Explicit Control of Diversity and Effective Variation Distance in Linear Genetic Programming

Markus Brameier Wolfgang Banzhaf

Department of Computer Science
University of Dortmund
44221 Dortmund
Germany
email: brameier,banzhaf@LS11.informatik.uni-dortmund.de

Abstract

We investigate structural and semantic distance metrics for linear genetic programs. Causal connections between changes of the genotype and fitness changes form a necessary condition for analyzing structural differences between genetic programs and for the two major objectives of this paper: (i) Distance information between individuals is used to control structural diversity of population individuals actively by a two-level tournament selection. (ii) Variation distance of effective code is controlled for different genetic operators—including an effective variant of the mutation operator that works closely with the used distance metric. Numerous experiments have been performed for a regression problem, a classification task, and a Boolean problem.

1 Introduction

A *fitness landscape* on the search space of programs is defined by a distance metric for program structures and a fitness function that reflects the quality of program semantics. The application of a genetic operator corresponds to performing one *step* on the landscape. Both the roughness of the fitness landscape and the step size of the operator decide on the success of the search process to find solutions that are optimal or close to the optimum. On the one hand, a genetic operator has to allow progress in steps that are small enough. On the other hand, the steps must be related through a distance measure on the fitness landscape that is smooth at least in local regions (local strong causality). Otherwise, evolutionary search may not be more powerful than random search. In a totally rugged fitness landscape the (structural) neighbors of a search point (program) do not promise a moderate change in fitness, i.e., program semantics. However, the surface of the fitness landscape depends not only on the problem but on the configuration of the GP system, too, especially on the provided set of function operators.

In contrast to other evolutionary search algorithms like evolution strategies (ES), genetic programming (GP) may fulfill the principle of *strong causality*, i.e., small variations in genotype space imply small variations in phenotype space [13], only weakly [15]. Obviously, changing just a small program component may lead to almost arbitrary changes in program behavior. However, it can be expected that, on average, smaller variations of the genotype lead to smaller variations in fitness.

The edit distance, sometimes referred to as *Levenshtein distance*, [6] between varying length character strings has been proposed as a metric for representations in genetic programming [10, 14]. Such a metric not only permits to analyze genotype diversity within the population but offers a possibility to investigate the effect (*step size*) of variation operators. In [8] correlation between edit distance and fitness change of tree programs has been demonstrated for different variation operators and test problems.

This work introduces efficient structural distance metrics that operate selectively on *substructures* of the program representation in linear GP (LGP). Correlation between structural and semantic distance as well as distribution of distances are documented for two different types of variation. One uses recombination while the other one is based on (macro) mutations only. In the course of these basic experiments the development of structural and semantic *diversity*, i.e., the average distance between two (randomly selected) individuals, is analyzed over a run.

One major objective of this contribution is to apply distance metrics for an *explicit control* of diversity within LGP populations. Therefore, we introduce a two-level tournament selection that selects for fitness on the first level and for diversity on the second level. We will see that this is less motivated by a better *preservation* of diversity during run but by a control of a diversity *level* that is depending on the configuration of the selection process. Instead of a premature loss of diversity we observed an inherent increase of *structural* diversity with our linear GP approach.

The simplest form of diversity preservation might be to seed randomly created individuals regularly into the population during runtime. In [10] a more explicit maintenance of diversity has been proposed by creating and seeding individuals that fill "gaps" of under-represented areas in genotype space. However, experimental evidence has not been given for this rather complicated and computationally expensive approach. Until now, explicit diversity control is a rarely investigated technique in genetic programming. Only recently, de Jong et al. could improve parsimony pressure through Pareto-selection of fitness and tree size by adding a (third) diversity objective. In contrast, a more *implicit control* of genetic diversity offer semi-isolated sub-population, called *demes*, that are widely used in the area of evolutionary computation (see, e.g., [17, 3]). Only a certain percentage of individuals is allowed here to migrate from a deme into another deme during each generation.

The second major objective of this paper refers to the structural distance between a parent program and its offspring, i.e., the *variation distance* induced by a variation operator. While the effect on the overall program structure may be controlled by designing appropriate genetic operators—as demonstrated in [4] for linear GP—the change of the effective, i.e., fitness-relevant, part of code has to be measured explicitly. The latter may differ significantly from the amount of absolute change on structural level. By monitoring the *effective* variation distance during runtime variation step sizes may be restricted more precisely in relation to the change of program semantics. In particular, genetic operations that disrupt the effective part of program completely may be avoided. We will see that even strong restrictions of the maximum allowed mutation distance do not restrict freedom of variation significantly.

Designing variation operators that walk on the fitness landscape in small steps favors the use of mutations. We apply a variant of linear GP that uses *effective* mutations exclusively [4]. These operations reduce the number of neutral variations by concentrating on effective instructions. In this way, step sizes performed by effective mutations are more closely related to the effective distance.

2 Basics on Linear GP

Programs in tree-based genetic programming (TGP) denote expressions from a functional programming language like LISP [11]. In linear genetic programming (LGP) [1], instead, the program representation consists of variable-length sequences of instructions from an imperative programming language like machine code [12] or C [3]. An instruction incorporates an operator that manipulates variables (*registers*) and constants and assigns the result to a destination register, e.g., $r_i := r_j + 1$. The imperative program code is divided into *effective* and *non-effective* instructions while only the effective code may influence program behavior. The non-effective instructions are referred to as *introns*. This separation of instructions results from the linear program *structure*—not from program *execution*—and can be computed efficiently during runtime [3]. In the following program example only the instructions that are *not* commented can have an influence on the final output that is hold in register r_0 here at the end of execution. *pdiv* and *ppow* represent protected versions of the division and the exponentiation operator that map undefined input ranges to high penal values.

```
void gp(r)
  double r[5];
{
   ...

   r[1] = r[4] * r[0];
   r[0] = ppow(r[1], 2);
// r[4] = r[2] * r[4];
   r[4] = pdiv(r[2], r[0]);
// r[0] = r[3] - 1;
// r[1] = r[2] * r[4];
// r[1] = r[0] + r[1];
// r[0] = r[3] - 5;
// r[2] = ppow(r[1], r[0]);
   r[2] = r[3] - r[4];
   r[4] = r[2] - 1;
   r[0] = r[4] * r[3];
// r[4] = r[0] + 2;
// r[1] = pdiv(r[0], r[3]);
// r[1] = r[1] + 9;
// r[4] = ppow(r[4], r[3]);
   r[3] = r[2] + r[3];
   r[4] = pdiv(r[2], 7);
// r[2] = r[2] * r[4];
   r[0] = r[0] + r[4];
   r[0] = r[0] - r[3];
}
```

We distinguish two different variants of linear GP in this work. While the standard approach applies recombination by crossover to vary program length the other approach works with mutations exclusively. The *linear crossover* operator exchanges two arbitrarily long sub-sequences of instructions between two individuals. If the operation cannot be executed because one offspring would exceed the maximum length crossover is performed with equally long sub-sequences. *Macro mutations* include deletions or insertions of single (full) instructions here and represent an alternative growth operator to crossover. *Micro mutations* change the smallest program components that comprise a single operator, a register or a constant.

Effective mutations always guarantee that the effective code is altered. This reduces the probability that a mutation stays neutral in term of a fitness change. If an instruction is inserted its destination register is chosen in such a way that the instruction is *effective* at the corresponding program position.

3 Structural Distance Metrics for Linear Genetic Programs

3.1 Edit Distance

The *string edit distance* [6] operates on arbitrarily sequences of characters. It measures the distance between two strings by counting the number of basic operations—including insertion and exchange of single elements—that are necessary to transform one string into another. Usually each operation is assigned the same costs (one) independently from the affected type of element. The string edit distance is calculated in time $O(n^2)$ [6] with n denotes the maximum number of components that are compared between two individual programs.

We apply the edit distance to measure the structural distance between the effective part of programs (*effective distance*) because a difference in effective code may be more directly related to a difference in program behavior. In contrast to a distance metric regarding full program code (*absolute distance*) this includes some information on program semantics. The relatively high rate of non-effective code—about 50-60% in the experiments below—that emerges with recombination, makes clear that there is a stronger correlation between semantic and structural distance when restricting the distance measure to the effective code only. It is important to realize that the effective distance is not included in the absolute distance. Actually, two programs can have a small absolute distance while their effective distance is comparatively large (see Section 6). On the other hand, two equally effective programs might differ significantly in their non-effective code.

Additionally, we regard the sequence of operators (from the effective instructions) only. The sequence corresponding to the example program from Section 2 is

$(-, +, /, +, *, -, -, /, pow, *)$.

when starting with the last effective instruction. The distance of effective operator symbols has proven to be sufficiently precise to differentiate between program structures provided that the used operator set is not too small. This is due to the observation that in most cases the modification of an effective instruction changes the effectivity status of at least one instruction (see Section 8). Note that this is only true for the effective distance of operators. The absolute operator sequence would not be altered by the exchange of single registers. Because identical exchanges of program components are avoided updating a constant within an effective instruction is the only type of variation that is not registered at all. Actually, a registration of absolutely every structural difference should not be necessary if we take into account that the correlation between semantic and structural distance is probabilistic.

Beyond that, less different genotypes are distinguished by this metric that represent the same phenotype (fitness). By including the program registers into distance calculation the distance measure would become more ambiguous. This is true because most registers are used temporarily only during calculation and may be replaced partly by others without altering the behavior of a program. In fact, only the last assignment to an output register in (effective) program and all readings of an input register before its contents is overwritten for the first time are invariable.

Another important motivation for restricting the number of components in compared programs is that time of distance calculation is reduced significantly. By regarding only the sequences of effective operators calculation time of edit distance directly depends on the (average) number n of effective instructions (*effective length*) only. Depending on the percentage of non-effective code there are k times more elements to compare if one regards the full sequence of operators in programs. Extending the distance metric to registers and constants of instructions, again, results in a factor of 4 maximum. In conclusion, computational cost of the edit distance would increase by a total factor of $(4k)^2$ up to $O(16k^2 \cdot n^2)$.

3.2 Alternative Distance Metrics

In all experiments of this paper we have applied the edit distance metric as described above. However, even if a reduction of identifying program elements already accelerates distance calculation significantly, there are more efficient metrics possible on linear genetic programs.

One step toward a more efficient distance calculation between two effective programs is to give up the order of operators and to compare only the numbers of each operator type. Then program distance may be reduced to the Manhattan distance between two pattern vectors v and w of equal length n (n = size of operator set). Each vector position v_i represents the frequency of an operator type in the genetic program corresponding to v. The *Manhattan distance* is measured along axes at right angles and simply calculates the sum of absolute differences between equal vector positions, i.e., $\delta(v, w) = \sum_{i=1}^{n} |v_i - w_i|$. This requires runtime $O(n)$ only. Note that n is much smaller here than for the edit distance (n = maximum program length). Although the accuracy of this structural distance is definitely lower than edit distance it has proven to be sufficient for an explicit control of diversity.

Another, more efficient distance metric than edit distance is applicable for controlling mutation step sizes. If a certain program position is mutated, we may calculate how many of the depending *previous* instructions in program have changed their effectivity status. This is exactly the *Hamming distance* between the status flags and takes time $O(n)$ only with n is the maximum program length here. A control of variation distance is possible with this metric, even if it is more imprecise than edit distance. That is distance zero occurs more frequently because more variations are not registered. Note that efficiency of distance calculation is less important for controlling variation distance than for controlling diversity (see below).

4 Semantic Distance Metrics

The most obvious metric to evaluate the behavior of a genetic program is the fitness function. This usually calculates the distance of the predicted outputs $gp(i_k)$ returned by a program and the desired outputs given by n fitness cases, i.e., input-output examples (i_k, o_k). For example, in Equation 1 this is simply the Manhattan distance between the two output vectors.

$$fit(gp) = \sum_{k=1}^{n} |gp(i_k) - o_k| \tag{1}$$

Correspondingly, the semantic differences between two genetic programs may be expressed by their relative *fitness distance* (Equation 2). In this case, the quality of solving the overall problem is considered.

$$\delta_{fit}(gp_1, gp_2) = |fit(gp_1) - fit(gp_2)| \tag{2}$$

Another possibility is to compare the outputs of two programs directly. The same distance metric as in the fitness function may be used for computing the distance between the output vectors of programs (see Equation 3). In the following this will be referred to as *output distance*. Note that the relative output distance between two programs is independent from their performance in terms of solving a prediction task. Actually, two programs may have a similar fitness while their output behavior differs significantly, e.g., different subsets of the training data may be approximated with a different accuracy.

$$\delta_{out}(gp_1, gp_2) = \sum_{k=1}^{n} |gp_1(i_k) - gp_2(i_k)| \tag{3}$$

Analogously, for discrete problems like classifications where the fitness function calculates a classification error, i.e., the number of wrongly classified examples, a *Boolean output distance* is defined as follows:

$$\delta_{boolout}(gp_1, gp_2) = \sum_{\substack{class(gp_1(i_k)) \neq class(gp_2(i_k)) \\ k=1,..,n}} 1 \tag{4}$$

Function *class* in Equation 4 hides the classification method that maps the continuous program outputs to discrete class identifiers. The Boolean output distance may be used with continuous problems, too. In this case, the distance between non-equal outputs will be constantly one.

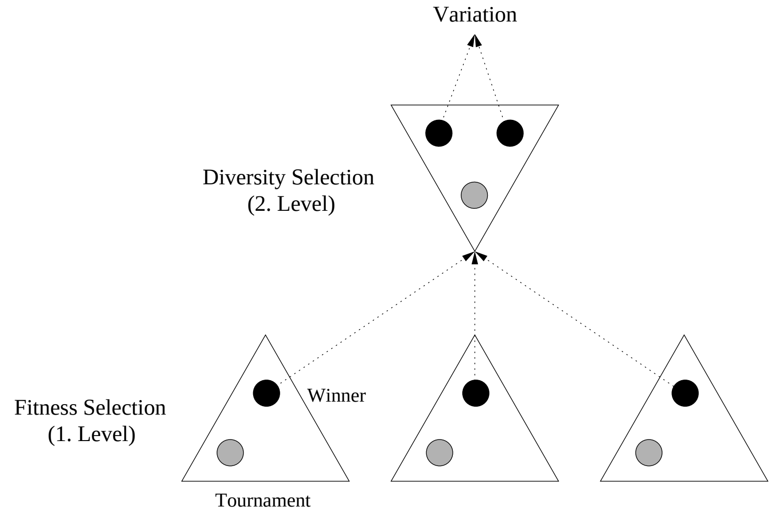

Figure 1: Two-level selection process.

5 Diversity Selection

The effective edit distance between programs is applied for an active control of *genotype diversity*, that is the average *structural* distance between individuals in population. Therefore, we introduce the two-level tournament selection shown in Figure 1. On the first level, individuals are selected by fitness. On the second level, the two individuals with *maximum* distance are chosen among three *fitter* individuals. While an *absolute* measure like fitness may be compared between two individuals selection by a *relative* measure like distance or diversity necessarily requires a minimum of *three* individuals. In general, two out of n individuals are selected with the largest sum of distances to the $n-1$ other individuals. While selection pressure on the first level depends on the *size* of fitness tournaments the pressure of diversity selection on the second level is controlled by the *number* of these tournaments. Additionally, a second control parameter reduces the selection rate on second level.

The number of fitness calculations and the processing time, respectively, do not increase with the number of tournaments if the fitness of individuals is saved and is updated only after variation. However, diversity selection itself becomes more computationally expensive the more individuals participate in it. Since n individuals require $\binom{n}{2}$ distance calculations an efficient distance metric is needed all the more.

One advantage of the two-level selection process introduced here over applying fitness selection *or* diversity selection independently from each other on the *same* level is that the rate of fitness selection is not reduced. The multi-objective selection method prefers individuals that are fit *and* diverse in relation to others at the same time. Selecting individuals only by diversity for a certain probability, instead, does not result in more different directions among *better* solutions in the population. Dittrich et al. [5] report on a spontaneous formation of groups when selecting the most distant of three individuals that are represented by single real numbers.

Selection for a linear combination of both objectives, fitness and diversity, as this is often practiced with fitness and length (parsimony pressure), would require an appropriate weighting. This, however, is rather difficult to find. Another problem is that fitness and diversity still have the same priority. With the two-level selection, instead, fitness selection is not only decoupled from diversity selection but has always a higher priority.

An explicit control of structural diversity increases the average distance of individuals. Graphically, the population spreads more widely over the fitness landscape. Thus, there is a lower probability that the evolutionary process gets stuck in a local minimum and more different search directions may be explored in parallel.

While increasing the effective distance in population affects the diversity of problem solutions, the absolute distance reassures a more general diversity including the non-effective code. Selection for absolute distance has also been practiced but found to improve results less (undocumented). Apart from the fact that this is more time-consuming it confirms that the absolute distance measures the effective distance only imprecise (see Section 3).

Increasing the average distance between programs by diversity selection has the side-effect of accelerating the growth of (effective) program length. In order to avoid that this may influence results, we select for (effective) edit distance *minus* distance in effective length. This is possible because both edit distance and length distance operate on instruction (operator) level here. By doing so, difference in length is no longer rewarded directly during selection. As a result, the difference in average effective length with and without applying diversity selection becomes negligible.

The diversity level can be *lowered*, too, by a selection for *minimum* distance. This might have a positive influence if population diversity is already quite high, i.e., because of a low fitness selection pressure or a low reproduction rate. In this case, especially crossover might profit from a reduction of diversity such that variation step sizes become indirectly smaller. In our experiments, however, selection for minimum distance resulted in the opposite (negative) effect as selection for maximum distance (undocumented).

Selection for *phenotype diversity*, i.e., for maximum *semantic* distance of programs, has been practiced by comparison. Semantic diversity is controlled by using the output distance defined in Section 4. A selection for maximum output distance may be implemented efficiently in both calculation time and memory usage, if only the outputs of individuals are saved that participate in the current tournament(s).

Selection for fitness distance has been found less suitable, instead. Even if both program fitness as well as program outputs are related to an absolute optimum, the relative output distance between programs allows semantic differences in much more detail. Moreover, increasing relative fitness distance necessarily increases the diversity of fitness values in population which might promote worse solutions. Finally, selection by fitness distance is almost without any effect on problems that implicate a rather narrow and discrete fitness distribution.

6 Control of Variation Distance

One property of program representations in GP is that already smallest variations on structural level may effect program semantics and fitness heavily. In linear GP these variations include the exchange of a single operator or register. Several instructions that precede a varied instruction in a program may become effective or non-effective respectively. In this way, micro mutations may not only affect the fitness but the flow of data in linear genetic programs. This is in contrast to tree-based GP where point mutations manipulate the contents of single *nodes* but do not change *edges* of the tree structure. Even if bigger variations of program behavior are less likely with smaller structural variation steps, this effect is rather undesirable.

An *implicit control* of structural variation distance is to impose respective restrictions on the variation operators [4]. For instance, in linear GP the maximum length of the instruction segment that is exchanged during recombination may be limited. Alternatively, macro mutations that insert or delete single instructions may be used exclusively without using recombination. Moreover, by concentrating variations on the effective code structural variations become more closely related to semantic variations. This is true because only the degree of variation on the effective code decides about the difference in fitness. Unfortunately, a variation operator—even if it is operating on the effective code exclusively—can only guarantee for the *absolute* program structure that a certain maximum variation step size is not exceeded. Variation steps on the *effective* code, instead, may still be much bigger though appear with a lower probability.

One concern of this contribution is an *explicit control* of variation step size. That means the structural distance between parent and offspring induced by a variation operator is measured explicitly in relation to a distance metric. Restrictions of *absolute* variation distance can be controlled implicitly from the operator side (as just discussed). Our interest here is in controlling the *effective* variation distance. Therefore, the variation of a parent program is repeated until the measured distance of the offspring falls below a *maximum threshold.*

The following example represents the result of applying a micro mutation to the program example from Section 2. In instruction number 10 from the top the first operand register `r[3]` has been exchanged by register `r[2]`. As a consequence, five preceding (formerly non-effective) instructions become effective which corresponds to an effective variation distance of five.

```
void gp(r)
  double r[5];
{
   ...

   r[1] = r[4] * r[0];
   r[0] = ppow(r[1], 2);
// r[4] = r[2] * r[4];
   r[4] = pdiv(r[2], r[0]);
   r[0] = r[3] - 1;
   r[1] = r[2] * r[4];
   r[1] = r[0] + r[1];
   r[0] = r[3] - 5;
   r[2] = ppow(r[1], r[0]);
   r[2] = r[2] - r[4];      <- Effective mutation point
   r[4] = r[2] - 1;
   r[0] = r[4] * r[3];
// r[4] = r[0] + 2;
// r[1] = pdiv(r[0], r[3]);
// r[1] = r[1] + 9;
// r[4] = ppow(r[4], r[3]);
```

```
   r[3] = r[2] + r[3];
   r[4] = pdiv(r[2], 7);
// r[2] = r[2] * r[4];
   r[0] = r[0] + r[4];
   r[0] = r[0] - r[3];
}
```

Besides restricting the maximum size of variation steps, we tested a *minimum threshold* as well. If small variation steps are avoided or, at least, reduced in frequency, evolutionary progress might be accelerated. However, this is only true if the minimum step size is not too large for the evolutionary process to continuously approach good solutions. For a higher efficiency and for the fact that most variations are macro variations, i.e., operate on instruction level, we have decided for a distance metric in Section 3 that does not register every micro variation, i.e., single modification *within* an instruction. As a result, depending on the variation operator many variations result in either distance one or are not even registered at all (distance zero). For that reason and from our experimental experience (see Section 8) smallest variations are essential here.

Using an explicit control of fitness distance between parent and offspring, instead, requires an additional fitness calculation for each iteration and can become computationally expensive, especially if a larger number of fitness cases is involved. By comparison, a structural distance like edit distance has to be re-calculated only once after each repeated variation while its computational costs do not directly depend on the number of fitness cases. It is also difficult to find appropriate maximum thresholds for fitness distance because those are usually problem-specific. Finally, it is not sensible to restrict *positive* fitness changes at all.

7 Experimental Setup

All techniques discussed above have been tested with three benchmark problems including an approximation, a classification, and a Boolean problem. Table 1 summarizes problem attributes and problem-specific parameter adjustments of our LGP system.

Problem ID	*sinpoly*	*iris*	*8-parity*
Problem type	Approximation	Classification	Boolean function
Problem function	$sin(x) \times x + 5$	real-world data set	even-N-parity (N=8)
Input range	$[-5, 5]$	$[0, 8)$	$\{0, 1\}$
Output range	$[0, 7)$	$\{0, 1, 2\}$	$\{0, 1\}$
Number of inputs	1	4	8
Number of outputs	1	1	1
Number of registers	1+4	4+2	8+0
Number of examples	100	150	256
Fitness function	SSE	CE	SE
Number of generations	500	500	250
Instruction set	$\{+, -, \times, /, x^y\}$	$\{+, -, \times, /, if >, if \leq\}$	{AND, OR, NOT, if}
Set of constants	$\{1, .., 9\}$	$\{1, .., 9\}$	$\{0, 1\}$

Table 1: Problem-specific parameter settings

The first problem is referred to as *sinpoly* in the following and denotes an approximation of the sinus polynom $sin(x) \times x + 5$. The provided set of instructions does not include any trigonomical functions. Thus, given the facts that the maximum length of genetic programs is limited and that the *sine* function is defined by an infinite Taylor-series the optimum cannot be found. Besides the input register—that is identical to the output register—there are four additional calculation registers used with this problem. This additional program memory becomes very important in linear GP if the number of inputs is low by definition. With only one register calculation potential is too much restricted in any case. The program fitness is the *sum of square errors* (SSE) between its predicted outputs and the given example outputs. 100 fitness cases have been selected uniformly distributed over input range $[-5, 5]$.

The second problem *iris* is a popular classification data set that originates from the *UCI Machine Learning Repository* [2]. The real-world data contains 3 classes of 50 instances each, where each class refers to a type of iris plant. Fitness equals the *classification error* (CE), i.e. the number of wrongly classified inputs. A program output $p(i_k)$ is considered as correct if the distance to the desired class identifier $o_k \in \{0, 1, 2\}$ is smaller than 0.1, i.e. $|p(i_k) - o_k| < 0.1$. Note that solution finding becomes easier if this error threshold is extended to the maximum (0.5 here).

Finally, we have tested a parity function of dimension eight (*even-8-parity*). This function computes one if the number of set input bits is even, otherwise the output is zero. The Boolean branch in the instruction set is essential for a high number of successful runs.

General configurations of our linear GP system are given in Table 2. Only one variation operator is applied at a time. Macro mutations *or* crossover are always applied for 75% while micro mutations cover the remaining 25%. Macro mutations include two times more insertions than deletions here. This explicit growth tendency of the operator has proven necessary for a sufficient increase of program complexity since only one instruction is modified at a time. Especially when *effective* mutations are used programs grow much more slowly in size [4].

Parameter	Setting
Population size	2000 programs
Maximum program length	200 instructions
Initial program length	2–20 instructions
Maximum fitness	1000
Fitness tournament size	4
Reproduction	100%
Micro mutation	25%
Macro mutation	75%
Instruction deletion	33%
Instruction insertion	67%
Crossover	75%

Table 2: General parameter settings.

8 Results

8.1 Structural and Semantic Program Distance

First of all, we demonstrate experimentally that there is a causal connection between the structural distance and the semantic distance (fitness distance) of linear genetic programs. By doing so, we apply the edit distance metric on sequences of effective instruction operators as defined in Section 3.

In the first experiment distances of 2000 pairs of *randomly* selected individuals have been registered in each generation. Figures 2, 3, and 4 visualize the resulting relation of (effective) program distance and fitness distance together with the corresponding distributions of program distances. In case of all test problems there is a clear positive correlation between program distance and fitness distance for the vast majority of measured distances. Program distances above a certain maximum fitness distance, however, occur only very rarely. This is why correlation becomes more irregular. Actually, structural and semantic distance are found negatively correlated in this distance range. In principle similar phenomena are observed here with the crossover-based and the mutation-based variant of linear GP (see Section 2).

In a second experiment that is relevant in this context we investigate structural *variation* distance, i.e., the distance between parent and child or, more precisely, the distance of a modified individual from its original state. Let us have a look at Figures 5, 6, and 7. There is a strong causality between program distance and fitness distance for almost all meassured variation distances. In general, correlation between structural and semantic distances has been found even more clearly between parent and offspring than between arbitrary individuals. Moreover, the distribution range of distances—induced by crossover or effective mutations—is significantly smaller than in the first experiment, as might be expected. That means the structural distance between parent and child is smaller, on average, than between two arbitrary individuals or between two parents. In general, variation distances occur the more frequently the shorter they are.

In crossover runs a relatively high amount of operations results in effective distance zero, especially with the two discrete problems *iris* and *8-parity*. This is obviously due to the much higher intron rate that emerges with this variation operator. As already introduced in Section 2 effective mutations vary the effective code of programs exclusively. In runs using (effective) mutations distance zero is mostly caused by effective *micro* mutations that affect a single register, operator or constant. Not all exchanges of registers in effective instructions change the effectivity of instructions and, thus, the sequence of operators necessarily. The reason is that effectivity is very often guaranteed by more than one succeeding instruction. However, still many of the 25% micro mutations induce a variation distance that is larger than zero.

Furthermore, distance distribution Figures 5, 6, and 7 show that almost two thirds of all effective mutations induce distance one. Interestingly, even though 75% macro mutations delete or insert full (effective) instructions—including one operator and three registers at maximum—the effectivity of other (preceding) instructions changes for less than one third only. Obviously, evolution develops program structures whose effectivity is less fragile against variations of that kind. When using crossover the proportion of non-effective instructions in a program acts as a second implicit mechanism that reduces variation

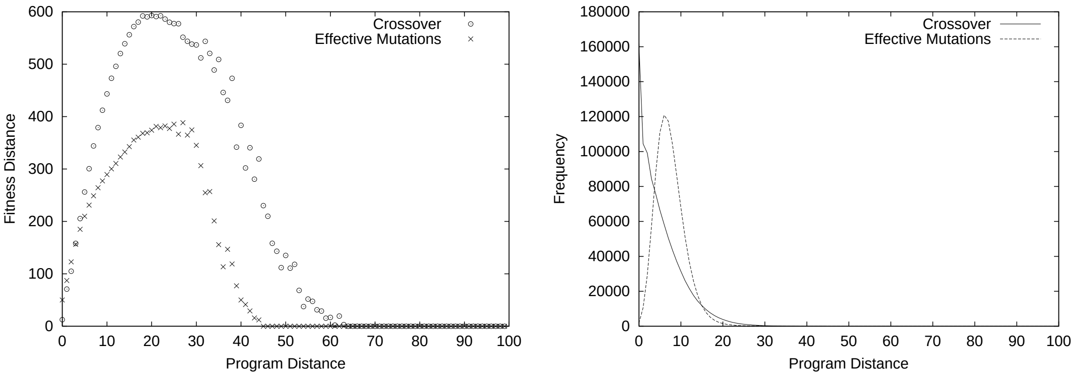

Figure 2: *sinpoly*: Distance relation and distance distribution for two randomly selected individuals in crossover runs and in runs using effective mutations. Average figures over 100 runs.

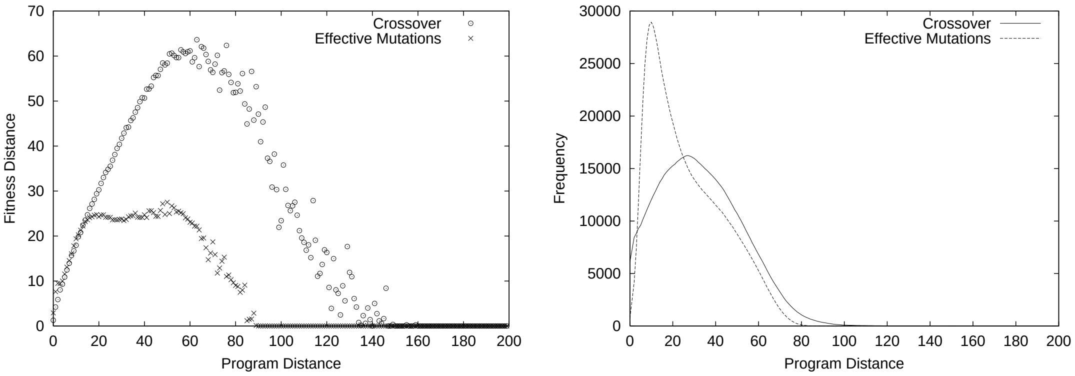

Figure 3: *iris*: Distance relation and distance distribution.

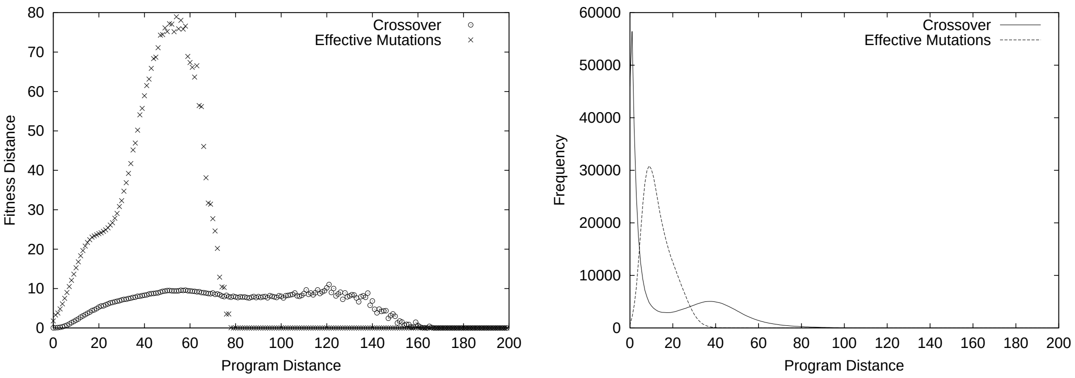

Figure 4: *8-parity*: Distance relation and distance distribution.

strength.

Even exchanging an operator may induce a variation distance that is larger than zero. If the new operator requires a different number of (register) parameters than the former operator, registers may be deactivated or reactivated within an instruction. Preceding instructions in program that depend on such a register parameter may change their effectivity status then. This, in turn, may have an influence on the variation distance. It is a

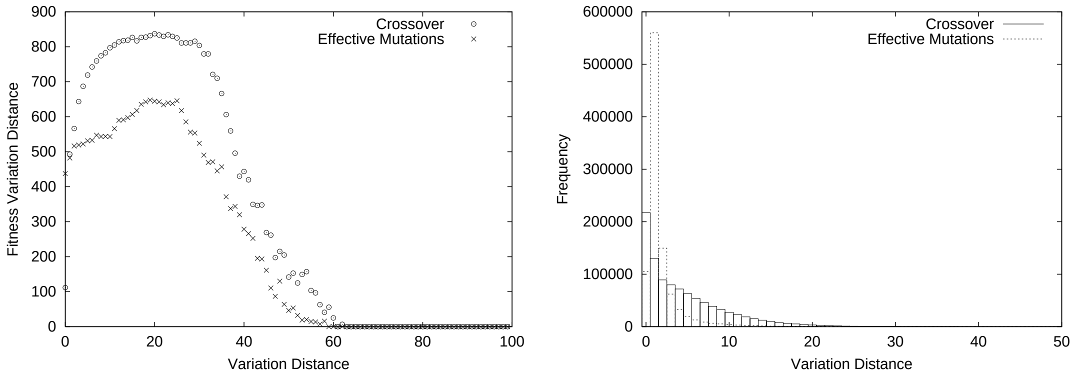

Figure 5: *sinpoly*: Variation distance relation and variation distance distribution. Average figures over 100 runs.

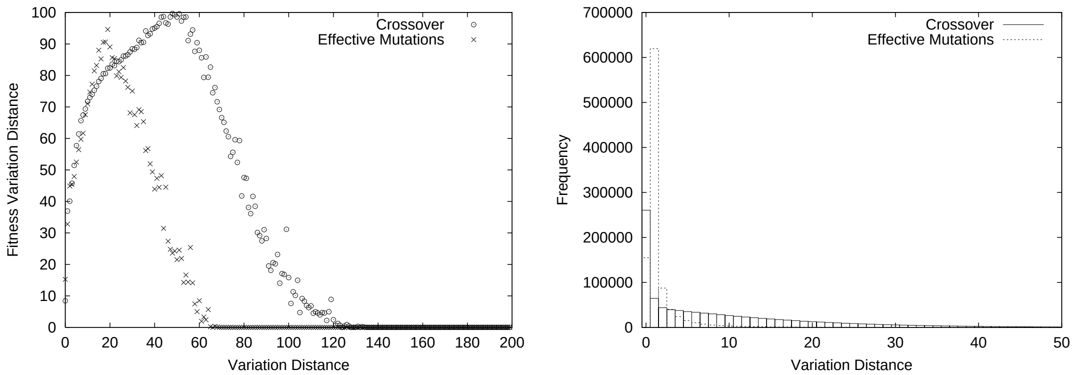

Figure 6: *iris*: Variation distance relation and variation distance distribution.

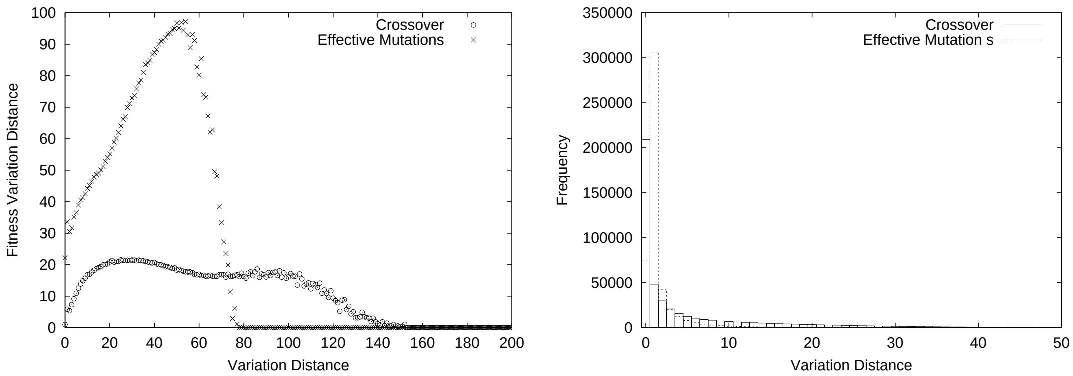

Figure 7: *8-parity*: Variation distance relation and variation distance distribution.

feature of linear GP that non-effective code in form of instructions, registers, or constants remains in a program as, so called, *structural introns* that may be reactivated later by evolution. In tree-based GP, by comparison, a sub-tree is lost if the arity of its root node is reduced because of the constraints of the tree structure.

8.2 Structural Diversity Selection

We will now report on our results obtained with diversity selection. Table 3 shows average error rates obtained with and without selecting for structural diversity for the three test problems introduced in Section 7. Different selection pressures have been tested. For the minimum number of fitness tournaments (three) that are necessary for a diversity selection on second-level (see Section 5) we used selection probabilities 50% and 100%. Higher selection pressures are induced by increasing the number of tournaments up to four or eight (*sinpoly* only).

It is conspicuous that in all three test cases linear GP works significantly better by using (effective) macro mutations instead of crossover. In [4] we have already demonstrated that the linear program representation, in particular, is much more suitable for being developed by mutations only, especially if those are directed towards effective instructions. Nonetheless, the experiments with linear crossover show here that diversity selection is not depending on a special type of variation. Moreover, the application of this technique is demonstrated with a population-dependent operator. For each problem and variation operator performance increases continuously with the influence of diversity selection in Table 3. The highest selection pressures tested result in a more than two times better prediction error, on average, than without increasing diversity explicitly.

Variation	Selection		sinpoly		iris		8-parity	
	%	#T	*mean* (*std*)	Δ (%)	*mean* (*std*)	Δ (%)	*mean* (*std*)	Δ (%)
Crossover	0	2	3.01 (0.35)	0	2.11 (0.10)	0	58 (3.4)	0
	50	3	2.89 (0.34)	4	1.42 (0.08)	33	35 (2.4)	40
	100	3	2.77 (0.34)	8	1.17 (0.07)	44	27 (2.2)	53
	100	4	1.96 (0.22)	35	1.09 (0.07)	48	19 (1.8)	67
	100	8	0.69 (0.06)	77	—	—	—	—
Effective Mutations	0	2	0.45 (0.04)	0	0.84 (0.06)	0	15 (1.2)	0
	50	3	0.43 (0.03)	4	0.63 (0.05)	25	12 (1.0)	20
	100	3	0.30 (0.02)	33	0.60 (0.05)	29	10 (1.1)	33
	100	4	0.23 (0.02)	49	0.33 (0.04)	61	7 (0.8)	53
	100	8	0.17 (0.01)	62	—	—	—	—

Table 3: Second-level selection for *structural* diversity with different selection pressures. Selection pressure controlled by selection probability and number of fitness tournaments (T). Average error over 200 runs. Statistical standard error in parenthesis. Percental difference from baseline results.

Diversity of a population is defined as the average distance between two randomly selected individuals. Figures 8–10 illustrate the development of structural diversity during run for different selection pressures. Obviously, the higher the selection pressure is adjusted the higher is the diversity. We can see that the average effective program distance does not drop even without applying diversity selection. This is true even with the applied 100 percent reproduction and a selection pressure of four individuals per tournament. On the contrary, with crossover diversity increases until a certain level and stays rather constant then. With effective mutations increase is more linear.

Two major reasons can be found to explain this behavior: First, genetic programming is working with a variable-length representation that grows continuously during a run.

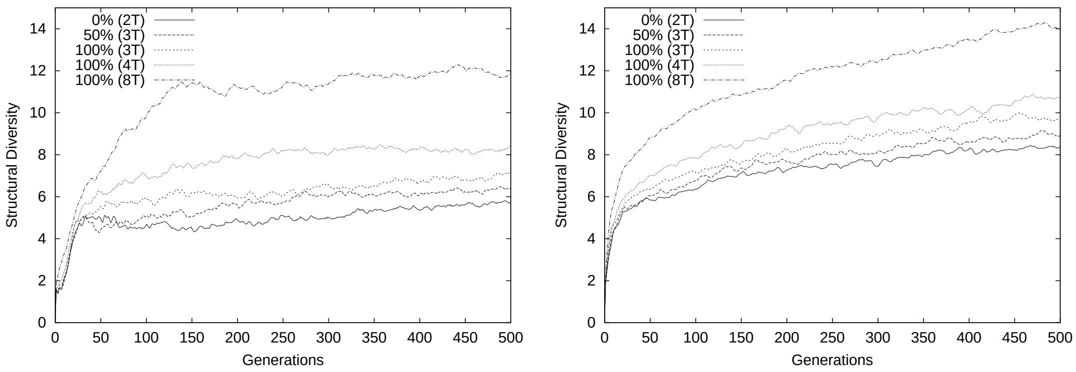

Figure 8: *sinpoly*: Diversity levels after structural diversity selection with different selection pressures. Selection pressure controlled by selection probability and number of fitness tournaments (T). Average figures over 100 runs. Macro variation by crossover (left) or effective mutations (right).

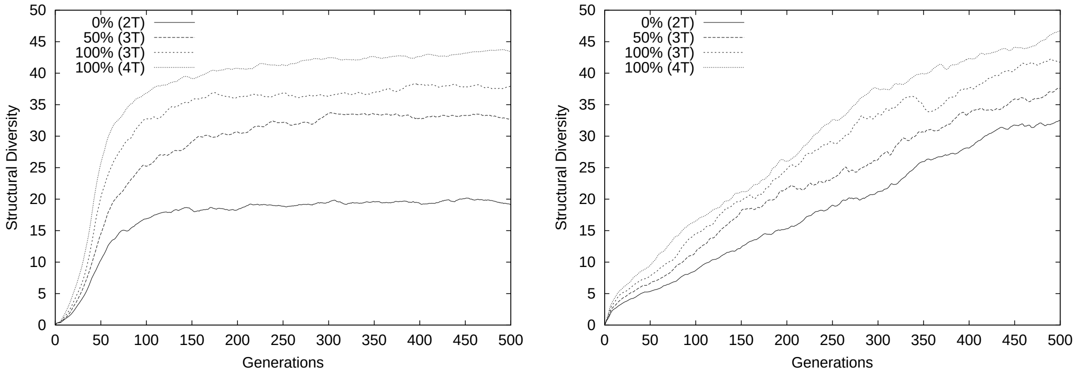

Figure 9: *iris*: Diversity levels after structural diversity selection with different selection pressures. Macro variation by crossover (left) or effective mutations (right).

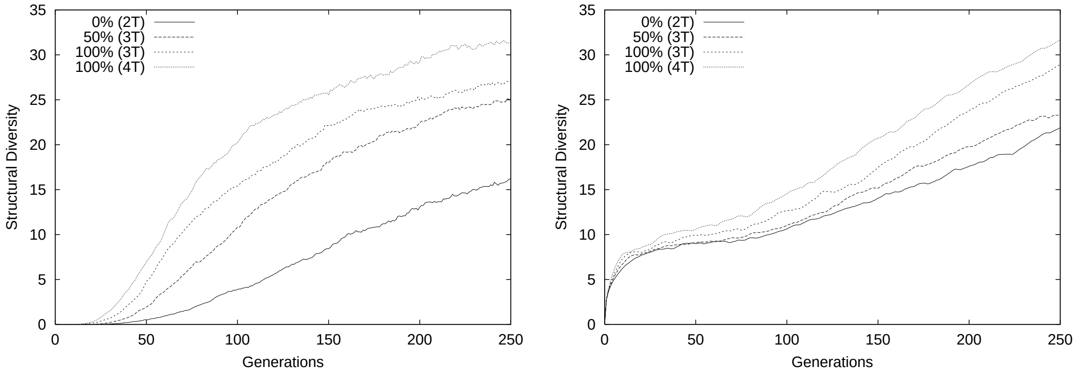

Figure 10: *8-parity*: Diversity levels after structural diversity selection with different selection pressures. Macro variation by crossover (left) or effective mutations (right).

In linear GP this is especially true for the *effective* program length which may still grow even if the absolute length has reached the maximum. The longer effective programs become the bigger effective distances are possible. Actually, the growth of effective code is restricted earlier with crossover by the maximum limit because a much higher proportion of non-effective code emerges with this operator.

Second, both forms of variation, linear crossover and effective mutation, maintain program diversity over a run implicitly. This is true even without applying selection for diversity explicitly. For linear crossover the reason might be in its high variation strength. Additionally, the higher amount of non-effective code contributes to a preservation of (effective) code diversity here. The non-effective part of code may differ more strongly between programs than the effective code because this intron code is independent from fitness and selection pressure (on fitness), respectively.

By using mutations exclusively, instead, a high degree of innovation is introduced continuously into the population. This leads to a higher diversity than it occurs with crossover. The stronger it is selected for diversity, however, the more diversity is gaining ground in crossover runs. Obviously, there is a stronger influence of diversity selection on population diversity with crossover than with mutations. In comparison to mutations the success of recombination depends more strongly on the composition (diversity) of the genetic material in the population. The more different the recombined solutions are the higher is the expected innovativity of their offsprings.

8.3 Semantic Diversity Selection

The computational overhead of a structural distance control has been found affordable for linear genetic programs, especially if the order of instructions is not regarded (see Section 3). In order to justify its usage more generally we test a semantic diversity selection for comparison. Semantic diversity is defined here as the average *output* distance of two individuals that have been randomly selected from the population (see Section 4). For each problem the same distance metric has been used as in the corresponding fitness function (see Table 1).

Variation	Selection		sinpoly		iris		8-parity	
	%	#T	*mean* (*std*)	Δ (%)	*mean* (*std*)	Δ (%)	*mean* (*std*)	Δ (%)
Crossover	0	2	3.09 (0.23)	0	2.11 (0.10)	0	58 (3.4)	0
	50	3	2.40 (0.22)	22	1.82 (0.09)	14	40 (2.5)	31
	100	3	3.51 (0.25)	-12	1.62 (0.08)	23	46 (3.1)	21
	100	4	3.42 (0.25)	-10	1.80 (0.09)	15	42 (2.8)	28
Effective	0	2	0.40 (0.03)	0	0.84 (0.06)	0	15 (1.2)	0
Mutations	50	3	0.33 (0.02)	18	0.77 (0.06)	8	13 (1.2)	13
	100	3	0.43 (0.03)	-7	0.68 (0.05)	19	12 (1.1)	20
	100	4	0.49 (0.05)	-18	0.42 (0.05)	50	9 (0.9)	40

Table 4: Second-level selection for *semantic* diversity with different selection pressures. Selection pressure controlled by selection probability and number of fitness tournaments (T). Average error over 200 runs. Statistical standard error in parenthesis. Percental difference from baseline results.

When comparing results in Table 4 with the results in Table 3, it follows that semantic diversity selection, in general, has a smaller effect on prediction quality than selection for structural diversity. Especially the continuous problem *sinpoly* could not be solved more successfully by semantic diversity selection. For the two discrete problems we can observe a higher influence on effective mutation runs than on crossover runs.

One explanation is that, in contrast to program structure, program semantics is related to a unique optimum. For the program outputs this is the set of desired outputs given by the fitness cases. Hence, the number of possibly different output patterns reduces the closer fitness approaches optimum zero. Compared with this diversity of program structure is much more independent from fitness.

If we compare the development of semantic diversity over run a decrease is observed with the two discrete problems, *iris* and *parity* (see Figures 12 and 13). That means the outputs of programs become more similar to the desired outputs. The progression of semantic diversity, that has been found with the continuous problem *sinpoly*, is increasing, instead, (see Figure 11). Obviously, there is no clear convergence of outputs for the majority of programs in those runs.

Interestingly, with all problems selection for *maximum* output distance results in a *decrease* of semantic diversity. This is the opposite direction of improvements as observed when selecting for structural diversity (see Figures 8-10). The more different programs are selected for variation the better solutions and, thus, the more similar program outputs become, on average. Again this results from the fact that semantic distance, as opposed to the structural distance, is more dependent on fitness. Although semantic diversity decreases we may not talk about a "maintenance" of diversity because the level of semantic diversity is further decreased by the diversity control.

The relative influence on output distance is highest with *sinpoly* which might be due to the fact that the average distance increases here over generations. In this case, there is a higher potential for improvements (reduction) by diversity selection. With the other problem solutions become semantically much more similar while they adapt to the desired behavior, i.e., fitness cases. Selection for maximum semantic distance is contrary to this adaptation process and can only be advantageous to a certain extend.

Selection for *smallest* output distance has been tried, too, and resulted in exactly the opposite result. That is lower prediction performance and increase of semantic diversity.

8.4 Diversity and Fitness

Another interesting observation can be made when comparing the progress of best fitness and population diversity over a *single* run. First of all, there is no continuous increase of diversity as one might conclude from the average figures over multiple runs (see Figures 8–10). The development of structural diversity in Figures 15 and 16 is interrupted by sudden rapid drops (*diversity waves*). If a new best program or a program with an innovative piece of code occurs, that is during periods of fast fitness convergence, diversity decreases suddenly. This is possible because successful information spreads in the population within a few generations via reproduction and variation. How quickly program diversity recovers after such an event depends on how many generations have elapsed so far. The longer the current average program length is the sharper is the increase. Typical example runs in Figures 8–10 demonstrate that structural diversity increases on fitness plateaus, i.e., during periods where the best fitness stagnates. During that time the population individuals spread over the fitness landscape and explore the search space of programs more widely. The achieved diversity level depends on both the duration of the stagnation period and the current number of generations. Comparable runs have been found with both kinds of macro variations.

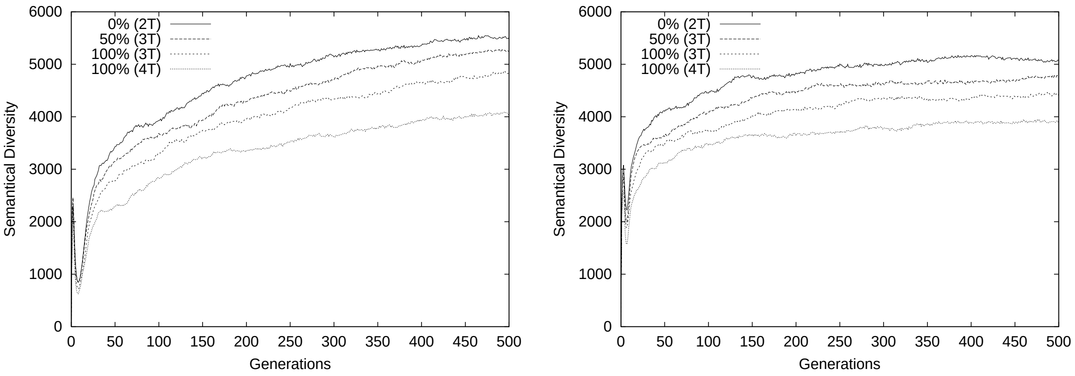

Figure 11: *sinpoly*: Diversity levels after semantic diversity selection with different selection pressures. Selection pressure controlled by selection probability and number of fitness tournaments (T). Average figures over 100 runs. Macro variation by crossover (left) or effective mutations (right).

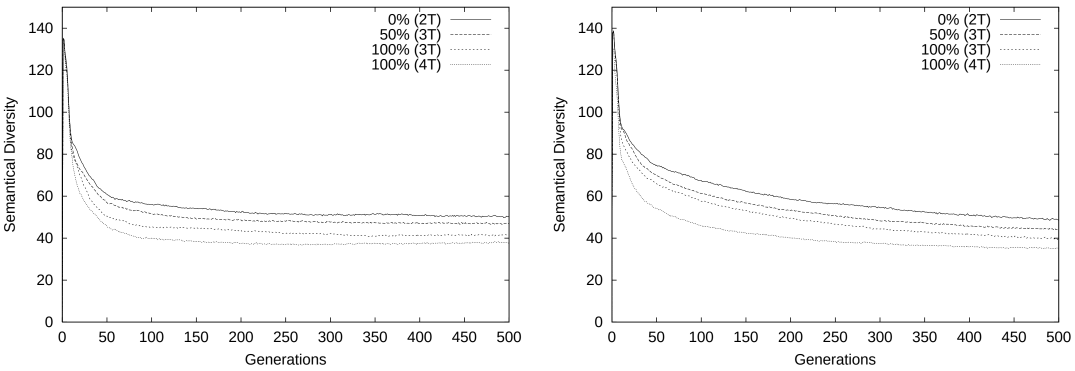

Figure 12: *iris*: Diversity levels after semantic diversity selection with different selection pressures. Macro variation by crossover (left) or effective mutations (right).

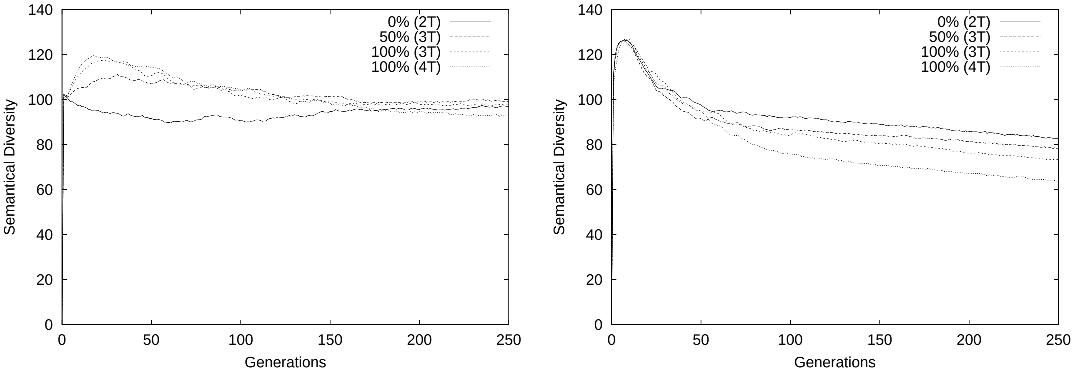

Figure 13: *8-parity*: Diversity levels after semantic diversity selection with different selection pressures. Macro variation by crossover (left) or effective mutations (right).

A different behavior has been observed with the continuous problem (*sinpoly*). Structural diversity progresses wave-like, too, but with a higher frequency and a smaller amplitude (see Figure 14). Similar correlations with best fitness as found with the two other test problems may exist but are hardly visible here because best fitness improves in smaller

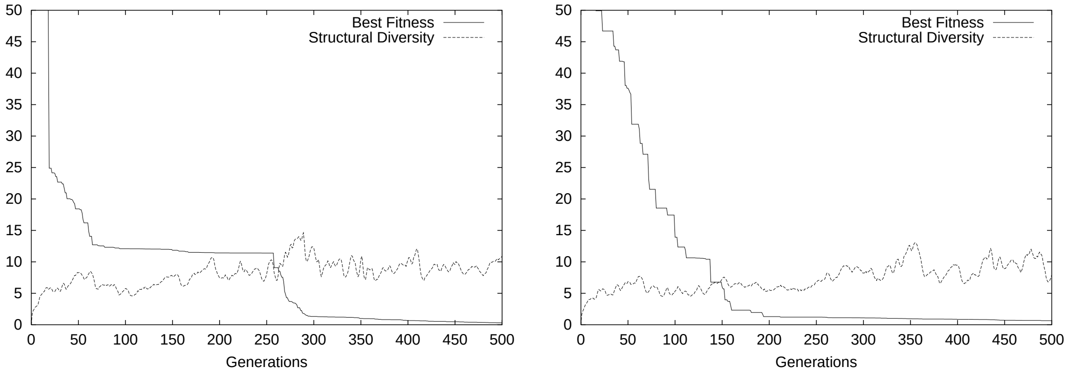

Figure 14: *sinpoly*: Relation between best fitness and structural diversity. Two typical example runs.

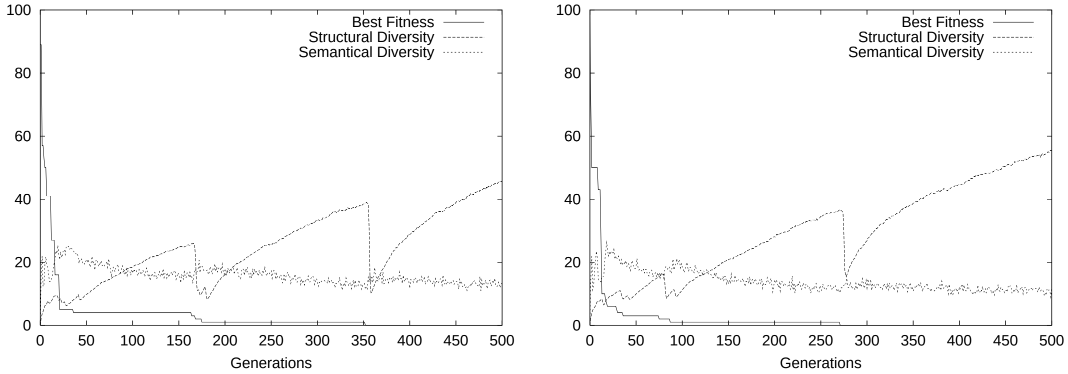

Figure 15: *iris*: Relation between best fitness, structural diversity, and semantic diversity. Two typical example runs.

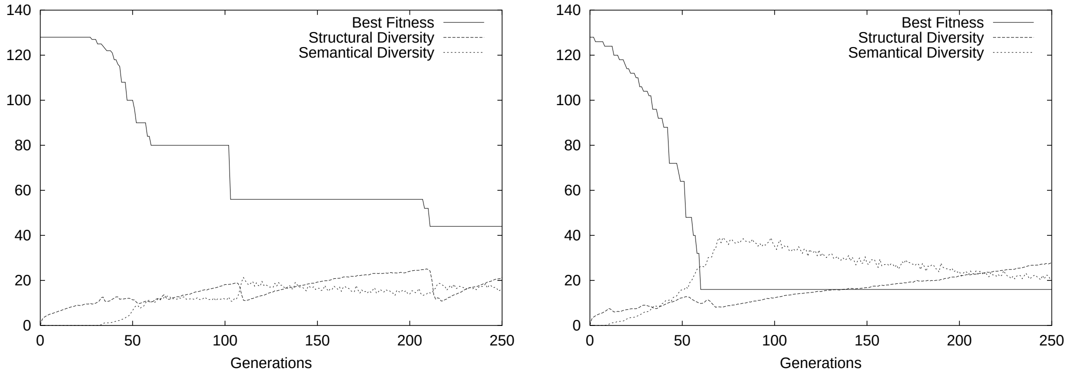

Figure 16: *8-parity*: Relation between best fitness, structural diversity, and semantic diversity. Two typical example runs.

steps. This is why fitness plateaus become less wide.

While structural diversity decreases quickly with the discrete problems when best fitness improves, a sudden increase of semantic diversity, i.e., average fitness distance here, can be observed. This phenomenon may be explained by a fast propagation of the new best fitness value in the population again by what semantically divers individuals are selected more frequently. During a period where best fitness stays constant the average fitness distance decreases again. The wider fitness range of the continuous problem, instead, allows stronger outliers. As a consequence, the average fitness distance developes too irregularly here (not printed).

It is important to note that the increase of structural diversity on fitness plateaus happens implicitly, that is without applying an explicit control of diversity. Using diversity selection increases the structural distance between individuals on fitness plateaus accordingly. Radical drops of diversity, however, as a consequence of sudden accelerations of convergence speed are just as possible as without diversity selection. This shows that an active increase of structural diversity does not slow down the *global* convergence of the best fitness over run. On the contrary, better prediction results have been observed with diversity selection (see Table 3).

8.5 Control of Effective Variation Distance

As motivated in Section 6 we are interested in controlling the *effective* distance between parent and offspring. Even if this is possible with crossover as well as with (macro) mutations, in principle, we restrict to the latter form of variation here. Concerning the absolute distance between program structures crossover step sizes are bigger than mutations step sizes already by definition. This is true for the effective program distance, too, as illustrated at the distribution of variation distances in Figures 5–7. By comparing prediction results obtained with crossover and with effective mutations in Table 3 it becomes obvious that smaller implicit variation steps yield a better quality of solutions. In consideration of those arguments, a reduction of crossover steps by checking the induced degree of structural variation explicitly on the effective program code is rather of minor interest. Instead, we want to find out if solution quality can be further improved by a further reduction of effective mutation distances. Therefore, a program is mutated repeatedly until its distance to the offspring falls below a maximum threshold. Each time a mutation is not accepted its effect on the program is reversed while the choice of the mutation point is free in every iteration.

The applied effective distance metric regards instructions (operators) as smallest distance units (as defined in Section 3) and corresponds to (effective) macro mutations that operate on instruction level. Recall that the effective distance may be altered by register mutations, too. The absolute distance becomes one after an operator mutation only and stays zero otherwise.

In order to guarantee a sufficient growth of programs macro mutations are applied more frequently than micro mutations. Since, in this way, the average step size is not further reducible on the operator side, measuring the distance between *full* effective programs does not promise a higher precision. This is another reason, besides the arguments given in Section 3, why (effective) operator sequences represent a sufficient basis for distance calculation between linear genetic programs.

Table 5 compares average prediction errors for different maximum limits of mutation distance. The maximum possible distance equals the maximum program length (200 instructions) and imposes no restrictions. Setting maximum effective distance to zero is not considered. This would not allow programs to grow sufficiently even if it is possible that inserting an instruction does not change the effective distance.

Variation	Maximum	sinpoly		iris		8-parity	
	Distance	*mean* (*std*)	Δ (%)	*mean* (*std*)	Δ (%)	*mean* (*std*)	Δ (%)
Effective	—	0.46 (0.06)	0	0.90 (0.06)	0	16 (1.2)	0
Mutations	10	0.35 (0.04)	24	0.72 (0.06)	20	13 (1.2)	19
	5	0.33 (0.04)	28	0.74 (0.06)	18	12 (1.2)	25
	2	0.28 (0.03)	39	0.68 (0.05)	24	11 (1.1)	31
	1	0.26 (0.03)	42	0.54 (0.05)	40	9 (0.9)	44

Table 5: Maximum restriction of effective mutation distance. Average error over 200 runs. Statistical standard error in parenthesis. Percental difference from baseline results.

For all three benchmark problems best results are obtained with maximum effective distance one. This is all the more interesting if we consider that a restriction of variation distance always implies a restriction in variation freedom. That is certain modifications might not be executed at certain program positions because too many other instructions would be affected.

In general, measuring effects during multiple repetitions of a variation operator requires a distance metric that is calculated efficiently. Moreover, it has to be guaranteed that the restrictions imposed on the range of variation distances may always be fulfilled by the variation operator for every composition of program. Otherwise, iterations of a variation must be stopped after a maximum number of trials and variation is executed without or with less strong restrictions.

Variation	Maximum	Iterations		
	Distance	sinpoly	iris	8-parity
Effective	—	1.00	1.00	1.00
Mutations	10	1.02	1.02	1.02
	5	1.06	1.05	1.05
	2	1.18	1.12	1.12
	1	1.37	1.18	1.20

Table 6: Average number of iterated variations until the respective maximum distance is met.

As we can see from Table 6 the average number of iterations during run increases only slightly if the maximum threshold is reduced. Not even one and a half iterations are necessary, on average, with the smallest distance. Besides, the maximum number of iterations (10) has hardly ever been exceeded. Both aspects emphasize that freedom of variation is not restricted significantly and that computational costs of this distance control are not expensive.

The results found here further correspond to the distribution of mutation distances in Figures 5–7 where only about 20-30% of all measured step sizes are larger than one. Obviously, larger disruptions of effective code as demonstrated with the example program in Section 6 occur less likely. Effective programs emerge to be quite robust against bigger effective mutation steps, obviously because their survival probability is higher then. In particular, the effectivity of an instruction is depending on more than one succeeding instruction in program.

9 Discussion and Future Research

Basically, development of population diversity over run is depending on the following EA parameters: *population size*, *fitness selection pressure* (*tournament size*), and *reproduction rate*. The distance metrics introduced here for linear GP allow a detailed analysis of such parameter dependencies. In the diversity selection experiments these parameters have been configured with standard settings. Nonetheless, we experienced that diversity selection works with very different configurations, too, including smaller and bigger population sizes. The only adaption that might be necessary is a reconfiguration of selection pressure on diversity. For instance, in significantly bigger populations the influence of diversity selection might require a higher selection pressure in order to reach the same gain in performance. Another interesting question that has to be clarified in this context is whether diversity selection is able to achieve a significant reduction of population size, in general, without leading to worse performance. In order to answer these questions more complex analyses are necessary that fall, however, beyond the scope of this paper.

The two-level selection process may also be used with a selection for minimum *complexity* (on second level). As mentioned in Section 5 the advantages over a weighted complexity term in the fitness function are that fitness selection is less influenced and has a higher priority. Furthermore, a parsimony pressure controlled by a selection probability is more constant over run and does not depend on how both objective terms develop in size.

The separation of linear genetic programs in effective and non-effective instructions offers the possibility for a *selective* complexity selection, i.e., a selection by smallest *effective*, *non-effective*, or *absolute* program length. Imposing a specific pressure on the effective code, for instance, does not influence the growth of non-effective code. In this way, solution size may be punished more specifically while the control function of the intron code for crossover step sizes is retained.

Diversity and complexity selection can be applied in combination, too. Either a third selection level is added or both objectives are combined into a weighted sum which is selected for on second level. In the latter case selection priority of diversity and complexity may be more-or-less the same. In the first case, this may be achieved by using an independent selection probability for each level, that is selection for minimum length may happen on third level even if selection for maximum distance is skipped on second level.

10 Conclusion

An explicit control of population diversity was introduced in form of a two-level selection process. In doing so, we tested different structural and semantic distance metrics for measuring diversity of linear genetic programs. Especially, structural diversity control was found to improve solution performance significantly for three different benchmark problems. In the course of these experiments, we analysed development of diversity over the generations and showed correlations with fitness progress.

Restricting maximum mutation distance on effective code level turned out to be most successful if only a single effective instruction is allowed to change its effectivity status. Thereby the average number of iterated mutations was small that is necessary to keep these conditions. In general, effective variation distances on linear programs were measured much smaller then it might have been expected. Actually, effective program structures emerged to be quite robust against bigger destruction.

Acknowledgements

This research was supported by the Deutsche Forschungsgemeinschaft (DFG), collaborative research center SFB 531, project B2.

References

[1] W. Banzhaf, P. Nordin, R. Keller and F. Francone, *Genetic Programming — An Introduction. On the automatic Evolution of Computer Programs and its Application.* dpunkt/Morgan Kaufmann, Heidelberg/San Francisco, 1998.

[2] C.L. Blake and C.J. Merz, *UCI Repository of Machine Learning Databases* [`http://www.ics.uci.edu/~mlearn/MLRepository.html`]. University of California, Department of Information and Computer Science, 1998.

[3] M. Brameier and W. Banzhaf, *A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining.* IEEE Transactions on Evolutionary Computation, vol. 5(1), pp. 17–26, 2001.

[4] M. Brameier and W. Banzhaf, *Effective Linear Program Induction.* Collaborative Research Center (SFB) 531, Computational Intelligence, Technical Report No. CI-108/01, University of Dortmund, 2001.

[5] P. Dittrich, F. Liljeros, A. Soulier, and W. Banzhaf, *Spontaneous Group Formation in the Seceder Model.* Physical Review Letters, vol. 84, pp. 3205–3208, 2000.

[6] D. Gusfield, *Algorithms on Strings, Trees and Sequences.* Cambridge University Press, 1997.

[7] C. Igel, *Causality of Hierarchical Variable Length Representations.* In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 324–329, IEEE Press, 1998.

[8] C. Igel and K. Chellapilla, *Investigating the Influence of Depth and Degree of Genotypic Change on Fitness in Genetic Programming.* In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1061–1068, MIT Press, Cambridge, 1999.

[9] E.D. de Jong, R.A. Watson, and J.B. Pollack, *Reducing Bloat and Promoting Diversity using Multi-Objective Methods.* In *Proceedings of the Genetic and Evolutionary Computation Conference*, , MIT Press, Cambridge, 2001.

[10] R. Keller and W. Banzhaf, *Explicit Maintenance of Genetic Diversity on Genospaces*, Internal Report, University of Dortmund, 1995.

[11] J.R. Koza, *Genetic Programming.* MIT Press, Cambridge, MA, 1992.

[12] P. Nordin, *A Compiling Genetic Programming System that Directly Manipulates the Machine-Code.* In K.E. Kinnear (ed.) *Advances in Genetic Programming*, 311–331, MIT Press, Cambridge, MA, 1994.

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

[14] U.-M. O'Reilly, *Using a Distance Metric on Genetic Programs to Understand Genetic Operators.* In J.R. Koza (ed.), *Late Breaking Papers at the Genetic Programming '97 Conference*, Standford University, 1997.

[15] J.P. Rosca and D.H. Ballard, *Causality in Genetic Programming.* In L.J. Eshelmann (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 256–263, Morgan Kaufmann, San Francisco, 1995

[16] D. Sankoff and J.B. Kruskal (eds.), *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, 1983.

[17] R. Tanese, *Distributed Genetic Algorithms.* In J.D. Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms*, 434–439, Morgan Kaufmann, San Mateo, CA, 1989.