Genetic Programming using Genotype-Phenotype Mapping from Linear Genomes into Linear Phenotypes

Robert E. Keller   Wolfgang Banzhaf
University of Dortmund
Department of Computer Science, LS11
D-44221 Dortmund, Germany
{keller,banzhaf}@ls11.informatik.uni-dortmund.de

ABSTRACT

In common genetic programming approaches, the space of genotypes, that is the search space, is identical to the space of phenotypes, that is the solution space. Facts and theories from molecular biology suggest the introduction of non-identical genospaces and phenospaces, and a generic genotype-phenotype mapping which maps unconstrained genotypes into syntactically correct phenotypes. Neutral variants come into effect due to this mapping. They enhance genetic diversity and allow for escaping local optima in phenospace via high-dimensional saddle surfaces in genospace. We propose a concrete mapping that maps linear binary genotypes into linear phenotypes of an arbitrary context-free programming language. Empirical results are presented which show that the mapping improves the performance of GP under mutation and reproduction.

1 Introduction

Common GP approaches (e.g. "Koza (1992)") do not distinguish between a genotype, i.e. a point in search space, and its phenotype, i.e. a point in solution space, that is search space and solution space are identical.

Recently, however, a strict distinction between the search space and the solution space as well as a genotype-phenotype mapping (GPM) from the search space into the solution space have been suggested in "Banzhaf (1994)". In that contribution, the author extends the common GP approach by a GPM and demonstrates the practicability of this *GP/GPM approach.*

Note that a GPM in itself is not a special GP variant. The principle behind a GPM is the distinction between the search space and the solution space of an underlying search process. The process locates points in the search space as results of the search. However, a potential solution must be a point from the solution space. Thus, a GPM is needed which maps an arbitrary search point onto a solution point. This principle can be used with every search process, like an arbitrary evolutionary algorithm.

Finally, it is hypothesized in "Banzhaf (1994)" that the GP/GPM approach enhances the performance of systems using evolutionary algorithms (EA).

The present paper focuses on a basic empirical investigation of this hypothesis with respect to the area of GP. To that end, we compare the performance of both approaches by applying them to the same non-toy test problem. In order to do so, we use our Binary-Genetic-Programming system, which incorporates both approaches.

One starting point towards the GP/GPM approach given in "Banzhaf (1994)" is Kimura's neutral theory of molecular evolution "Kimura (1968)" "Kimura (1983)" which postulates that molecular evolution is essentially driven by mutations almost neutral considering natural selection. This includes different genotypes (*neutral variants*) coding for precisely the same phenotype. The phenomenon of neutral mutations, according to Kimura, is a major reason for the high genetic diversity in natural populations. Note that this theory has been supported by empirical results "Mukai (1985)".

The motivation for applying this theory to the field of evolutionary algorithms is that an EA solving a problem often faces solving a *constrained* optimization problem. Not only should the EA evolve one or more phenotypes with highest possible quality (fitness), it is also forced to evolve only such phenotypes which obey given restrictions.

Common GP approaches, for instance, face a hard constraint given by the syntax of the underlying programming language: all generated individuals must be legal, that is syntactically correct. These approaches handle this constraint like GAs handle hard constraints: all operators modifying individuals, e.g. crossover, are

constrained such that they only generate legal individuals.

Thus, large regions of the search space - which equals the solution space - are infeasible, thereby presenting a twofold problem to common GP approaches. First, the large potential of genetic diversity existing in the infeasible regions is not accessible. Second, on the search for high-quality individuals, sequences of individuals get generated. The larger the infeasible regions are, the higher is the probability they include such individuals which build short individual-sequences leading from the individuals of the actual generation to high-quality individuals of the next generation. To put it vividly: GP tries to wander from good to better individuals, and infeasible regions force it to detour.

This train of thought motivates the extension of a common GP approach by a distinction of search space and solution space, by unconstrained operators performing on the search space only, and by a GPM from search space into solution space. The search space equals the set of all genotypes, legal and illegal. A GPM maps each genotype into a legal phenotype, thereby matching the syntactical constraint.

By this extension, the problem of infeasible regions does not exist any more: the complete search space has become evolvable along short individual-sequences. Under mutation as one of the search operators, this leads to a random genetic drift in the population. In the ideal case, this drift leads to a genotype distribution such that for *each* legal phenotype p there exists a genotype g *in the actual generation* such that g is very close to some genotype h of p "Eigen (1992)". "Close" means there is a short individual-sequence leading from g to h. This enlarges the probability of finding p.

Especially, this enables evolution to "escape local optima on saddle surfaces", as it is put in "Banzhaf (1994)". If a population gets trapped in a suboptimum, there are many neutral variants mapping into the trapped phenotypes. Some of these variants will be close to neutral variants of phenotypes representing better (sub)optima. Thus, there is a high probability that evolution will find these phenotypes. On the other hand, there is a high probability that evolution will find worse phenotypes. Due to the fitness-based performance of reproduction and search operators, however, the corresponding worse genotypes will have few chances to proliferate.

In order to keep the comparison of both approaches transparent, certain restrictions will be imposed on the evolutionary process. Since mutation is the major search operator in Kimura's theory, we will use mutation as only search operator. Furthermore, the genotypes of the GP/GPM approach will all feature the same fixed length.

Note that it is not intended to solve the problem. The restrictions just mentioned will probably prevent the finding of a perfect solution anyway.

2 Representations, GPM, operators

Concerning the GP/GPM approach, some thought must be given to the representation of genotypes. In "Banzhaf (1994)", each genotype is identified with a binary string. This provides for the simplest and most universal representation of information, and it allows the use of simple search operators, which could even be standard GA operators. Additionally, it enables a sort of mapping such that the generation of neutral variants is enhanced, as will be shown subsequently. Finally, it provides exactly one type of search space, a space of binary strings, which is independent from the representation of phenotypes in the actual solution space.

As GPM to be actually used, an instantiation of a generic GPM "Banzhaf (1994)" motivated by molecular biology "Watson et al.(1987)", is taken. The generic GPM is a metaphor of the biological process of *protein synthesis*. During this process, DNA (genotype) gets transcribed into messenger RNA (mRNA), which is corrected with respect to the functionality of the protein it is coding for. The corrected mRNA then gets translated into amino acid sequences, which fold into a protein (phenotype) that features a certain functionality (behavior) with respect to the cell metabolism.

Protein synthesis is a very complex process. The GPM to be actually used is a simplified metaphor of this process. A binary string (genotype) gets transcribed into a *raw sequence* of symbols (*transcription*). Each such symbol is an element of either the function or the terminal set which both underlie a genetic-programming approach.

The raw sequence gets corrected, if necessary, according to the syntax of the used programming language, thus yielding a *legal symbol sequence* (*repairing*).

Then, *editing* turns this sequence into an *edited symbol sequence* by adding standard information, e.g. a main program frame enclosing the repaired sequence.

Finally, the last phase of the mapping, which is *compilation* of the edited symbol sequence, transforms this sequence into a machine-language program that can be executed in order to evaluate the fitness of the corresponding individual. Alternatively, *interpretation* of the edited symbol sequence can be used for fitness evaluation.

There remains the question of the representation of phenotypes in the GP/GPM approach. A legal symbol sequence s is the abstract expression of the behavior defined by that genotype that got mapped onto s. Since editing and compilation or interpretation do not add behavior-changing information to such a sequence, it is logical to consider this sequence as a phenotype. Thus, in the context of the GP/GPM approach, the solution space can be defined as the set of all legal symbol sequences.

Note that, with respect to the employed GPM/protein synthesis metaphor, a legal symbol sequence corresponds to a protein. Since amino acids are the essential components of a protein, an amino acid corresponds to a symbol.

We now detail the processes of transcription and repairing used during the GP/GPM approach. To that end, we introduce the notion of a *codon*. In molecular biology, this is a triplet of nucleic acids which uniquely encodes one amino acid, at most. Since an amino acid corresponds to a symbol, a codon corresponds to the encoding of a symbol.

This puts forward the question for the nature of the encoding to be used. At the beginning of this section, an argumentation in favor of a binary representation of a genotype has been given. This argumentation is also valid with respect to a codon. Thus, for the GP/GPM approach, a *codon* shall term a bit sequence of b bits length which encodes a symbol. In order to provide for the unique encoding of all symbols, b must be chosen such that for each symbol there is at least one codon which encodes this and only this symbol. For instance, if there are 20 symbols, $b \geq 5$ is a proper value, since 2^b is greater than 20.

The idea of codons implies that a genotype, which is a binary string by definition, consists of a sequence of n codons. Let the codons of a genotype be identified by positional numbers, starting with 0 and ending with $n-1$.

An arbitrary mapping M from the set of codons into the set of symbols can now be defined. Thus, M defines the encoding of symbols by codons. Transcription scans a genotype, starting at codon 0, ending at codon $n-1$. Using M, transcription maps each encountered codon onto the corresponding symbol, thereby creating a raw symbol sequence from the genotype.

In molecular biology, the analogy to our mapping M is called *genetic code*. The mapping M shall be termed likewise.

The natural genetic code is the mapping from codons into amino acids. It is important to note that this code is highly redundant: there are many different codons which encode the same amino acid. We will return to this point later with respect to the design of an artificial genetic code represented by some mapping M.

For later use, each codon of a certain artificial genetic code shall be identified with an index. That could be, for instance, the integer number represented by the codon. For instance, the codon 0110 could be identified with 6.

In order to define repairing, we only consider context-free LALR(1) (look-ahead-left-recursive, look ahead one symbol) grammars "Aho et al. (1986)". These grammars have a nice property with respect to compiler construction. An essential phase of the compiling of a program, that is a symbol sequence, is the parsing of this sequence in order to verify the syntactical correctness of the program. When parsing a symbol sequence according to a LALR(1) grammar, it is always and efficiently possible to compute the current *legal symbol set* with respect to the actually scanned symbol s. Each symbol of this set is syntactically correct in place of s. Especially, this set can be computed by solely considering the symbol preceding s.

Note that, since we only consider LALR(1) grammars for the GP/GPM approach, a GP system using the GP/GPM approach as described here can only evolve programs in languages that are defined by such a grammar. This, however, is no restriction with respect to the practicability of the approach, since many practically relevant languages like C are LALR(1) languages.

A symbol that represents a syntax error shall be called *illegal*. When parsing a raw sequence, it is highly probable that an illegal symbol s will be scanned. The legal symbol set can be computed with respect to s. In such a situation, a *minimal-distance set* can be defined. It is a subset of the actual legal symbol set, and it contains all those symbols whose codons are Hamming-closest to the codon of the illegal symbol s. With respect to the illegal symbol, such a symbol is called *closest*. Depending on the genetic code, there can be more than one such symbol.

We can now define the repairing of a raw sequence. The sequence gets parsed, and, in case an illegal symbol is encountered, it will be replaced by its closest symbol. If there are more than one closest symbol, that one will be taken which is encoded by a codon with the lowest index among all codons of closest symbols.

A common special case must be handled. Often, the described type of repairing will produce a symbol sequence that terminates unfinished. For instance, this sequence could be an unfinished arithmetical expression like $sin(a)*cos(b)+$. In order to handle that case, we assign a *termination number* to each symbol. This number indicates how appropriate the symbol is for shortest possible termination of an unfinished sequence. In the above expression, for instance, a variable would terminate the sequence, while an operator symbol like sin would call for at least three more symbols ('(', variable, ')'). By definition, the termination number of a symbol shall equal the number of these additionally needed symbols. Thus, in the example, a variable-symbol like 'a' has termination number 0, while a symbol like sin has termination number 3.

Note that the number of symbols *actually* needed for termination when using a certain symbol can be larger than its termination number. For instance, there could be open parantheses prior to the end of the unfinished sequence, which have to be closed. Such context-sensitive circumstances shall not be reflected in a termination number.

In case there are several symbols with equal minimal

termination numbers, that one shall be taken which is encoded by a codon with the lowest index among all codons of the symbols in question.

We give a simple example for the functioning of the described GPM. The underlying syntax shall be that of simple arithmetic expressions. Consider the redundant genetic code 000 $\rightarrow a$; 001 $\rightarrow b$; 010 $\rightarrow +$; 011 $\rightarrow$ $*$; 100 $\rightarrow a$; 101 $\rightarrow b$; 110 $\rightarrow +$; 111 $\rightarrow *$.

The genotype 000 001 011 gets transcribed into the raw sequence $ab*$. Repairing scans a as first and legal symbol. It then scans b as an illegal symbol with 001 as its codon in the genotype. $\{+,*\}$ is the legal symbol set. The symbol closest to b is $*$. Thus, b gets replaced by $*$, thereby terminating the repair of the syntax error. The partially repaired raw sequence now equals $a**$. Repairing continues, replacing the last symbol $*$ with b. The repaired sequence $a*b$ is passed on to editing, which adds, for instance, a function frame. The edited sequence might look like

```
double fnc(double a, double b) {return a*b;}.
```

This sequence could now, for example, be integrated, together with other edited sequences, into a main program which could be given to any ANSI-C compiler.

Some more examples illustrate how the existence of many neutral variants for certain phenotypes can be explained out of the presented GPM. For instance, as shown above, 000 001 011 gets mapped into the phenotype $a*b$. However, 010 011 101, 000 011 001 and 100 111 101 get all mapped into $a*b$ as well. When analyzing this phenomenon, it becomes immediately clear that both the redundancy of the genetic code and the repair mechanism are responsible for this effect.

Note that the presented mapping mechanism indeed defines a *mapping* from genospace into phenospace: each genotype gets mapped always into exactly one phenotype. Otherwise, a genotype very likely would get associated with different phenotypes and thus with different fitness values over time. This would not allow for a proper fitness-related selection on genospace.

We now present the creation and mutation operators separately for the GP/GPM approach and the common GP approach. These operators perform selections on sets of codons, of symbols and of bits. All of these selections shall operate randomly and equally distributed.

For the GP/GPM approach, creation generates a random individual as random binary string which consists of n codons.

For the common GP approach, creation builds up a syntactically legal random symbol sequence which consists of at least n symbols. This is done by starting with an empty symbol string. The first symbol is selected from the set of symbols which are legal first symbols. The following symbol is selected from the set of symbols which are legal as follow-up symbols to the first symbol. This method iterates until the n-th symbol has been selected. The result is a legal sequence of n symbols. However, the sequence may be unfinished. In that case, it is terminated in the same way that has been described for the case of repairing. Note that this leads to individuals with varying length.

For the GP/GPM approach, two mutation operators shall be devised. The *coupled mutation*, applied to a genotype, selects a codon. It then flips two randomly selected different bits in that codon. This operator reflects the fact that some mutations tend to change nucleic acids in a coupled sort of way.

The *unrestricted mutation* selects a codon. In that codon, it then flips one randomly selected bit with 0.5 probability, or two randomly selected different bits with 0.35 probability, or three randomly selected different bits with 0.1 probability, or four randomly selected different bits with 0.05 probability. The probability distribution reflects the natural *principle of variation*: small changes occur more often than big changes.

For the common GP approach, mutation selects a symbol in the sequence. It then replaces this symbol by another randomly selected symbol which is legal in the respective position.

Note that this mutation operator does not produce vast changes in the genotype by replacing complete syntactical units as it is the case, for instance, with the mutation operator presented in "Koza (1992)". We feel that the presented operator shows a behavior closer to natural mutation and thus better matches the spirit of artificial evolutionary processes.

Note that an individual of the common GP approach is encoded as a symbol sequence, that is a string. It is not represented in a tree-like form like it is often done in work related to genetic programming. However, both the string encoding and the tree encoding are equivalent in the sense that each arithmetic expression represented as a string can be encoded as expression tree and vice versa.

Moreover, given corresponding mutation operators, each mutation done on an expression tree can be performed equivalently on the corresponding string and vice versa. It is interesting to note that this is also true with respect to crossover, although crossover is not used in the current context.

We finally sketch our binary genetic-programming system used for the empirical comparison of both approaches. The system incorporates the standard GP algorithm, as it is presented in "Koza (1992)", as kernel. Tournament selection on two individuals is used in order to select an individual for subsequent reproduction only or, alternatively, for reproduction followed by mutation. Both alternatives have 0.5 probability of getting applied as next operation on a selected individual.

Adjusted fitness is used as fitness measure. The squa-

re-error sum produced by an individual i over all fitness cases shall define i's standardized-fitness value, which is needed for the computation of i's adjusted-fitness value "Koza (1992)". Thus, all possible fitness values exist in $[0, 1]$. A perfect individual has fitness value 1.

In case the system uses the GP/GPM approach, the kernel calls the GPM between creation and fitness evaluation of the initial generation, and between the completed evolution of the actual generation and its fitness evaluation, respectively. Dependent on the actually used approach, the corresponding representations and operators are employed.

3 Method

The programming language to be used for the evolution shall be ANSI-C. Thus, a phenotype will be a symbol sequence obeying ANSI-C syntax. The domain of the test problem will suggest variables, unary and binary arithmetic functions, and paranthesis operators as elements of the terminal and function sets.

The test problem is a symbolic function regression on a four-dimensional parameter space. The function in question is

$$\sin(m) \cdot \cos(v) \cdot \frac{1}{\sqrt{e^q}} + \tan(a).$$

All parameter values shall be real-valued.

In order to allow for protection against division-by-zero, we devise the division function `D(x)` which returns the reciprocal value of its single argument. If the argument is zero, the result equals 1. We supply a protected square root function `sqrt(x)` which returns the square root of the absolute value of its argument. Furthermore, an overflow-protected exponential function `exp(x)` is provided which returns e^x. In case the value of x causes an overflow, the returned value is 1.

As codons, we supply the numbers 0 up to 15 in their binary representation, i.e. $0000, 0001, .., 1110, 1111$. In this order, the codons get mapped into a set of 16 symbols, featuring 14 different symbols:

$$+ \ * \ * \ D \ m \ v \ q \ a \ (\ ) \ \mathit{sin} \ \mathit{cos} \ \mathit{tan} \ \mathit{sqrt} \ \mathit{exp} \ ).$$

Note there is only little redundancy in this genetic code, so that in case the theory of neutral variants works at all, the GP/GPM approach will not have vastly more power than the common GP approach.

For the GP/GPM approach, the genotype length shall be 25, that is each genotype consists of 25 codons. Since there are 16 different codons in the genetic code, the search space contains 16^{25} or approximately 1.3E30 genotypes.

When using the unrestricted mutation operator which can mutate each single bit independently the GP/GPM approach faces $25 \cdot 4 = 100$ degrees of freedom, since each codon consists of 4 bit. In other words, in addition to the relatively large size of the search space, it is high-dimensional, featuring 100 dimensions.

For the common GP approach, the individual's length shall be at least 25, too. Thus, each individual is a symbol sequence of at least 25 symbols. Due to the above described phenomenon of unfinished sequences, the actual length of individuals after repairing can surpass 25. Imagine, for instance, the improbable case of an individual looking like

```
D(D(D(D(D(D(D(D(D(D(D(D(D  (a))))))))))))).
```

Prior to the spaces within the individual, there are 25 symbols. Shortest possible termination of the individual still requires the appending of 15 more symbols, most of them being ')', in order to close still open paranthesis levels.

However, the search space in the common GP approach has a size of much less than 14^{25} or approximately 4.5E28 individuals. That is because the symbols appended have not been evolved but non-stochastically computed, and because many symbol sequences are illegal. By using the mutation operator, which can mutate each of the 25 evolved symbols in a sequence independently, the common GP approach faces 25 degrees of freedom. Thus, the search space has only 25 dimensions.

Note that, for the GP/GPM approach, the maximal length of a phenotype can surpass 25, too, for the same reason that has been given above.

Due to the real-valued four-dimensional parameter space, a fitness case consists of four real input values and one real output value. We only supply 10 fitness cases in order to further increase the difficulty of the problem by lack of information. All experimental runs use a population size of 500 individuals and run for 50 generations.

In order to compare both approaches with respect to their performance, three series of experiments are run using the binary-genetic-programming system. Each series consists of 19 runs. All runs of the same series shall be indexed from 1 up to 19. All runs of the same series run with different randomizer seeds. Runs with identical indices, but from different series, run with the same randomizer seed.

A run from series 1 is done with the common GP approach. A run from series 2 features the GP/GPM approach, using the coupled mutation operator. A run from series 3 is done with the GP/GPM approach, using the unrestricted mutation operator. In all other respects, the runs perform under identical conditions, which have been described above.

4 Results and discussion

Fig. 1 shows the mean average fitness over time for all 3 series of runs. The graphs titled "GPM-coupled" and "GPM-unrestricted" belong to the series featuring

the GP/GPM approaches with coupled and unrestricted mutation. The graph titled "Common" belongs to the series featuring the common GP approach.

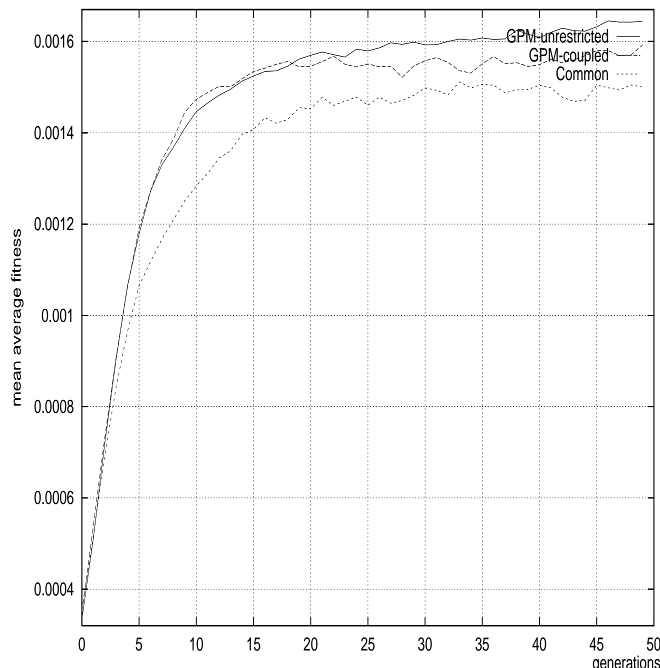

Figure 1 Mean average fitness of unrestricted, coupled and common GP approach

Both the coupled and the common GP approach start at generation 0 with practically identical fitness values, indicating that there is no bias in the GPM-independent initial conditions which might favor the GP/GPM approach.

Both graphs rise steeply until generation 5, with the GPM-coupled graph showing a clearly higher increase rate. At generation 5, the Common graph suffers a significant decrease in its inclination, while the GPM-coupled graph still rises strongly with only a light decrease in its inclination. This indicates the effectiveness of neutral variants helping to escape from local optima.

It is not until generation 9 that the inclination of the GPM-coupled graph decreases significantly. The inclination of both graphs dwindles down to more or less 0 over the next generations.

In the final generation, the coupled approach features a 6 percent higher fitness relative to the common GP approach. After generation 0, the fitness values of the common GP approach never reached or surpassed those of the coupled approach.

Both the coupled and the common GP approach get significantly outperformed by the unrestricted approach. As can be seen from the graphs, this approach behaves very similar to the coupled approach, at first. Especially, it starts in generation 0 with approximately the same fitness value as the other approaches. From generation 18 onward, however, the unrestricted approach clearly leaves the other approaches behind. In the final generation, the unrestricted approach features a 10 percent higher fitness relative to the common GP approach. After generation 1, the fitness values of the common GP approach never reached or surpassed those of the unrestricted approach.

We offer a hypothesis for the superiority of the unrestricted approach over the coupled approach. Given some codon, coupled mutation cannot mutate this codon into each other arbitrary codon, since it *always* flips 2 bits. For instance, no sequence of coupled mutations can change 0011 into 0001.

This implies that, given some genotype g, the search process cannot reach all genotypes. Thus, if the search process gets stuck in a local optimum, it can only reach certain other genotypes from there. If none of these genotypes has a better fitness than those representing the local optimum, the process cannot escape. Unrestricted mutation, however, can change each codon into each other codon, since it can flip one bit only. Thus, the search process can potentially escape.

In the final generation, the mean *best* fitness of the unrestricted and coupled approaches equal approx. 0.97 and 0.96 compared to approx. 0.93 for the common GP approach. The mean best-fitness values of the common GP approach never reached or surpassed those of the GP/GPM approaches.

Note also that the mean best-fitness values of all three approaches get rather close to the perfect fitness value 1.0, although no crossover operator has been used. This indicates that mutation should deserve more attention in GP instead of merely being considered a secondary operator.

Note that the superior performance in mean and best average fitness of both GP/GPM approaches compared to the common GP approach occur though search space size and dimensionality are significantly higher compared to the common GP approach, and despite the fact the used genetic code features not very much redundancy, which restricts the chance for the generation of neutral variants. This stresses the advantage of the GP/GPM approach.

Note that the genetic code used for the runs is redundant with respect to the multiplication and the closed-paranthesis operator. This might have positive and negative effects on the performance.

For instance, the redundancy in the multiplication operator could result in more such operators as parts of a phenotype. This can be advantageous since the problem function features two such operators.

On the other hand, the redundancy in the closed-paranthesis operator could pose a handicap to evolution, since it enlarges the probability that a needed long subexpression never emerges.

5 Conclusion

A first comparison of the common GP approach vs. the GP/GPM approach has been the main focus of this contribution. The presented empirical results argue in favor of the use of GPM, at least in GP systems where mutation is employed as a search operator. Furthermore, mutation appears to be a useful operator when applied by both the common or the GP/GPM approach.

6 Further research

Clearly, the genetic code is a key issue in GPM. It is a non-trivial task to determine the "optimal" code dependent on the actual problem. To put it more formally, the challenge is to define a GPM approach such that the resulting search space topology allows for an efficient evolution with respect to the underlying fitness landscape. Real-world problems, however, usually present "terra-incognita" landscapes.

Taking a closer look at *Evolutionsstrategien* "Schwefel (1995)" and the important subject of self-adaptation of parameters that control the search process suggests that an individual genetic code as part of a genotype might be beneficial for the search process as a whole. The individual codes could evolve along with the other features of the genotypes. This would be especially valuable in cases when a GP system gets confronted with dynamic fitness landscapes.

Furthermore, the results and their discussion suggest the use of a truly unrestricted mutation operator which flips between 1 to n bits under normal distribution, n being the number of all bits in the genotype. This operator should lead to even better performance of a GP/GPM approach than the presented unrestricted mutation operator.

The role of crossover in a GP/GPM approach and the relationship between crossover and mutation in that context call for intense research.

Obviously, the phase of repairing an illegal raw sequence is essential to a GP/GPM approach. We have presented a repairing type that replaces an illegal symbol by a legal one.

However, repairing could also be done by inserting or deleting symbols. For instance, the illegal raw sequence `a++b` can be turned into, for example, `a+a+b` by inserting `a` between the addition operators. On the other hand, the sequence can be transformed into `a+b` by deleting an addition operator.

Further research must go into potential connections between convergence properties of a GP/GPM approach and the type of employed repairing.

Finally, the GP/GPM approach and the common GP approach must be compared on a large set of further hard and diverse problems, using different genetic codes.

Bibliography

A.V. Aho et. al.: Compilers. Addison-Wesley. 1986.

W. Banzhaf: Genotype-Phenotype-Mapping and Neutral Variation – A case study in Genetic Programming. In: Y. Davidor et al. (eds.): Parallel Problem Solving from Nature III. Berlin: Springer. 1994.

M. Eigen: Steps toward Life: a perspective on evolution. Oxford: Oxford University Press. 1992.

M. Kimura: Evolutionary rate at the molecular level. Nature 217, 624-626. 1968.

M. Kimura: The Neutral Theory of Molecular Evolution. Cambridge: Cambridge University Press. 1983.

J.R. Koza: Genetic Programming. Cambridge (USA): MIT Press. 1992

T. Mukai: Experimental Verification of the Neutral Theory. In: T. Ohta et. al. (eds.): Population Genetics and Molecular Evolution. Berlin: Springer. 1985.

H.-P. Schwefel: Evolution And Optimum Seeking. New York: Wiley. 1995.

J.D. Watson et al.: Molecular Biology of the Gene. Amsterdam: Benjamin / Cummings Publishing Company. 1987.