Dynamic subset selection based on a fitness case topology (preprint)

Christian W.G. Lasarczyk christian.lasarczyk@cs.uni-dortmund.de
Department of Computer Science, University Dortmund,
Joseph–von–Fraunhofer–Str. 20, 44227 Dortmund, Germany

Peter Dittrich dittrich@minet.uni-jena.de
Jena Centre for Bioinformatics and Friedrich–Schiller–University Jena, Department of Mathematics and Computer Science, Bio Systems Analysis Group, 07743 Jena, Germany

Wolfgang Banzhaf banzhaf@cs.mun.ca
Department of Computer Science, Memorial University of Newfoundland,
St. John's, NL, A1B 3X5, Canada

Abstract

A large training set of fitness cases can critically slow down genetic programming, if no appropriate subset selection method is applied. Such a method allows to evaluate an individual on a smaller subset of fitness cases. In this paper we suggest a new subset selection method that takes the problem structure into account, while being problem independent at the same time. In order to achieve this, information about the problem structure is acquired during evolutionary search by creating a topology (relationship) on the set of fitness cases. The topology is induced by individuals of the evolving population, through increasing the strength of the relation between two fitness cases, if an individual of the population is able to solve both of them. Our new topology–based subset selection method chooses a subset, such that fitness cases in this subset are as little as possible related with respect to the induced topology. We compare topology–based selection of fitness cases with dynamic subset selection and stochastic subset sampling on four different problems. On average, runs with topology–based selection show faster progress than the others.

Keywords

Genetic Programming, search space, topology, diversity

1 Introduction

Evolving programs is often a time–consuming task. Usually, the most costly part is to evaluate the fitness of individuals. To evaluate an individuals fitness GP–systems use a set of fitness cases. A fitness case is an input/output pair, fitness measures how well an evolved individual predicts the output(s) from the input(s). In order to reduce the effort, many methods try to reduce the number of fitness cases evaluated during fitness calculation. These methods differ in how they choose proper subsets of the set of all fitness cases for evaluation.

The simplest technique is to use a static subset. *Historical subset selection* (Gathercole and Ross, 1994), for instance, records all fitness cases that are not solved by the best population member in any given generation over a small number of runs. These fitness cases become part of a static subset and are used in further GP runs.

Random subset selection (Gathercole and Ross, 1994) chooses a new subset for each generation. Each fitness case is selected independently with equal probability, which leads to varying subset sizes. *Stochastic sampling* (Nordin and Banzhaf, 1997; Banzhaf et al., 1998) chooses a new subset for each generation and for each individual, respectively, with all fitness cases having the same probability of being selected. In this article we use a third variant that we call *stochastic subset sampling* (SSS). With stochastic subset sampling we choose a new subset each generation with a fixed subset size. These stochastic methods can be used to balance accidentally caused advantages or disadvantages of certain programs given particular fitness cases, in order to prevent a biasing influence of subset selection on evolution.

Dynamic subset selection(DSS) (Gathercole and Ross, 1994; Gathercole and Ross, 1997; Gathercole, 1998) is a procedure based on two assumptions: (i) There is a benefit in focusing GP's abilities on difficult fitness cases, i.e., the ones that are frequently misclassified; and (ii) there is a benefit in checking fitness cases that have not been looked at for several generations. *Dynamic subset selection*, hence, assigns a difficulty D and an age A to every fitness case i and updates these measures in every generation g. Initial difficulty is zero and increases in integer steps each time an individual is not able to solve the corresponding fitness case. The difficulty is reset each time the fitness case is selected for the subset. Age represents the duration since the fitness case was selected for the subset the last time. Initial age is one and is incremented as long as the fitness case is not part of the subset. A reset occurs upon selection. In order to balance between an individual's age and its difficulty, A and D are taken to the power of some parameters a and d, respectively, and summed up to give a weight W for each fitness case i:

$$W_i(g) = A_i(g)^a + D_i(g)^d \quad .$$

For our comparison we set the age exponent a to 3.5 and the difficulty exponent d to 1.0(Gathercole, 1998). A subset with target size S can now be assembled from a total T of fitness cases (training set) by selecting fitness case i with probability

$$P_i(g) = \frac{W_i(g) \cdot S}{\sum_{j=1}^{T} W_j(g)} \quad .$$

Active data selection (Zhang and Cho, 1998) associates training cases with individuals. During evolution these individual subsets are recombined and enlarged by small numbers of fitness cases taken from the base data set. This procedure should ensure diversity of the training data. In the final generation, every individual uses all fitness cases. Zhang and Cho call this process *incremental data inheritance*.

None of the methods mentioned considers the problem's structure in terms of a relation on the fitness cases. In this article we suggest to gather information about the structure of a problem by creating a topology on the set of fitness cases. This relation will be created on the fly, during evolution. The relation between two fitness cases will be strengthened, if an individual of the population is able to solve both fitness cases. The resulting topology reflects the problem structure. The exploitation of informations contained in this topology helps to improve the performance of genetic programming by allowing to select dynamically smaller and more suitable subsets.

This paper is organized as follows: Section 2 presents our method, how we define topology, how we select fitness cases based on this topology, and some more details.

Section 3 then reports our results, partitioned into the different search problems and a comparison with *dynamic subset selection* and *stochastic subset selection*. Section 4 discusses our results in the light of population diversity and contrasts our approach to guided local search and fitness sharing. Section 5 summarizes and gives perspectives.

2 Topology–based Selection

2.1 Motivation

During evolution individuals acquire “knowledge”[1] about how to solve fitness cases. Usually, an individual of the initial population is not able to solve all fitness cases, but can handle only a small subset. If the GP optimization process goes well, individuals become better and better from generation to generation, meaning that the best individual is able to solve more and more fitness cases. But often the population stops improving while the best individual just solves a subset of all fitness cases, and an increasing number of individuals tends to solve the same fitness cases the same way. Evolution settles down into a so called local optimum. So why are these individuals able to solve some but mostly not all fitness cases?

It is well known that structure influences the efficiency of heuristics working on it (Hogg, 1996; Walsh, 1999). Inspired by this, we shall take a relation between the fitness cases into account, a relation detected during evolution, coded into an individual’s genotype, and spread through the population by recombination or other conservative operations.

If one or a group of individuals is able to solve a subset of all fitness cases better than the rest of the fitness cases, then we suppose that these individuals contain some kind of knowledge about the relationship between the fitness cases solved. Fitness cases can be neighbors in a space we call *similarity space* (according to Goldberg and Richardson (1987)), a space formed by the knowledge of all individuals.

An individual’s knowledge can spread through the population, if it leads to a higher fitness of the individual (Holland, 1975; Koza, 1992b). On the other hand, the lower the fitness gain is, the slower this knowledge spreads.

2.2 Topology Definition

During evolution the population induces a structure on the set of fitness cases V. We represent the topology of this structure by an undirected weighted graph $G = (V, E)$, where E is the set of all possible edges between the fitness cases V. A weight is associated with each edge. The weight represents the information gathered by individuals during evolution on how closely the fitness cases are related. High values mean a close relation. At the beginning, all weights are initialized to zero.

An individual can strengthen the relation between two fitness cases. If it is able to solve both of them, the weight on the edge between them is increased. Therefore, a binary rating of an individual’s ability to solve a fitness case is required. For a continuous regression problem like the sine approximation in Sec. 3.2, a threshold value is used to decide whether the individual is able to solve the fitness case or not. For a classification problem, correct classification means that this fitness case is solved.

We suggest that these relations are not of the same importance, because there must be a reason, why some of the individuals of the initial population solve many fitness cases while others solve only two or three. Hence the weight of a detected relation will

[1]Knowledge could be anything leading to the individual’s result, such as blocks of code, automatically defined functions(ADF) etc.

depend on the number of fitness cases solved by an individual.

At the end of every generation, we adapt the edge weights between fitness cases for each individual of the population. Formally, after each generation we perform the following two steps:

Step 1: For each individual, let $V' \subset V$ be the set of fitness cases solved by the individual, E' the set of all edges between nodes from V', and w_e the weight of edge e. If $|V'| > 1$ each edge weight $w_e, e \in E'$ is adapted by

$$w_e := w_e + \frac{2}{|V'|(|V'|-1)}, \quad e \in E' \ . \tag{1}$$

Step 2: To reduce the impact of relations detected in the past each edge $e \in E$ is multiplied by the *loss rate* $\lambda < 1$:

$$w_e := \lambda w_e, \quad e \in E \ . \tag{2}$$

Test runs showed that λ can be chosen between 0.5 and 0.9. For the experiments reported here, $\lambda = 0.7$ has turned out to be a good value.

An edge value between two fitness cases above average indicates a similarity between these two fitness cases. Thus, there is a higher probability that a randomly chosen individual is able to solve both fitness cases instead of just one of them.

2.3 Subset Selection — Algorithm

Our target is to evolve an individual able to solve as many fitness cases as possible. Emerging clusters of heavily related fitness cases show that different knowledge is accumulated within the population. Weak connections between clusters are evidence that knowledge is missing on how to solve fitness cases from different clusters. Our new method should direct the attention of evolution towards this kind of knowledge by weighting clusters equally. To do so, one fitness case is selected from each cluster for the subset. In order to outcompete others, an individual has to acquire knowledge that spans clusters.

To encourage this type of progress all fitness cases connected with an edge weight higher than an adaptive threshold value are excluded from selection into the same subset. In the next section (Sec. 2.4) we will describe how to adapt this threshold value.

Fitness cases for the subset are selected according to the following algorithm at the beginning of each generation:

1. *Empty* the set of selected fitness cases and the set of excluded fitness cases.

2. *Select* randomly a fitness case into the subset from the set of all fitness cases not yet selected or excluded.

3. *Exclude* all fitness cases connected with the selected one, provided the edge value exceeds the threshold.

4. *Goto Step 2, until* no fitness case is left or the desired subset size is reached.

Figure 1 illustrates the two steps of the algorithm for subset selection.

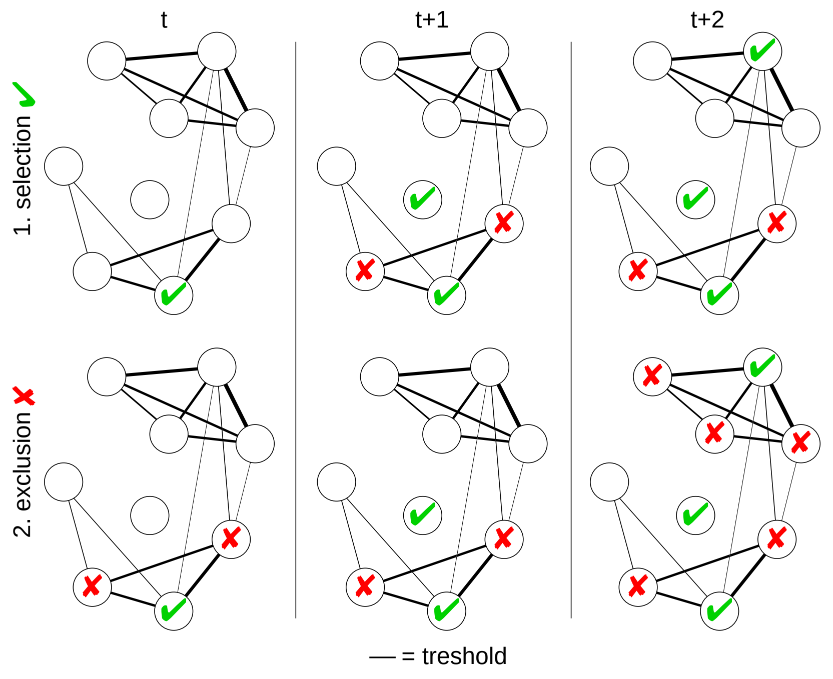


Figure 1: The figure illustrates the selection of three fitness cases into the subset. Each node in the graph represents one fitness case. Thickness of the lines represents the weight of edges. Selection and exclusion take turns alternating. After random selection of a fitness case into the subset, topology–based selection excludes every fitness case that is connected with the selected fitness case via an edge weight higher than a threshold or has already been selected before. The random choice is restricted to the remaining fitness cases.

2.4 Adapting the Threshold Value

The selection of one fitness case to be included into the subset leads to the exclusion of all fitness cases that are connected to it stronger than an adaptive threshold. Thus, the threshold value plays an important role. A badly chosen value could lead to the following problems: If the threshold is *too high*, one does not exclude enough fitness cases from selection into the subset. In the extreme case, the algorithm would not exclude any fitness case and would always select randomly. In this case *topology selection* does not differ from *stochastic subset sampling*. If the threshold is *too low*, too many fitness cases are excluded, and the desired subset size will not be reached. The resulting subset size would lead to an overrated influence of single fitness cases.

Choosing the threshold is a dynamic task, because edge weights vary during evolution. For this reason we determine the threshold value based on the distribution of all weights. Therefore we sort the list of all weights and adjust an index position j we take the threshold τ from. Given a desired size m of the subset, the set of all fitness cases V (training set), and the number of fitness cases $n = |V|$, the threshold τ is adapted by the following binary–search–type of algorithm (see also Fig. 2):

1. Sort all edge weights in ascending order.

The weight of non–existing edges is zero.
Let τ_i be the i–th weight in the sorted list.

2. Initialize the step size σ (we use the number of fitness cases here, $\sigma := n$). Set the current index j to the value of the last selection (0 at the outset).
3. Repeat s_{max} times or until the desired subset size is achieved:
   (a) Set the threshold to the edge weight at the current index position: $\tau = \tau_j$.
   (b) Select a subset according to the algorithm explained previously (Sec. 2.3). Let m' be the size of that subset.
   (c) If the selected subset is too small ($m' < m$), do:
      i. If the selected subset had been previously too large, half the current step size: $\sigma := \sigma/2$.
      (This means, if we change the direction of adaptation, we decrease the step size in oder to approach the optimum more carefully.)
      ii. Increase j by the current step size: $j := j + \sigma$.
   (d) If the selected subset is too large ($m' > m$), do:
      i. If the selected subset had been previously too small, half the current step size: $\sigma := \sigma/2$.
      ii. Decrease j by the current step size: $j := j - \sigma$.

For each subset selection the algorithm can take up to $s_{max} = 30$ attempts to select a good subset, but on average requires less then five to adapt τ. For an analysis of the adaptation behavior see Lasarczyk (2002). Alternatively we could start a binary search in the middle of the list of sorted edge values, but this would take more adaptation steps.

As can be seen, topology–based subset selection requires additional computing time for adapting the topology, adjusting the threshold value and selecting the subset. The most expensive task is the adaptation of the topology and sorting of the edge values, which scales approximately quadratically with training set size[2]. Therefore we recommend this method for problems where the evaluation of a fitness case is costly, such as evolving control programs for robots (Miglino and Walker, 2002).

2.5 Example of the Induced Structure and its Time Evolution

Figure 3 shows how the structure of fitness case relations changes during evolution of a sample problem. Only edges with a weight exceeding the threshold value are drawn. In the first generation just a small number of fitness cases have been part of the first subset, therefore the population detected only relations between those few fitness cases. After $t = 150$ generations all fitness cases have been part of a subset at least once. It is difficult to visually detect clusters during this stage of evolution. Later ($t = 300$) clusters become more visible. Although the optimal solution is found in this example, it seems that many individuals are trapped in suboptima and induce large clusters. Maybe the "knowledge" required to solve the fitness cases of these large clusters can be easily protected against destruction.

Figure 4 shows graphs for the same problem. Here, only those edges are drawn that connect a selected fitness case and those excluded through it. Each fitness case in

[2]If V is the training set, the time complexity of sorting the $|V|^2$ edge values is bounded by $O(|V|^2 \log |V|^2)$.

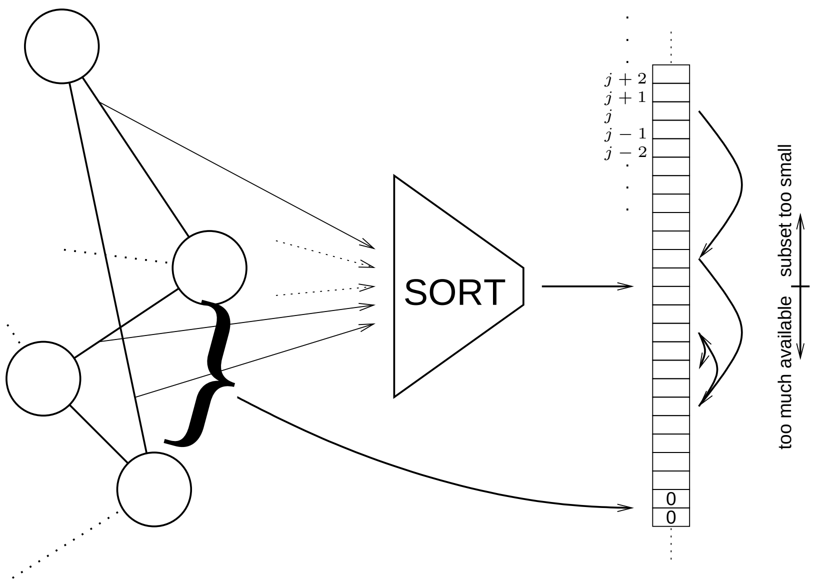


Figure 2: Adaptation of the threshold is performed with an index on a sorted list of all edge values. Topology–based selection has more than one attempt to select a subset, because the size of the selected set could vary heavily. Starting with index j adopted from the previous subset selection, the index is decreased if selection is not strong enough and the index is increased if subset gets too small.

the subset represents a different cluster. While the selected fitness case is connected to every other in this cluster by an edge with an edge weight above the threshold, the nodes of a cluster are not necessarily fully connected. Note that the first fitness case selected will cause the largest expected exclusion. And the last selected fitness case will only exclude a small number of fitness cases, because there are not many left to be excluded. This also explains the different cluster sizes visible in Fig. 4.

3 Results

To show the advantages of the new subset selection method we compare it with *dynamic subset selection* and *stochastic subset sampling* on four different problems, namely, two approximation and two classification problems. Despite of the problem domain, our experimental settings differ in population size, number of available fitness cases, and subset size. Tables 2 and 3 summarize the settings for the GP–system described next.

3.1 GP–System and Settings

We use a simple linear genetic programming system (Banzhaf et al., 1998). Each instruction consists of four integers coding for an operation, two source registers, and one target register, respectively. The target register could be one out of five registers initialized with zero. The source register could be one of these registers, registers containing the problem specific input, or registers containing constant values (0, 1, 2, 5, 10, 20, 50, 100, π and e). The operator set consists of arithmetic operators, trigonometric operators, logic operators, and a conditional assignment. The tables detail which oper-

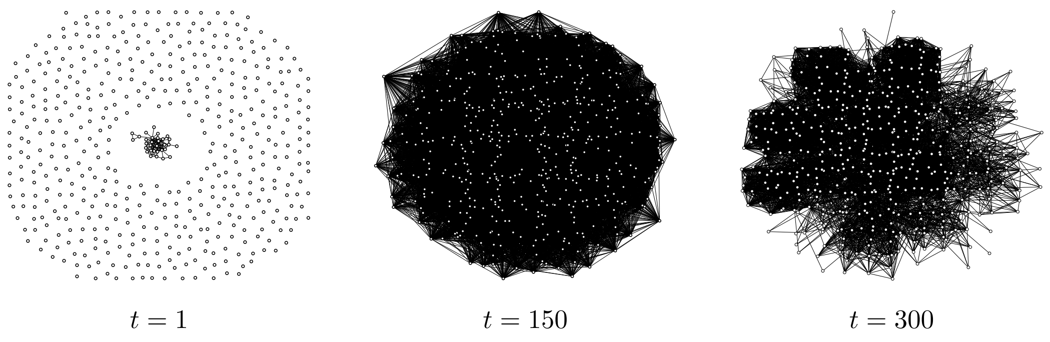


Figure 3: Evolution of the fitness case connections for a simple test problem. Each fitness case is represented by a dot, connections are drawn when the edge value is above the threshold. The topology is valid for the whole population of one generation.

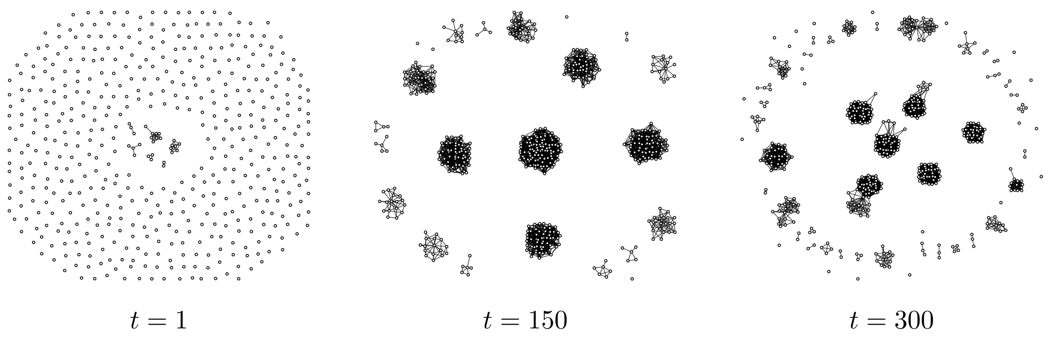


Figure 4: Same graphs as above (Fig. 3), with drawing edges between a fitness case and those that are excluded form selection if this fitness case is selected.

ations are provided for which problem. We allow a maximum of 256 instructions per individual, thus length of an individual is variable but bounded. The return value of an individual is the result of the last operation, independent from that operation's target register.

With *dynamic subset selection* and *topology–based selection* the fitness of the best individual varies heavily on the current subset used to evaluate the individuals and the testing set. The reason for this behavior is that these subset selection methods concentrate sometimes on subsets, that do not represent a uniformly distributed subset of all fitness cases. For this reason we determine the best individual by using a third set, the so called *validation set*, then we calculate the best validating individual's fitness on the test set. The "mean" of the this value (as listed in the following tables; Tab. 4, 5, 6, and 7) is computed as the average over 100 independent runs. Validation set and test set are independent from the training set V. They are never used for selection and do not influence the evolutionary process. They are solely for the purpose of monitoring.

As search operators we use mutation as well as crossover. A single mutation operates by replacing an instruction by a randomly created new instruction. The crossover operator is a simple one–point crossover. If an offspring exceeds the allowed length of 256 instructions, we cut off instructions at its front. We do so to underline the conservative aspect of crossover. Individuals return the result of their last operation and so

	Setting 1	**Setting 2**
Crossover	60%	0.1%
Reproduction	40%	99.9%
Mutation probability	90%	99.9%
Number of mutations	8	8

Table 1: Settings for evolution parameters

their tail is the more important half.

In order to demonstrate that the improvement by our new subset selection method does not stem from a specific setting of the search operators, we use two quite different settings (see table 1 for direct comparison):

Setting 1: With this setting 60% of the offspring are produced by crossover. For crossover we perform tournaments of size four and recombine the two best individuals[3] of each tournament in order to get two offspring, which replace the two losers of the tournament. 40% offspring are created by tournaments of size 2, where the looser is replaced by a copy of the winner. Each offspring is mutated with a probability of 90% ; in that case eight instructions (lines of the linear program) are replaced by random instructions.

Setting 2: Crossover rate is reduced significantly. Just 0.1% of all offspring are created by crossover. 99.9% of the offspring are created by reproduction. All offspring are mutated with probability 99.9% .

3.2 Sine Function Approximation

Approximation of the sine function with non–trigonometric functions is a non–trivial but illustrative problem. The set of fitness cases $V = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ is created in the following way: In the interval $[-\pi, \pi]$ $n = 400$ equidistant values x_i are used to calculate values $y_i = sin(x_i)$, $i \in \{1, 2, \dots, n\}$.

Given a subset V' of the training set V, the fitness function is the mean squared error of the individual I applied to all fitness cases of the subset:

$$f(I) = \left(\sum_{(x,y)\in V'} (I(x) - y)^2 \right) \Bigg/ |V'| \quad .$$

(x, y) denotes a fitness case in the subset V' of size $|V'|$, x the input and y the desired output.

As mentioned above, we have to define a function that specifies when a fitness case is solved by an individual. We define a fitness case as *solved* by an individual if the error $(I(x) - y)$ is less than 0.1. Hence we increase an edge value between two fitness cases, if an individual's error is less than 0.1 on both of them.

Table 4 summarizes the performance for all three subset selection methods. For both settings, evolution benefits from topology–based subset selection. This is most obvious for Setting 2 (mainly mutation). We can also see that all methods benefit from Setting 2.

[3]For clarity note that the individuals are evaluated on the subset selected from the training set V.

Problem ID	**sin**	**F6 (mod.)**
Problem type	Approximation	Approximation
Problem function	$\sin(x)$	$0.5 + \frac{sin^2\sqrt{\lvert x+y\rvert}-0.5}{[1.0+0.001\cdot(x^2+y^2)]^2}$
Number of inputs	1 real value	2 real values
Fitness function	mean square error	correlation
Training set size	400	3000
Subset size	25	400
Ratio	0.0625	0.1333
Validation set size	400	1000
Testing set size	400	1000
Generations	1000	1000
Size of population	2500	1500
Instruction set	$+, -, \cdot, /, \vee, \wedge,$ `if`, $=, <$	$+, -, \cdot, /, \vee, \wedge,$ `if`, $=, <, \sin(x), \cos(x), \tan(x), \sqrt{x}, x^2$

Table 2: Problem–specific parameter settings for the two approximation problems.

3.3 Modified F6–Function

As a second approximation problem we modify the $F6$–Function taken by Schaffer et al. (1989) [4]

$$F6_{\text{mod.}}(x_1, x_2) = 0.5 + \frac{\sin^2\sqrt{|x_1 + x_2|} - 0.5}{[1.0 + 0.001 \cdot (x_1^2 + x_2^2)]^2} \quad . \tag{3}$$

5000 points $\vec{x} = (x_1, x_2)$ are created with uniform random distribution within a distance of 100 from the origin. Corresponding values $y = F6_{\text{mod.}}(\vec{x})$ are determined. 1000 points are set apart as validation set and 1000 as the testing set. The remaining 3000 fitness cases form the training set V, we have to choose the subsets from.

As in Keijzer (2001) , we use the square of the correlation coefficient r (stability index) to rate an individual's fitness. One can think of the stability index as a value that shows how much of the variation of one value could be described via a linear transformation by the variation of a second value. In this case one value is the desired output y, the other value is the individuals output $I(\vec{x})$. Let $(\vec{x}_i, y_i)$ be the i–th fitness case. $\vec{x}_i$ is the input and y_i the desired output of an individual I. Given a set of n fitness cases, the correlation coefficient is defined as:

$$r(I) = \frac{\sum_{i=1}^{n}\left(I(\vec{x}_i)\cdot y_i\right) - \sum_{i=1}^{n} I(\vec{x}_i)\cdot\sum_{i=1}^{n} y_i/n}{\sqrt{\left(\sum_{i=1}^{n} I(\vec{x}_i)^2 - \left(\sum_{i=1}^{n} I(\vec{x}_i)\right)^2/n\right)\cdot\left(\sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2/n\right)}} \quad . \tag{4}$$

[4]Schaffer et al. (1989) used the $F6$–Function as an objective function (fitness function) in order to evaluate the performance of genetic algorithms. Our modification as well as using the stability index to rate the individual fitness have been necessary to succeed in this approximation problem.

Problem ID	**spiral**	**thyroid**
Problem type	Classification	Classification
Problem function	2 intertwined spirals	real world data
Number of inputs	2 real values	21 (6 real values, 15 binary values)
Fitness function	classification error	
Training set size	194	3772
Subset size	20	200
Ratio	0.103	0.053
Validation set size	192	1000
Testing set size	192	2428
Generations	2000	1000
Size of population	2000	2000
Instruction set	$+$, $-$, $\cdot$, $/$, $\vee$, $\wedge$, `if`, $=$, $<$, $\sin(x)$, $\cos(x)$, $\tan(x)$	

Table 3: Problem–specific parameter settings for the two classification problems considered.

The fitness of individual I is defined by:

$$f(I) = 1 - r(I)^2 \quad .$$

We define a fitness case as *solved* by an individual, if the Euclidean distance between the desired and the individuals output is less than 0.05. As before, the algorithm increases the edge weight between two fitness cases, if one individual produces a smaller error for both of them.

Figure 5 shows a plot of the modified F6–function on the left side. On the right side the input–output function of the best individual found by evolution is plotted. We can see significant deviations at the corners of the functions domain, because they have a distance greater than 100 and so there are no fitness cases that would punish a deviation in that region.

Table 5 shows the average fitness and the confidence interval for the paired differences between the best fitness values. The results are based on 100 runs for each of the three different selection methods. Runs using topology–based selection show the best results on average for both settings.

3.4 Intertwined spirals problem

To solve the intertwined spirals problem, a solution has to classify points belonging to one of two spirals in the x–y–plane. The spirals are intertwined and described by 97 points each. This problem has been already examined by means of neural networks (Lang and Witbrock, 1989) and genetic programming (Koza, 1992a).

Instead of using the 194 example sized dataset, however we created an additional set including 192 examples. For this validation set we choose points next to the original points in the training set. We use this set to monitor the individuals during the evolution and we use the whole original dataset to test the individuals. Figure 6 displays the fitness cases from the original dataset. To make the task more difficult, we use a subset

Time (Gen.)	**mean** TBS	DSS	SSS	**95% confidence interval** TBS-DSS	TBS-SSS	DSS-SSS
				Setting 1		
250	0.71	0.79	0.79	[-0.128, -0.033]	[-0.136,-0.034]	[-0.052,+0.043]
500	0.59	0.67	0.71	[-0.133, -0.034]	[-0.166,-0.068]	[-0.082,+0.016]
750	0.54	0.62	0.65	[-0.130, -0.031]	[-0.157,-0.057]	[-0.079,+0.025]
1000	0.49	0.58	0.60	[-0.136, -0.035]	[-0.151,-0.052]	[-0.067,+0.035]
				Setting 2		
250	0.62	0.69	0.71	[-0.152,+0.017]	[-0.165,-0.024]	[-0.106,+0.053]
500	0.49	0.58	0.61	[-0.184,+0.002]	[-0.203,-0.044]	[-0.118,+0.054]
750	0.40	0.50	0.54	[-0.186, -0.006]	[-0.218,-0.050]	[-0.122,+0.047]
1000	0.34	0.43	0.47	[-0.178, -0.010]	[-0.212,-0.049]	[-0.117,+0.044]

Table 4: Mean squared error for sine function approximation. For different time steps we take the best individual of the validation set and evaluate its fitness on the test set. For every subset selection method we show its mean fitness averaged of 100 runs on the left and the 95% confidence interval for the paired differences between the selection methods on the right.

size of $n' = 20$ fitness cases. Because each fitness case can be classified either correctly or not, this leads to 20 different fitness levels.

Table 6 summarizes the average results. For the first setting, runs using the topology–based selection show the best results on average. All subset selection techniques profit from the second setting. If one compares the results of both settings one can see a premature convergence into a local optimum with the second setting. It is assumed that the loss of diversity in conjunction with convergence leads to less information about the relation between fitness cases. As a result similar individuals would perceive similar relations between fitness cases and there is not much to be gained from topology.

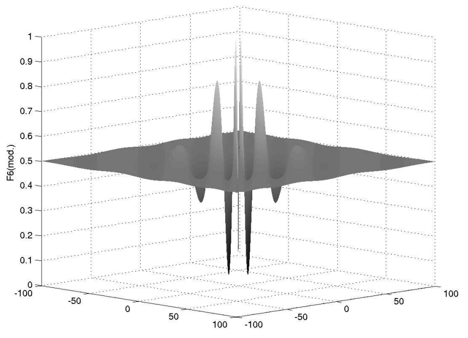
3.5 Thyroid–Problem

The thyroid–problem is a real world problem. The individual's task is to classify humans thyroid function. The dataset was obtained from the UCI–repository (Blake and Merz, 1998). It contains 3772 training and 3428 testing samples, each measured from one patient. A fitness case consists of a measurement vector containing 15 binary and 6 real valued entries of one human being and the appropriate thyroid function (class).

There are three different classes for the function of the thyroid gland, named *hyper function*, *hypo function* and *normal function*. As Gathercole (1998) already showed, two out of these three classes, the hyper function and the hypo function, are linearly separable. Given the measurement vector as input, an individual of the GP system should decide whether the thyroid gland is normal functioning, or should be characterized as hyper or hypo function.

Because more than 92% of all patients contained in the dataset have a normal function, the classification error must be significantly lower than 8%. The classification error is the percentage of misclassified individuals.

The selection algorithms picks its subsets out of the 3772 training examples. From



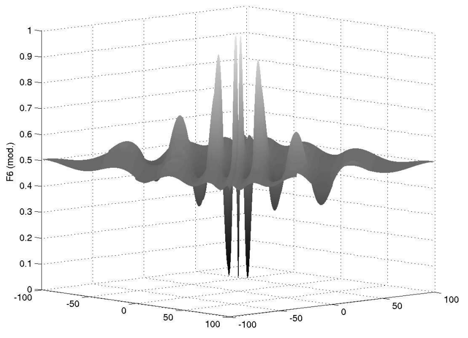


Figure 5: Left: A plot of the modified F6–function. Right: The approximation of the best individual.

the set of testing examples we remove the first 1000 examples to form a validation set. The remaining examples form the testing set.

As Gathercole did, we assign the following meaning to the output of the individuals. A positive output (≥ 0) denotes normal function, otherwise hyper or hypo function.

Table 7 shows that topology–based selection acquires the best average results for the first setting. For the second setting there is no statistic significant difference between the three selection methods.

4 Discussion

In the previous section we have empirically compared topology–based subset selection to other techniques for subset selection. We found evidence that evolution can be significantly faster for both settings. Now we shall take a closer look at what causes the population to evolve faster.

One of the main targets of good GP runs is to prevent the population from premature convergence. Thus, a closer look at the population's diversity (Sec. 4.1) is warranted. In Sec. 4.2 and Sec. 4.3 we compare topology–based selection with two other methods (*fitness sharing* and *guided local search*), which examine many peaks without focusing the entire population on one peak alone.

4.1 Population Diversity

In order to demonstrate what happens to the population when using topology–based selection, diversity is measured as the average normalized edit distance of the effective code between individuals of the population. For the purpose of this discussion, we chose the optimization problem that has led to the largest difference between selection methods, namely the regression problem of the modified F6–function with parameter setting 2(see Tab. 5).

Topology–based selection is expected to increase diversity. High diversity alone, however, cannot explain good performance[5]. For an explanation we take the concept of "appropriate diversity" (see Goldberg and Richardson (1987)). Usually, a population

[5]E.g., increasing the mutation rate leads to higher diversity, but not necessarily to higher performance.

Time	**mean**			**95% confidence interval**		
(Gen.)	TBS	DSS	SSS	TBS-DSS	TBS-SSS	DSS-SSS
Setting 1						
250	0.82	0.89	0.93	[-0.122,-0.030]	[-0.151,-0.071]	[-0.076,+0.005]
500	0.74	0.84	0.89	[-0.151,-0.043]	[-0.199,-0.103]	[-0.103, -0.005]
750	0.69	0.81	0.87	[-0.171,-0.059]	[-0.234,-0.125]	[-0.118, -0.011]
1000	0.66	0.77	0.85	[-0.164,-0.044]	[-0.249,-0.133]	[-0.143, -0.031]
Setting 2						
250	0.83	0.91	0.94	[-0.125,-0.036]	[-0.151,-0.069]	[-0.068,+0.008]
500	0.75	0.86	0.91	[-0.167,-0.052]	[-0.213,-0.105]	[-0.099, -0.001]
750	0.67	0.83	0.89	[-0.222,-0.092]	[-0.276,-0.161]	[-0.114, -0.010]
1000	0.60	0.80	0.87	[-0.277,-0.125]	[-0.338,-0.208]	[-0.130, -0.014]

Table 5: Mean fitness (on the test set of the best individual on the validation set) for approximation of the modified F6–Function (small values denote good performance) at different time steps is shown on the left side. The fitness value is inversely proportional to the square of the correlation coefficient r. On the right side you can see the confidence interval for the paired differences between the subset selection methods. The results are based on 100 runs for each method.

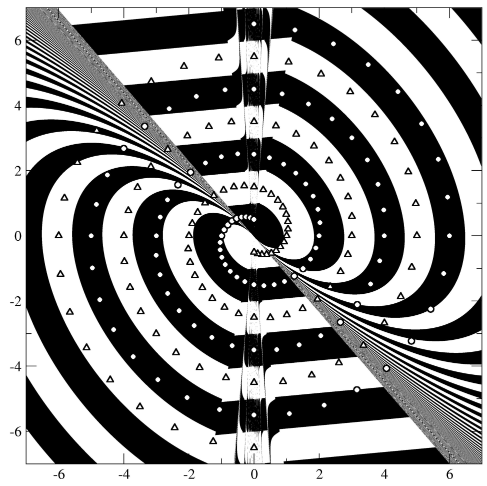
shows its greatest diversity after initialization. While this is a good point to start from, it is a bad point to stay, because the fitness of many individuals is low. Good individuals are the important base for evolution's success. Just looking at the pure average fitness of the population does not allow to predict the ability for further improvements. If all individuals are the same, average fitness may be good but improvements become less likely. Thus, good performance needs both, a good average fitness combined with a high diversity, which we call *appropriate diversity*

In order to measure diversity, we take 300 individuals randomly from the population, remove all instructions from the genome that are not used and calculate the distance between the remaining (effective) code of two individuals by the Levenshtein (1966) method(edit distance). Then the distance is normalized by dividing through the average number of instructions of effective code.

To compare the relation of diversity to average fitness for the different subset selection methods, we partition the space of 'average edit distance'–'average fitness'–pairs into squares (bins) of size 0.05×0.05. Then we count for each bin (and for all 100 runs) how frequently the population is in the respective state, namely possessing the respective average edit distance and average fitness. Figure 7 shows the histograms and the contour lines of equipotential surfaces for different frequencies. One can see that topology–based selection leads to populations with a good average fitness (here, equal to a low fitness value) and a high edit distance.

4.2 Fitness Sharing

Fitness sharing, proposed by Goldberg and Richardson (1987), is a frequently used technique in the field of genetic algorithms. Fitness sharing should avoid "premature convergence [. . .] before obtaining sufficiently near–optimal points". The authors point out that maintaining diversity for its own sake is not the issue, instead the aim of fitness



```
temp[3] = (1 < 100 );
temp[0] = cos(5);
temp[4] = temp[3] + temp[0];
temp[1] = input[0] * M_PI;
temp[1] = temp[1] * temp[4];
if (temp[1] != 0)
 temp[3] = input[1] / temp[1];
 else temp[3] = 0;
temp[1] = 5 + temp[3];
temp[3] = tan(temp[1]);
if (temp[3] > 0)  temp[1] = temp[0];
temp[1] = tan(temp[1]);
temp[2] = temp[1] * input[0];
temp[4] = M_E - 20;
temp[4] = temp[2] + temp[4];
temp[1] = input[1] * M_PI;
temp[4] = temp[4] - temp[1];
temp[1] = cos(temp[4]);

if (temp[1] >= 0) result=1; //BLACK
  else result=0;            //WHITE
```

Figure 6: Left: The white and black areas show the classification of the best individual (error approx. 7%). The triangular and round symbols represent the two classes of the intertwined spiral problem introduced by Lang and Witbrock (1989). Right: C code of the best individual used for the classification on the left side.

sharing is to achieve "appropriate diversity". Here, "appropriate" means that the size of a subpopulation exploring a fitness peak is adequate to the peak's fitness.

In fitness sharing, the fitness in a region of the search space becomes a resource shared by individuals in that region that form an own species. This leads to an additional source for competition. To decide whether or not individuals are of the same species a similarity measure, the sharing function, is introduced. Individuals whose distance is less than a threshold are defined to belong to the same *species*, and thus share their fitness values. This is achieved by dividing the original fitness by the so called *niche count*, a value proportional to the number of individuals an individual shares its fitness with.

While fitness sharing directly depreciates areas of the similarity space to valorize other areas, this happens indirectly with our topology–based subset selection. Here we depreciate areas in the set of fitness cases, which are used to evaluate the fitness. To do so, we first have to correlate fitness cases. This happens during evolution, where we build up and adapt the similarity space — a space describing the similarity of fitness cases from the current and past populations point of view.

In order to explain the effect of increasing diversity, assume that the subset used to evaluate the individuals contains only two *similar* fitness cases. In that case, a mutation that leads to an adaption to one fitness case will also likely lead to an adaption to the other fitness case, thus gaining a large amount of fitness. In a second scenario, assume that the subset contains two *different* fitness cases. Now it is harder to achieve a large fitness gain by a "simple" mutation, because it is more unlikely that an adaptation to one fitness case will lead also to an adaption to the other. On the other hand, in the second scenario, *two* different adaptation strategies can lead to a fitness gain. So, the space of viable offsprings that are fitter than their parents is larger than in the first scenario, which may explain the higher observed diversity when our topology–based

Time (Gen.)	**mean** TBS	DSS	SSS	**95% confidence interval** TBS-DSS	TBS-SSS	DSS-SSS
Setting 1						
500	0.38	0.40	0.42	[-0.040, -0.005]	[-0.056, -0.024]	[-0.033, -0.002]
1000	0.36	0.40	0.40	[-0.054, -0.017]	[-0.057, -0.019]	[-0.019,+0.014]
1500	0.35	0.40	0.40	[-0.059, -0.023]	[-0.065, -0.024]	[-0.020,+0.013]
2000	0.34	0.38	0.39	[-0.066, -0.030]	[-0.069, -0.037]	[-0.022,+0.012]
Setting 2						
500	0.31	0.32	0.33	[-0.024,+0.008]	[-0.032, -0.002]	[-0.025,+0.007]
1000	0.30	0.30	0.31	[-0.019,+0.015]	[-0.029,+0.005]	[-0.028,+0.008]
1500	0.29	0.29	0.30	[-0.021,+0.020]	[-0.026,+0.009]	[-0.028,+0.010]
2000	0.28	0.29	0.29	[-0.029,+0.015]	[-0.028,+0.010]	[-0.021,+0.017]

Table 6: Mean classification error on the test set for the intertwined spiral problem. For different time steps we take the best individual of the validation set and evaluate their fitness on the testing set. For every subset selection method you can see its mean fitness averaged of 100 runs on the left and the 95% confidence interval for the paired differences between the selection methods on the right.

subset selection is applied.

4.3 Guided Local Search

Guided local search (GLS) proposed by Voudouris and Tsang (1996) has been applied to a wide range of combinatorial optimization problems (see Voudouris (1998) for a list of references). GLS is a heuristics to guide search in vast search spaces by changing the objective function dynamically during evolution. This happens by adapting a set of additional penalty terms between two iterative search steps. A penalty term refers to a solution feature and a part of its domain. When search stagnates in a local optimum, the penalty terms of one or more features exhibited by solutions within this local optimum are incremented. Search continues with the modified objective function. As a result, search will avoid regions covered by these penalty terms and focus on more promising regions of the search space.

In contrast to fitness sharing and topology–based subset selection, GLS does not just depreciate detected local optima, but entire subspaces of solution features. While GLS adds penalty terms for one or more features of the local optimum and so depreciates all solutions that include those features, fitness sharing directly depreciates solutions that are close to other solutions within a distance in similarity space smaller than σ_{share}. As mentioned above our technique does not try to influence the solution space directly. Instead, topology–based selection tries to provide a global problem view to evolving individuals. Fitness cases solved by one or more individuals are therefore not prevented from getting into the subset. Each "known" region of the fitness case space is allowed to be represented by at least one fitness case. Excluded from the subset are only those cases that are often solved by the same individuals and already represented by one fitness case.

A further difference to GLS is that GLS does not start its counter measures until search gets caught in a local optimum. In contrast, our method continuously tries to

Time (Gen.)	**mean** ($\times 10$) TBS	DSS	SSS	**95% confidence interval** ($\times 10$) TBS-DSS	TBS-SSS	DSS-SSS
			Setting 1			
250	0.51	0.53	0.51	[-0.037,+0.004]	[-0.026,+0.024]	[-0.007,+0.039]
500	0.48	0.50	0.50	[-0.051,+0.003]	[-0.052,+0.006]	[-0.025,+0.026]
750	0.45	0.49	0.49	[-0.070, -0.010]	[-0.069, -0.009]	[-0.025,+0.027]
1000	0.43	0.47	0.48	[-0.070, -0.007]	[-0.072, -0.015]	[-0.035,+0.024]
			Setting 2			
250	0.50	0.49	0.50	[-0.019,+0.042]	[-0.023,+0.025]	[-0.041,+0.019]
500	0.46	0.46	0.47	[-0.041,+0.026]	[-0.047,+0.018]	[-0.039,+0.025]
750	0.43	0.44	0.45	[-0.054,+0.019]	[-0.064,+0.008]	[-0.042,+0.021]
1000	0.41	0.43	0.44	[-0.059,+0.018]	[-0.059,+0.016]	[-0.036,+0.034]

Table 7: Mean classification error on the test set for the thyroid problem (small values denote good performance). Because 92% of all fitness cases belong to one class, even the fitness of bad individuals is lower than 0.08. For different time steps we take the best individual of the validation set and evaluate their fitness on the testing set. For every subset selection method you can see its mean fitness averaged of 100 runs on the left and the 95% confidence interval for the paired differences between the selection methods on the right. Note that the fitness values are scaled by a factor of 10 for easy comparison.

prevent solutions from settling down in local optima during the search process.

5 Conclusion and Outlook

We have shown that topology–based selection of a subset accelerates evolutionary search. We investigated this behavior in genetic programming for four problems from two different problem domains, such as function approximation and classification. In order to ensure that the observed success does not depend on the chosen evolution settings, we performed experiments with two quite different parameter settings. We found that the topology on the fitness cases, which is induced by the population of individuals contains useful information, which can be exploited to control diversity and by that improve the performance of genetic programming. We defined appropriate diversity as a combination of high diversity and good average fitness. We showed exemplarily that runs using a topology–based selection exhibit such appropriate diversity.

Wagner and Altenberg (1996) describe evolvability (see Altenberg 1994) as a genome's ability to produce adaptive variants. In their opinion "adaptations are possible if improvement can be achieved in a cumulative or stepwise fashion". As a structural key feature they consider that "improvements in one part of the system must not compromise past achievements". Topology–based selection supports this requirement by not overvaluing achievements. Overvaluing would lead to an increasing selection of single achievements, penalizing others. By preventing this we hope to advance a higher degree of variability. Investigations on the influence of subset selection on evolvability are an interesting aspect for further research.

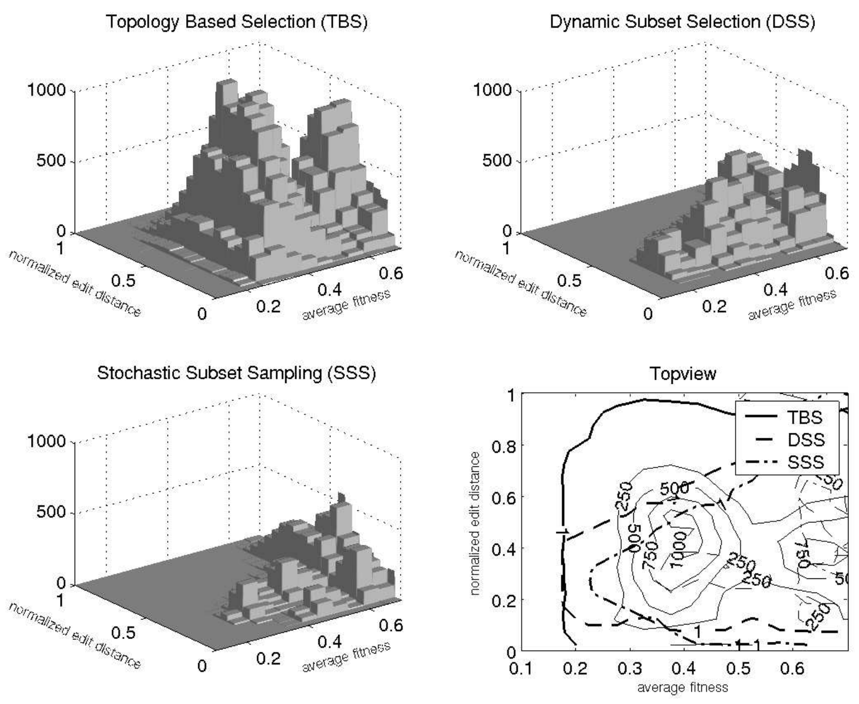


Figure 7: Histograms and contour lines of equipotential surfaces for different frequencies for a population being in a state with a specific average normalized edit distance and average fitness sampled over all generations of 100 runs for each selection method (bin size: 0.05×0.05). The solid contour lines are the level curves for the lowest (same) frequency for each selection method. Runs using topology–based selection are showing populations with good average fitness and high edit distance. While good individuals are the base of success, a large edit distance widens this, which can be taken as an explanation for the good performance of topology–based selection. (The 2–D–histogram shows the frequency only for average fitness better than 0.7. Here, a small fitness value denotes a good individual.)

The emerging topology might be an interesting source of information about the fitness cases. For example, experts of the problem domain could be interested in the reason why some individuals behave similarly with the same fitness cases. Or the topology might help to automatically detect measurement errors (outliers) within the fitness cases (e.g., if fitness cases are taken from measurements of a real world process). We suppose that in later phases of evolution such fitness cases are weakly connected to others.

The method and the definition of similarity of fitness cases is problem independent. Furthermore the method is independent from the learning method. Thus investigating the applicability of topology–based selection to other learning methods from artificial and computational intelligence is an interesting task for future research.

Acknowledgment

This research was funded by a grant to W.B. within the DFG program "Sozionik" under Ba 1042/7–2. P.D. acknowledges financial support by the BMBF (grant 0312704A).

References

Altenberg, L. (1994). The evolution of evolvability in genetic programming. In Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, pages 47–74. MIT Press, Cambridge, MA.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic Programming — An Introduction*. Morgan Kaufmann, San Francisco, CA.

Blake, C. L. and Merz, C. J. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Gathercole, C. (1998). *An Investigation of Supervised Learning in Genetic Programming*. PhD thesis, University of Edinburgh.

Gathercole, C. and Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In Davidor, Y., Schwefel, H.-P., and Männer, R., editors, *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pages 312–321, Berlin. Springer-Verlag.

Gathercole, C. and Ross, P. (1997). Small populations over many generations can beat large populations over few generations in genetic programming. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., and Riolo, R. L., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 111–118, Stanford University, CA, USA. Morgan Kaufmann.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J., editor, *Genetic Algorithms and their Applications (ICGA'87)*, pages 41–49. Lawrence Erlbaum Associates.

Hogg, T. (1996). Refining the phase transition in combinatorial search. *Artificial Intelligence*, 81(1-2):127–154.

Holland, J. (1975). *Adaption in natural and artificial systems*. MIT Press, Cambridge, MA.

Keijzer, M. (2001). *Scientific Discovery using Genetic Programming*. PhD thesis, Technical University of Denmark.

Koza, J. R. (1992a). A genetic approach to the truck backer upper problem and the inter-twined spiral problem. In *Proceedings of IJCNN International Joint Conference on Neural Networks*, volume IV, pages 310–318. IEEE Press.

Koza, J. R. (1992b). *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA.

Lang, K. J. and Witbrock, M. J. (1989). Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 52–59, San Mateo, CA. Morgan Kaufmann.

Lasarczyk, C. W. G. (2002). Trainingsmengenselektion auf der Grundlage einer Fitnesscase–Topologie. Diploma thesis, University of Dortmund.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics - Doklady*, 10(8):707–710. Original in Russian in *Doklady Akademii Nauk SSSR*, 163, 4, 845–848, 1965.

Miglino, O. and Walker, R. (2002). Genetic redundancy in evolving populations of simulated robots. *Artificial Life*, 8(3):265–277.

Nordin, P. and Banzhaf, W. (1997). An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behaviour*, 5(2):107–140.

Schaffer, J. D., Caruana, R. A., Eshelman, L. J., and Das, R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimisation. In Schaffer, J. D., editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 51–60, San Mateo, California. George Mason University, Morgan Kaufmann Publishers.

Voudouris, C. (1998). Guided local search – an illustrative example in function optimisation. *BT Technology Journal*, 16(3):46–50.

Voudouris, C. and Tsang, E. (1996). Partial constraint satisfaction problems and guided local search. In Wallace, M., editor, *Proceedings of the Second International Conference on the Practical Application of Constraint Technology (PACT'96)*, pages 337–356. The Practical Application Company Ltd.

Wagner, P. and Altenberg, L. (1996). Complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976.

Walsh, T. (1999). Search in a small world. In Thomas, D., editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, volume 2, pages 1172–1177, San Francisco, CA, USA. Morgan Kaufmann Publishers.

Zhang, B.-T. and Cho, D.-Y. (1998). Genetic programming with active data selection. In Newton, C., editor, *Proceedings of the Second Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'98)*, volume 1.