerusalem, October 9–14, 1994. Springer, Berlin.

[Rud97] G. Rudolph. *Convergence Properties of Evolutionary Algorithms.* Verlag Dr. Kovač, Hamburg, 1997.

[Sch95] H.-P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology Series. Wiley, New York, 1995.

[SHIK96] H. Sakanashi, T. Higuchi, H. Iba, and Y. Kakazu. An approach for genetic synthesizer of binary decision diagram. In *Proceedings of the International IEEE Conference on Evolutionary Computation*, pages 559–564, Nagoya, Japan, May 20–22 1996.

[Sie98] D. Sieling. On the existence of polynomial time approximation schemes for OBDD minimization (extended abstract). In *Proceedings of 15th Symposium on Theoretical Aspects of Computer Science STACS'98*, pages 205–215. Springer Verlag, 1998.

[SKvS97] B. Sendhoff, M. Kreutz, and W. von Seelen. A condition for the genotype-phenotype mapping: Causality. In Th. Bäck, editor, *Proceedings of The Seventh International Conference on Genetic Algorithms*, Michigan State University, East Lansing, MI, July 1997. Morgan Kaufmann.

[SS96] M. Schütz and J. Sprave. Application of parallel mixed-integer evolution strategies with mutation rate pooling. In L.J. Fogel, P.J. Angeline, and Th. Bäck, editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 345–354, Cambridge, MA, 1996. MIT Press.

[SW93] D. Sieling and I. Wegener. Reduction of BDDs in linear time. *Information Processing Letters*, 48:139–144, 1993.

[WM97] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[Yan95] M. Yanagiya. Efficient genetic programming based on binary decison diagrams. In *Proceedings of the International IEEE Conference on Evolutionary Computation*, pages 234–239, 1995.