Evolving Teams of Multiple Predictors with 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

This paper reports on the evolution of GP teams in different classification and regression problems and compares different methods for combining the outputs of the team programs. These include hybrid approaches where (1) a neural network is used to optimize the weights of programs in a team for a common decision and (2) a real-numbered vector of weights (the representation of evolution strategies) is evolved with each team in parallel. The cooperative team approach results in an improved training and generalization performance compared to the standard GP method. The higher computational overhead of coevolving several genetic programs is counteracted by using a fast variant of linear GP. In particular, the processing time of linear genetic programs is reduced significantly by removing intron code before program execution.

1 Introduction

Genetic programming (GP) has been formulated originally as an evolutionary method for breeding programs using expressions from the functional programming language LISP [6]. We employ linear GP [9, 2, 1], a genetic programming variant using sequences of instructions of an imperative programming language (C here), for the evolution of teams. The team approach is applied to prediction problems including both classifications and regressions.

The linear variant of GP operates on genetic programs being represented as linear sequences of C instructions. One strength of linear GP is that most of the introns, i.e. instructions that do not effect program behavior, can be removed before a genetic program is executed during fitness calculation. This does not cause any change to the individual representation in the population but results in an enormous speedup [2]. In this way intron elimination can compensate the increase in runtime caused by the evolution of teams.

Team evolution is motivated strongly by natural evolution. Many predators, e.g. lions, have learned to hunt pray in a pack most successfully. By doing so, they have a much better chance to survive than single fellows. In GP the parallel evolution of team programs is expected to solve a task more efficiently than the usual evolution of individuals. Thereto the team individuals have to solve the overall task in cooperation by specializing in subtasks for a certain degree. Evolution of heterogenous teams with restricted recombination is used to promote specialization of members.

Team solutions require the multiple decisions of their members to be merged into a collective decision. Several methods to combine the outputs of team programs are compared in this work. The team approach not only allows the combined error to be optimized but also an optimal *composition* of the programs to be found. In general the optimal team composition is different from simply taking individual programs that are already quite perfect predictors for themselves. Moreover, with the coevolutionary approach the diversity of the individual decisions of a team may become an object of optimization.

This contribution also presents a combination of GP and neural networks, the weighting of multiple team programs by a linear neural network. The neural optimization of weights results in an improved performance compared to standard combination methods. In another hybrid approach the representations of linear GP and evolution strategies (ES) [12] are coevolved in that a vector of programs (team) and a vector of program weights form one individual and undergo evolution and fitness calculation simultaneously.

2 Evolution of teams

In GP the evolution of teams has been investigated mostly in connection with cooperating agents solving multi-agent control problems. Luke and Spector [8] tested teamwork of homogeneous and heterogeneous agent teams in a predator/prey domain and showed that the heterogenous approach is superior. In contrast to heterogenous teams homogeneous teams are composed of completely identical agents and can be evolved with the standard GP approach. In [4, 5] Haynes et al. tested a similar problem with different recombination operators for heterogeneous teams. Recently Soule [14] published a paper where he solves a non-control problem, a parity function problem, with teams using majority voting to combine the individual decisions.

In our paper the team approach is applied to different prediction problems, two classification tasks and one regression task. In contrast to control tasks only heterogenous teams are of interest here, because for prediction tasks there is nothing to be gained from the combination of the outputs of completely identical programs (homogeneous teams).

2.1 Team representation

In general teams of individuals can be implemented in different ways. Firstly, a certain number of individuals can be selected randomly from the population and evaluated in combination as a team. The problem with this approach is known as the *credit assignment problem*: The combined fitness value of the team has to be shared and distributed among the team members (*fitness sharing*).

Secondly, team members can be evolved in separate subpopulations which provide a more specialized development. In this case, the composition and the evaluation of teams might be separated from the evolution of their members by simply taking the best individuals from each deme in each generation and combining them. However, this raises another problem: An optimal team is not necessarily composed of best individuals for each team position. Specialization and coordination of the team's individuals is not a matter of evolution there. These phenomena might only emerge accidentally.

The third approach favoured here is to use an explicit team representation that is considered as one individual by the evolutionary algorithm [5]. The population is subdivided into fixed, equal-sized groups of individuals. Each member is assigned a fixed position index. In this way team members undergo a coevolutionary process because they are always selected, evaluated and varied simultaneously. This eliminates the credit assignment problem and renders the composition of teams an object of evolution.

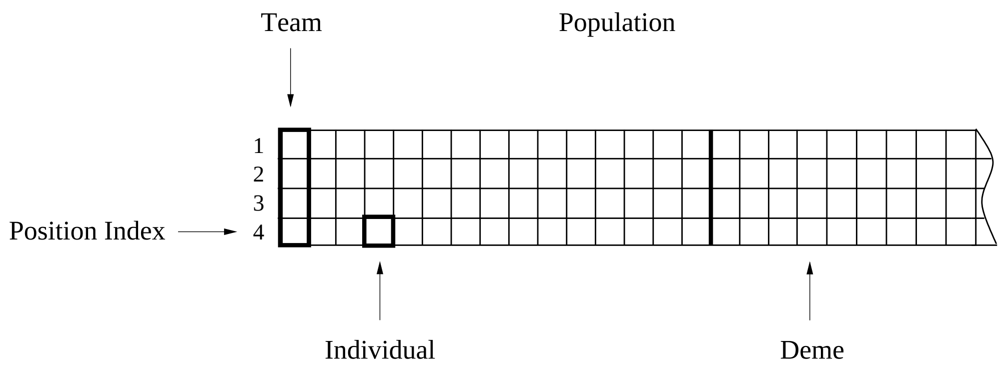

Figure 1: Population subdivded into teams and demes.

Figure 1 shows the partitioning of the total population used in the experiments described below. First, the population is subdivided into *demes* [15] which, in turn, are subdivided into *teams* of individual programs (*demes of teams*). Following the biological *island model*, individual teams are allowed to migrate between *arbitrary* demes. This is realized by selecting teams (tournament winners) occasionally from different demes and reproducing them in other demes. Demes are used here because they better preserve the diversity of the population [2]. This, in turn, would reduce the probability of the evolutionary process getting stuck in a local minimum.

The coevolutionary approach prohibits teams of arbitrary size because the complexity of the search space and the training time, respectively, grow exponentially with the number of coevolved programs. On the other hand, the team size has to be large enough to cause an improved prediction compared to the traditional approach, i.e. team size one. This trade-off situation and our experimental experience let us use rather moderate numbers of members (see Section 6).

2.2 Team operators

Team representations require special genetic operators, notably for recombination. Genetic operations on teams in general reduce to the respective operations on their members which can be selected randomly. Haynes et al. [5] found that a moderate number of crossover points works better than recombining either one or every team position per operation. This is due to the trade-off between a sufficient variation and the destructive effect of changing too many team members at the same time.

For recombination the participating individuals of the two parent teams can be chosen of arbitrary or equal position. If recombination between different team positions is not allowed, team members evolve independently in isolated “member demes”. Luke and Spector [8] already showed that team recombination restricted in this way can outperform free recombination. Isolated or semi-isolated coevolution of the team members is argued to promote specialization in behaviour. In this contribution we do not allow recombination between different team positions because we are interested in team programs which disagree on some decisions (see Section 3.1).

A possible alternative to a random selection might be genetic operators that modify the team members depending on their respective individual fitness. Members may be sorted by error and the probability that an individual becomes a subject of mutation or crossover depends on its error rank. But only a limited number of members is allowed to change simultaneously. By doing so, *worse* individuals are varied more often than better ones on average. On the one hand improving the fitness of worse individuals might have a better chance to improve the overall fitness of the team. But this does not hold for all combination methods discussed below. Beyond that, there is not necessarily a positive relationship between better member fitness and better team fitness for the problem definition considered. On the other hand this technique does not allow the error of the team members to differ much what might have a negative effect on specialization.

3 Combination of multiple predictors

In principle, this paper integrates two research topics, the evolution of teams discussed above and the combination of multiple predictors, i.e. multiple classifiers or regressors. In contrast to teams of agents teams whose members solve a prediction problem require the aggregation of the member's output to produce a common decision.

In the neural network community different approaches have been investigated dealing with the combination of multiple decisions in neural network *ensembles* [3, 10, 7]. Usually, neural networks are combined after training and are hence already quite perfect in solving a classification or approximation problem on their own. The ensemble members are not trained in combination and the composition of the ensemble does not undergo an optimization process.

In [17] neural networks are evolved and a subset of the final population is combined afterwards. Different combination methods—including averaging and majority voting—are compared while a genetic algorithm is used to search for a near optimal ensemble composition.

For genetic programming Zhang et al. [18] applied a weighted majority algorithm in classification to combine the Boolean outputs of a selected subpopulation of genetic programs after evolution. This approach resulted in an improvement in generalization performance, i.e. robustness, compared to standard GP and simple majority voting, especially with sparse and noisy training data.

The decisions of different *types* of classifiers including neural networks and genetic programs are combined by an averaging technique in [13]. As a result an improved prediction of thyroid normal and thyroid carcinoma classes has been achieved in a medical application.

3.1 Making multiple decisions differ

In principle, all members in a team of predictors solve the same full task. The problem is not artificially subdivided among the team positions and there are no subproblems (subsets of data) assigned to special members explicitly. Since in many cases the problem structure is completely unknown we are interested in teams where specialization, i.e. a partitioning of the solution, emerges from the evolutionary process itself.

Specialization strongly depends on the heterogeneity of the teams. Heterogeneity is achieved by evolving members that produce slightly diverging outputs for the same input situations. Nothing will be gained from the combination of the outputs of completely identical predictors (homologous teams) as far as the quality of the solutions is concerned. Note that this is in contrast to agent teams that solve a control task. Each agent program usually has side effects on the problem enviroment.

In genetic programming the inherent noise of the evolutionary algorithm already provides a certain heterogeneity of teams. Besides the restricted recombination scheme (see Section 2.2) used here there are more specific techniques to increase heterogeneity and, thus, promote the evolution of specialization:

One possible approach is to force the programs of a team to disagree on decisions and to specialize in different domains of the training data. This can be achieved by either using different fitness functions for the programs of a team or by training each team position with different subsets of the original training dataset. Both techniques require the individual errors of the team members to be integrated into the fitness function (see Section 5.2). Otherwise, the effect of the different input situations cannot be made known to the evolutionary algorithm if you take into account that only member outputs of equal input situations can be used to calculate the combined error of the team.

Leaving out non-overlapping subsets is similiar to *k-fold cross validation* (k is the number of team members), a method used to improve the generalization capabilities of neural networks over multiple runs. The training subsets can either be sampled randomly at the beginning of each run or, alternatively, resampled after a certain number of generations. The latter technique, called *stochastic sampling*, introduces some additional noise in the sampling process. It allows smaller and more different subsets to be used for the individual members since it guarantees that every team position over time is confronted with every training example.

On the other hand, different function sets can be chosen for different team positions to promote specialization as well. Of course, the team crossover operator has to be adapted in a way that only individual members from the same function set are allowed to be recombined. Also the recombination between individuals of different positions must be restricted, respectively.

3.2 Combination methods

Two main approaches can be distinguished concerning the combination of individual solutions in genetic programming: Either the individuals (genetic programs) can be evolved independently in different runs and combined *after* evolution. Or a certain number of individuals are *coevolved* in parallel as a *team*. The focus of this paper is on the second approach. Post-evolutionary combination suffers from the drawback that successfull compositions of programs are detected randomly only. That might require a lot of runs to develop a sufficient number of individual solutions. Coevolution of k programs instead will turn out to be much more efficient in time than k independent runs.

The problem that arises with the evolution of teams is in the combination of the outputs of the individual members during fitness evaluation of a team. Different *combination methods* have been tested here. All methods compute the resulting team output from a *linear combination* of its member's outputs. Non-linear methods cannot necessarily be expected to produce better aggregations of multiple predictions since the actual problem, linear or non-linear, is already solved by the single predictors. Figure 2 illustrates the general principle of the approach.

Moreover, only basic combination methods are documented and compared in this contribution. Even if there are hybridizations of the methods possible, e.g. EVOL/OPT or EVOL/MV (weighted majority voting), the concurrent application of two combinations is not necessarily more successfull. We noticed that more complicated combination schemes are rather difficult to handle for the evolutionary algorithm. These might be more reasonable with post-evolutionary combinations of (independent) predictors. Most of the methods—except WTA (see Section 3.2.6)—can be applied to parallel as well as to sequentially evolved programs

For classification problems there exist two major possibilities to combine the outputs of multiple predictors: Either the raw output values or the classification decisions can be aggregated. In the latter case the team members act as full (pre-)classificators themselves. The downside is that by mapping the continuous outputs in discrete class identifiers *before* combining them reduces the information content each individual might contribute to the common team decision. This could restrict specialization as well as cooperation. Therefore, we decided for the former and combined raw outputs—except for majority voting (see below) that requires class decisions implicitly.

Some of the combination methods are only applicable to classification tasks and are based upon one of the following two *classification methods*:

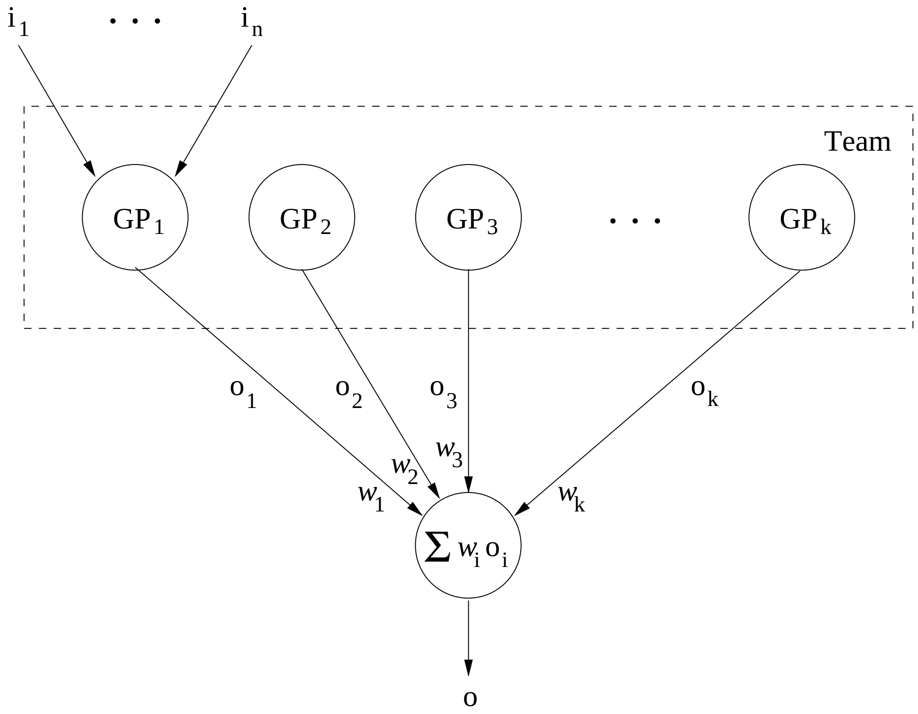

Figure 2: Linear combination of genetic programs.

Classification with intervals (INT). Each output class of the problem definition corresponds to a certain interval of the full range of the (single) program output. In particular, for classification problems with two output classes the continuous program output is mapped to class output 0 or 1 here — depending on a classification threshold of 0.5.

Winner-takes-all classification (WTA). Here for *each* output class exactly one program output is necessary. The output with the highest value determines the class decision of the individual. This method is especially interesting for higher dimensional program outputs.

These different combination methods are introduced for problems with two output classes while a generalization to more output classes is not complicated. Even more important is to note that none of the methods presented here produces relevant extra computational costs.

3.2.1 Averaging (AV)

There are different variants of combination possible by computing a weighted sum of the outputs of the team programs. The simplest form is to use uniform weights for all members, i.e. the *simple average* of k outputs as team output. In this way the influence of each individual on the team decision is exactly the same. The evolutionary algorithm has to adapt the team members to the fixed weighting only.

$$o_{team} = \sum_{i=1}^{k} \frac{1}{k} o_{ind_i} \tag{1}$$

3.2.2 Weighting by error (ERR)

An extended method is to use the fitness information of each team member for the computation of its weight. By doing so, better individuals get a higher influence on the team output than worse.

$$w_i = 1/e^{\beta E(gp_i)}. \tag{2}$$

$E(gp_i)$ is the individual error explained in Equation (10). β is a positive scaling factor to control the relation of the weight sizes. The error-based weighting gives lower weights to worse team members and higher weights to better ones.

In order to restrict their range the weights always undergo normalization in that they are all positive and sum to one:

$$w_i = \left\| \frac{w_i}{\sum_{j=1}^{k} w_j} \right\| \tag{3}$$

With this approach evolution decides over the weights of a program member by manipulating its error value. In our experiments the individual weights are adjusted during training using the fitness information. Using data different from the training data may reduce overfitting of teams and increase their generalization performance. It has, however, the drawback of increasing computation time.

In general, the error-based weighting approach has not been found to be consistently better than the average of member outputs. The reason might be that the quality of a single member solution must not be directly related to the fitness of the whole team. If the combined programs had been evolved in single independent runs, deriving the member weights from this independent fitness might be a better choice. In such a case stronger dependencies between programs—that usually emerge during team evolution by specialization—cannot be expected.

3.2.3 Coevolution of weights (EVOL)

With this approach member weights are evolved in parallel with every team in the population (see Figure 3). The real-valued vector of weights is selected together with the vector of programs (team) by tournament selection. During each fitness evaluation the weight vector is varied by a sequence of mutation operations ("macro mutation"). Only better mutations are allowed to change the current state of weighting, a method typical for an (1+1)ES [12]. The mutation operator updates single weight values by allowing a constant standard deviation (*mutation step size*) of 0.02. The initial weights are randomly selected from the interval $[0, 1]$. Recombination of the weight vectors is not applied.

Alternatively, a complete (1+1)ES run might be initiated to optimize the weighting of each team during fitness calculation. This, of course, increases the computational costs significantly depending on the run length. It also might not be necessarily advantageous since the program teams adapt to a given weighting situation concurrently. With our approach optimization of the weighting is happening in coevolution with the members, not

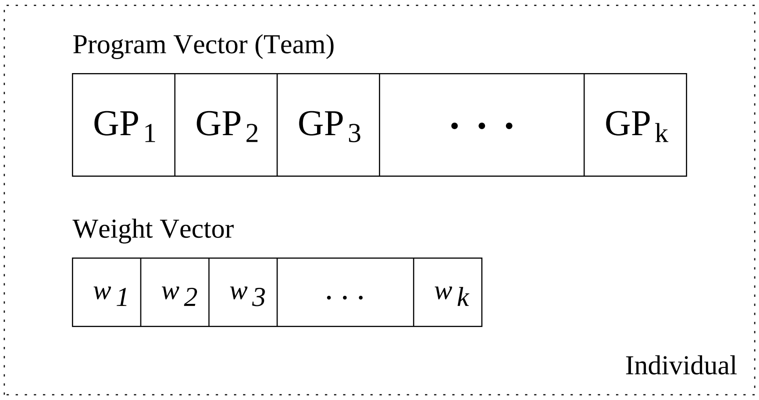

Figure 3: Coevolution of program team and vector of weights as individual.

during each team evaluation. Thus, the coevolutionary aspect that allows team solutions to adapt to different weighting situations is the most important point here.

Even if the diversity of the population decreases at the end of a GP run there are still improvements possible by changing the influences of the single team members.

3.2.4 Majority voting (MV)

A special form of linear combination is *majority voting* which operates on *class* outputs. In other words, the continuous outputs of team programs are transformed into discrete class decisions *before* they are combined.

Let us assume that there are exactly two output classes, 0 and 1. Let O_c denote the subset of team members that predict class c:

$$O_0 := \{i | o_{ind_i} = 0, i = 1, .., k\} \tag{4}$$

$$O_1 := \{i | o_{ind_i} = 1, i = 1, .., k\} \tag{5}$$

The class which most of the individuals predict for a given example is selected as team output:

$$o_{team} = \begin{cases} 0 & : & |O_1| < |O_0| \\ 1 & : & |O_1| \geq |O_0| \end{cases} \tag{6}$$

Note that clear decisions are forced for two output classes if an uneven number of team members participates. Majority voting works as well with an even number of members as long as the team decision is defined for equality (class 1 here).

3.2.5 Weighted voting (WV)

Another voting method, *weighted voting*, is introduced here for the winner-takes-all classification (see above) where each team program returns exactly one output value for each of

m output classes. For all classes c these values are summed to form the respective outputs of the team:

$$o_{team,c} = \sum_{i=1}^{k} o_{ind_i,c} \forall c \in \{0, .., m\} \tag{7}$$

The class with the highest output value defines the response class of the team as illustrated in figure 4.

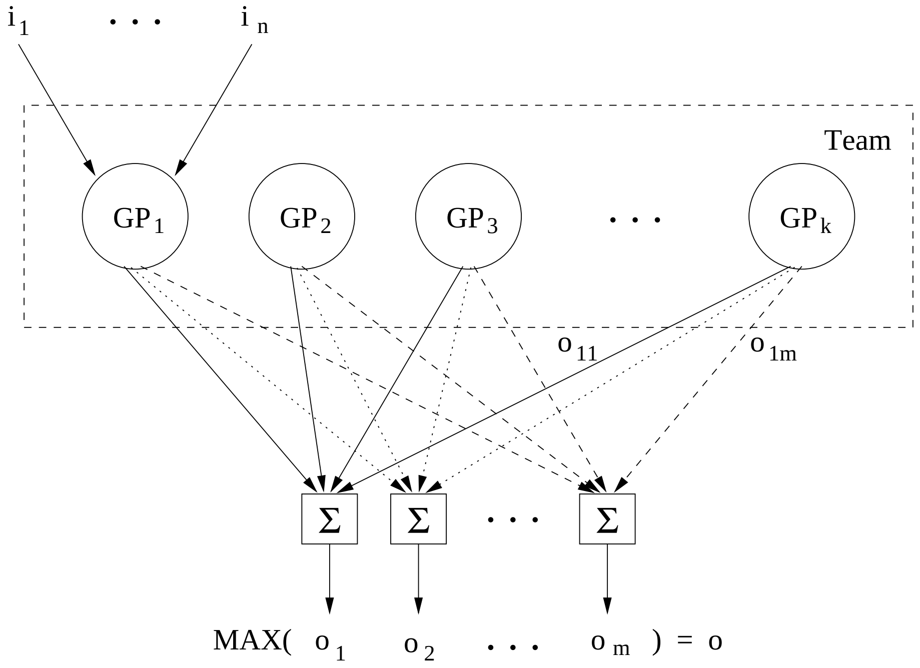

Figure 4: Combination of genetic programs by weighted voting.

With this combination method each team individual contributes a continuous "weight" for each class instead of a clear class decision as in Section 3.2.4. If discrete (class) outputs would be used the method corresponds to majority voting. Here the weighting comes from the member programs themselves. When using interval classification instead of WTA classification each program might compute its own weight in a separate (second) output variable alternatively.

3.2.6 Winner-takes-all (WTA)

Two different *winner-takes-all combination* methods are presented in this contribution:

The *first* WTA combination variant selects the individual with the *clearest class decision* to determine the output of a team. With interval classification the member output that is closest to one of the class numbers (0 or 1) is identified as the clearest decision. The winner may also be seen as the individual with the highest *confidence* in its decision. Specialization may emerge if different members of the team win this contest for different fitness cases.

$$o_{team} = o_{win} \tag{8}$$

If seperate outputs are used instead of output intervals (WTA *classification*) the clearest decision might be defined as the biggest difference between the highest output and the second highest output of a team member.

The *second* and simplest WTA combination (WTA2) just chooses the *lowest output* as team output. (Note that this is definition and could be the greatest output as well.) This selection happens *before* the continuous outputs are transformed into class decisions and is valid for interval classification. For WTA classification the member with the lowest sum of outputs could be choosen. This combination variant is also possible for regression problems.

Of course, it is not a feasible alternative to select the member with the best fitness. Than a decision on unkown data is only possible if the right outputs are known in advance and is not made by the team itself.

3.2.7 Weight optimization (OPT)

The final approach tested here uses a *linear* neural network in form of a perceptron *without* hidden nodes to find an optimal weighting of the team individuals. The learning method applied is RPROP [11], a backpropagation variant about as fast as Quickprop but with less adjustments of the parameters necessary. With this approach data is processed first by the team programs before the neural network combines their results (see also Figure 2). Actually, only a single neuron weights the connections to the genetic programs whose outputs represent the input layer of the linear neural network here. The outputs of the programs are, of course, only computed once for all data inputs before the neural weighting starts. In [16] a linear perceptron has been used to learn the averaging weights of an ensemble of trained perceptrons.

Like with the other approaches the neural weighting might be done each time the fitness of a team is calculated. Obviously, this has the drawback of an exponential increase in runtime even with a small neural network and a relatively low number of epochs trained. A much less time-consuming variant applied here is to use a neural network for optimizing the weights of the best teams only before (re)computing the training and validation error with the new weights. By doing so, the process of finding an optimum weighting for the members is decoupled from the contrary process of breeding team individuals with a more balanced share in cooperation. In other words, worse members cannot so easily be "weighted out" of a team just by assigning them very low weights.

Weighting is an inherent property of neural networks. The linear network structure assures that there is only a weighting of program outputs possible by the neural network and that the actual, non-linear problem is solved exclusively by the genetic programs. Thus, the genetic programs form some kind of "hidden layer" in the GP/NN hybrid.

Instead of using hill-climbing by a neural networks, evolutionary techniques, like evolution strategies or simulated annealing, might be applied for the adaptation of weights.

4 Linear genetic programming

In the experiments described below we use *linear* GP, a genetic programming approach with a linear representation of individuals that has been introduced by Nordin and Banzhaf [9, 1]. Its main characteristic in comparison to tree-based GP is that not expressions of a functional programming language (like LISP) but programs of an imperative language (like C or machine code) are evolved.

In the linear GP system used for our experiments [2] an individual program is represented as a variable length sequence of simple C instructions. All instructions operate on one or two indexed variables v_i or constants c from a predefined range and assign the result to a destination variable v_j, e.g. $v_j = v_i + c$. The operation set used for the experiments in this contribution includes *addition, subtraction, multiplication, division* and *exponentiation.*

4.1 Removing non-effective code

Non-effective code in a genetic program specifies instructions without any influence on the calculation of the output for *all* possible inputs. These so-called *introns* are believed to act as redundant code segments that protect advantageous building-blocks from being destroyed by crossover.

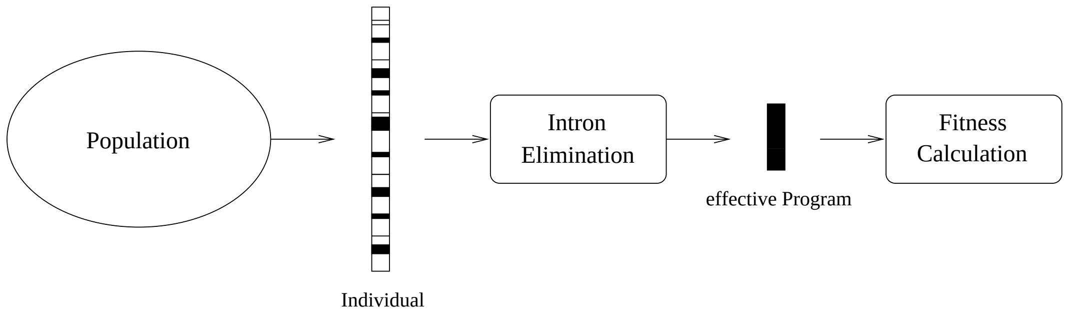

Figure 5: Intron elimination in linear GP.

The program structure in linear GP allows non-effective code to be detected and eliminated efficiently. The intron removal algorithm [2] achieves this in linear runtime $O(n)$, with n is the maximum length of the linear programs. Prior to fitness evaluation the *effective* instructions are copied to a temporary program buffer which is executed subsequently. By doing so, the representation of individuals in the population remains unchanged while the computation time for non-effective code is saved (see figure 5).

By skipping the execution of the non-effective code during program interpretation the evolutionary process is accelerated by a factor $\frac{1}{1-p}$, where p denotes the average percentage of redundant program part. In most applications documented below an average intron rate of about 80% has been observed resulting in a speedup factor of about 5 through intron elimination. In other words, about five effective team members could be executed with the same time requirements as a single standard individual including its introns. Thus, the additional computational overhead of team evolution reduces significantly with linear GP and the elimination of non-effective code.

5 Experimental setup

We examine the team approach with different combination methods discussed earlier using two classification problems and one regression problem. First of all, the structure of the data that represents the respective problems is documented in further detail.

5.1 Structure of the experimental data

The *heart* dataset is composed of four datasets from the UCI Machine Learning Repository (*Cleveland, Hungary,* and *Switzerland*) and includes 720 examples altogether. The input dimension is 13 while two output classes (1 or 0) indicate the diagnosis (ill or not ill). The heart problem incorporates noise because inputs—including continuous and discrete values—are missing and have been completed with 0. The diagnosis task of the problem is to predict whether the diameter of at least one of four major heart vessel is reduced by more than 50% or not.

Two chains denotes a popular machine learning problem where two chained rings that represent two different classes—of about 400 data points each—have to be seperated. The two rings in Figure 6 "touch" each other at two regions without intersection.

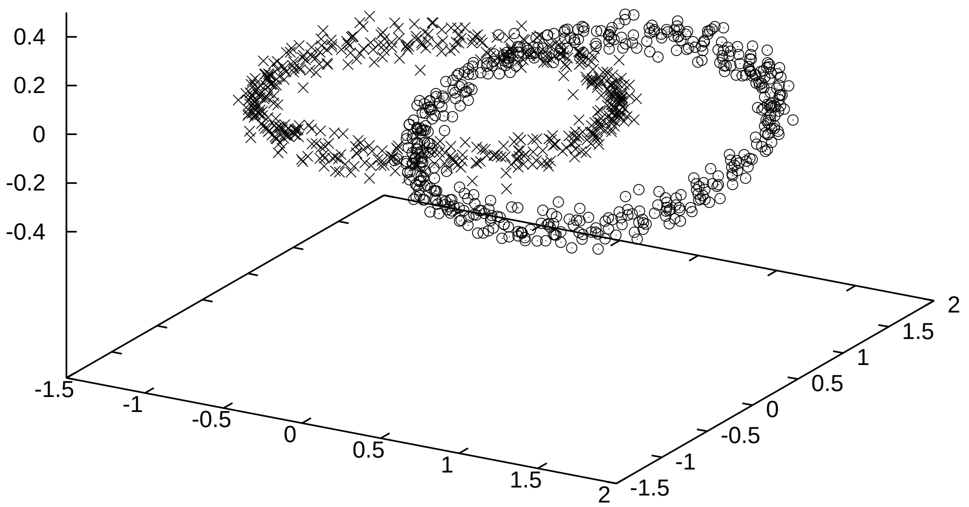

Figure 6: *Two chains* problem.

The regression problem *three functions* tests the ability of teams to learn three different functions at the same time which consist of a sinus, a logarithm and a half circle (see Figure 7). A function index has to be passed to the genetic programs as an additional input to distinguish the three functions.

In all cases, the data examples were subdivided randomly into three sets: training set (50%), validation set (25%) and test set (25%). Each time a new best team emerges its

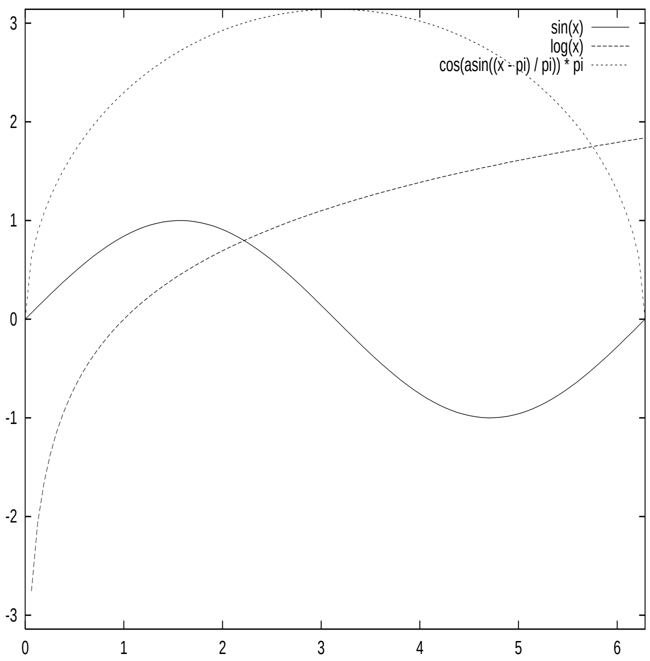

Figure 7: *Three functions* problem.

error is calculated using the validation set in order to check its generalization ability *during* training. From all these best teams emerging over a run the one with minimum validation error is tested on the test set once *after* the training is over.

5.2 Team fitness

The *fitness* F of a team might integrate two major goals: the overall error of the team $E(team)$ and (optionally) the errors of its program members $E(gp_j)$ should be minimized.

$$F(team) = E(team) + \delta \frac{1}{m} \sum_{j=1}^{m} E(gp_j) \tag{9}$$

The influence of the average member error on the team fitness is controlled by a multiplicative parameter δ. Including the individual errors as a second fitness objective (by choosing $\delta = 1$) has not been experienced to produce necessarily better results. One effect is that the average fitness of the members in a team becomes significantly better. Actually it might reduce the specialization potential of the members since the cooperating individuals are restricted to be good predictors of their own.

If, on the other hand, the individual errors are not included into the fitness function ($\delta = 0$) there is no direct relation between the fitness of the single members and the quality of the common team solution. This allows the errors of members to differ more strongly within a team and to be significantly worse than the team error. For all experiments documented in this work δ has been set to 0.

In Equation (9) E denotes the *error* of a predictor p that is computed as the sum of square distances between the predicted output $p(\vec{i_k})$ and the desired output $\vec{o_k}$ over n examples

$(\vec{i_k}, \vec{o_k})$:

$$E(p) = \sum_{k=1}^{n} (p(\vec{i_k}) - \vec{o_k})^2 + \alpha CE \tag{10}$$

The *classification error* (CE) is calculated as the number of incorrectly classified examples in Equation (10). The influence of the classification error is controlled by a weight parameter α. For classification problems α has been set constantly to 2 in order to favour classification quality (0 otherwise).

5.3 Parameter settings

Parameter	Setting
Number of generations	1000
Number of teams (population size)	3000
Number of team members	4
Number of varied team members	1-2
Number of demes	6
Migration rate	3%
Migration frequency (in generations)	1
Crossover probability for teams	100%
Mutation probability for teams	100%
Mutation step size for constants	5
Instruction set	$\{+, -, \times, /, pow\}$
Set of (integer) constants	{0,..,100}
Maximum individual length (in instructions)	128

Table 1: General parameter settings.

Table 1 lists the parameter settings of our linear GP system used for the evolution of teams and all problem definitions described above. The population size is 3000 teams while each team is composed of the same number of individual members. The population has been choosen sufficiently large to conserve diversity of the more complex team solutions. The total *number of members per team* and the *number of members that are varied* during crossover and mutation are the most important parameters when investigating the evolution of teams. Different settings of these parameters are reported in further detail in the next section.

The number of generations is limited to 1000, both for GP teams and the standard GP approach. Note that a single individual is varied much less—one or two member per team recombination only— than an individual during a standard GP run. While this reduces the progress speed of single team members it does not necessarily hold for the fitness progress of the whole team as we will see below.

A single program is not allowed to become longer than 128 instructions in our experiments. For all tested problems this has been experienced to be a sufficient length for representing powerful solutions. Longer programs cannot always be expected to produce better results. The effective part of best solutions usually depends strongly on the problem and does not vary much in size between runs [2]. Thus, the longer a program becomes the more non-effective code it has to maintain.

The selected standard set of instructions—including *addition, subtraction, multiplication, protected division,* and the *protected power* function—should be powerful enough for not producing too restrictive solutions for the three prediction tasks.

6 Results

We now document the results obtained by applying the different team approaches described in 3.2 to the three problems of Section 5.1. Prediction accuracies and code sizes are compared for the team configurations and a standard GP approach.

The team approach, in general, has been found to produce better results than the standard GP approach for all three prediction tasks. Mainly problems profit from GP teams whose solution can at least be divided partly into subsolutions and distributed to different problem solvers (team members). Especially data that hold linearly separable subsets can take advantage. Moreover, team solutions can be expected to be less brittle and more general in the presence of noise due to their collective decision making. Only if nearly optimal solutions already emerge with the standard approach teams cannot be expected to be benefical. In this case the additional computational overhead of the more complex team solutions outweighs the advantages.

6.1 Prediction accuracy

Table 2 summarizes the different configurations of the team approaches tested in this contribution. The outputs of the team members are continuous except for majority voting (MV) where the raw outputs have to be mapped on discrete class identifiers first. Only our weighted voting approach (WV) is based on the WTA classification method. All other methods use interval classification.

Method	ID	Combination	Classification	Outputs
GP	GP	—	INT	cont
TeamGP	AV	AVeraging (standard)	INT	cont
TeamGP	OPT	weight OPTimization	INT	cont
TeamGP	ERR	weighting by ERRor	INT	cont
TeamGP	EVOL	coEVOLution of weights	INT	cont
TeamGP	MV	Majority Voting	INT	class
TeamGP	WV	Weighted Voting	WTA	cont
TeamGP	WTA	Winner-Takes-All	INT	cont
TeamGP	WTA2	Winner-Takes-All	INT	cont

Table 2: Configuration of the different team approaches.

The following tables compare error rates of the standard GP approach and the different team approaches for the three test problems described in Section 5. Minimum training error and minimum validation error are determined among best solutions (concerning fitness) of a run. The solution with minimum validation error is applied to unknown data at the end of a run to compute the test error. All figures given in this paper denote average results from series of 60 test runs. In order to avoid unfair initial conditions and to give more reliable results each test series (configuration) has been performed with the same set of 60 random seeds.

Method	Training CE (%)	Member CE (%)	Validation CE (%)	Test CE (%)
GP	3.67	3.7	5.07	5.69
AV	0.64	11.7	1.25	2.20
OPT	0.59	26.7	0.93	2.44
ERR	1.31	20.9	1.91	2.73
EVOL	0.33	28.0	0.71	2.00
MV	0.37	25.7	1.48	2.17
WV	0.39	27.7	0.76	1.91
WTA	0.02	59.2	**0.00**	**0.33**
WTA2	**0.00**	64.3	0.00	0.65

Table 3: *Two chains*: Classification errors (CE) in percent. Best team results highlighted.

Considering the classification rates for the *two chains* problem in Table 3 the *standard team approach* (AV) reaches approximately a 5 times better performance than the standard GP approach.

Most interesting are the results of the winner-takes-all combination that select a *single* member program to decide for the team on a certain input situation. Both team variants (WTA and WTA2) nearly always found the optimum (0% CE) for training data and validation data. With standard GP the optimum solution has not even emerged once during 60 trials here. This is a strong indication of a high specialization of the team members. It demonstrates clearly that highly coordinated behaviour emerges from the parallel evolution of programs. This cannot be achieved by a combination of standard GP programs which would have to be evolved independently. Team evolution is much more sophisticated than just testing random compositions of programs. In fact, the different members in a team have adapted strongly to each other during the coevolutionary process.

Among the *real* team approaches which combine outputs of *several* individual members WV was the most successful alternative. This is remarkable because this method requires twice as much output values—two instead of one output per member—to be coordinated.

Method	Training CE (%)	Member CE (%)	Validation CE (%)	Test CE (%)
GP	13.6	13.6	14.5	19.0
AV	11.5	13.2	13.4	18.2
OPT	11.5	32.0	12.8	**17.5**
ERR	11.9	28.6	12.9	18.0
EVOL	11.4	32.9	**12.7**	18.1
MV	**10.9**	24.6	13.6	17.5
WV	11.5	32.4	12.9	17.9
WTA	11.9	60.5	14.5	18.5
WTA2	12.9	61.5	14.9	19.2

Table 4: *Heart*: Classification errors in percent. Best team results highlighted.

Table 4 shows the prediction results for the *heart* problem. This application demonstrates not only the ability of teams in real data-mining but also in noisy problem enviroments since many data attributes are missing or are unknown. The difference in prediction error between GP and TeamGP is about 2% which is significant in the respective real problem domain. The problem structure does not offer many possibilities for specialization, especially in case of the winner-takes-all approaches which do not generalize significantly better here than the standard approach. The main benefit of the other combination methods seems to be that they improve fitness and generalization quality for the noisy data by a *collective* decision making of *more than one* team program.

Method	Training MSE	Member MSE	Validation MSE	Test MSE
GP	16.9	17	16.2	16.6
AV	4.9	411	4.1	4.5
OPT	4.6	619	3.8	4.1
ERR	4.6	6340838	3.9	4.0
EVOL	**3.2**	33135	**2.6**	**2.7**
WTA2	11.0	154762629	9.8	10.1

Table 5: *Three functions*: Mean square error (MSE $\times$ 100). Best team results highlighted.

Experimental results for the *three functions* problem are given in Table 5. Note that not all team variants are applicable to a regression problem. The regression task at hand has been solved most successfully by EVOL teams. This combination variant allows different weighting situations to be coevolved with the program teams and results in nearly twice as small prediction errors compared to uniform weights (AV). The standard team approach is found to be about four times better in training and generalization than the standard GP approach. Note that the average member error is extremely high compared to the respective team error with this problem.

Finally, some general conclusions can be drawn from the three applications:

Teams of predictors have proven to give superior results for known data as well as unknown data. The improved generalization performance of teams results from the increased robustness of team solutions against noise in the data space. This, in turn, is mainly due to the combination of multiple predictions that absorb ("smooth") larger errors or wrong decisions made by single members. In all three test cases not only the given average results but also the standard deviations (not shown in the tables) reduce with teams. In general, there are less "outliers" among the test runs using teams.

Comparing the different team configurations among each other further shows that different combination methods dominate for different problems. A general ranking of the methods cannot be produced. It is worth trying several variants when dealing with the evolution of multiple predictors.

Optimizing the weights of the best teams (OPT) that occur during evolution by using a neural network improved the results (AV) significantly. But even more successful was the parallel evolution of weights together with the team programs (EVOL)—the second hybrid approach presented. In general, most methods that allow various weighting situations outperformed the standard team approach using uniform weights.

For all three examples the average member error was highest with winner-takes-all combinations. This is not suprising since only one member is selected to make a final decision for the whole team while outputs of the other team individuals could be arbitrarily worse

(WTA) or bigger (WTA2) respectively. Obviously specialization potential is highest with this combination. In case of all other team methods with varying weights (e.g. EVOL) the member errors are higher than with uniform weights (AV). The individual performance in an AV team again is worser than the performance of stand-alone GP individuals.

6.2 Code size

With linear GP the higher complexity of team evolution and the resulting increase in computation time are counteractive in two ways:

1. By executing the effective code only which makes the evolution of teams with a modest number of members computationally affordable.
2. By the fact that the effective code size of a team with k members is found significantly smaller than the effective size of k individual GP solutions.

Firstly, the non-effective intron code (see Section 4) does not cause any computational costs no matter how complex it might become during the evolutionary process. This reduces non-effective code and absolute size respectively to be interesting for protecting the effective program parts and for genetic diversity in general.

The second effect is demonstrated in this section by comparing effective code sizes (in number of instructions) for different team configurations and standard GP. In linear GP only the effective program code (as defined in Section 4) has an influence on fitness. If no parsimony pressure is used there is no selection pressure on the non-effective code parts possible. As a result, the absolute program length grows unbounded usually until the maximum size limit is reached.

For the three example cases Figures 8, 9, and 10 visualize the development in effective code size of teams holding four members. The absolute code size approaches mostly the maximum and is not given here. WV combination that is based on winner-takes-all classification produces the largest teams. WTA teams are found to be smallest in code size. Actually they are not much bigger than a single standard individual. This might be seen as another indication for the high specialization of the members in those teams.

In general—including all different combination variants—teams become only about twice as big as standard individuals. For the heart problem they are not even 50% bigger. That means that, on average, a team member is definitely smaller than a standard individual. All graphs show the effective length of best solutions. The average effective length in the population has developed quite similar. As a result, the differences in effective size correspond directly to the differences in computation time when using intron elimination in linear GP (see Section 4.1).

One reason for the reduced growth of the (effective) team members could be seen in the *lower variation probability* compared to standard GP individuals. We will see in the following Section 6.3 that it is not recommended to vary too many members concurrently during a team crossover operation. Best team prediction is obtained by varying about one member only. But if only one team member is changed the probability for crossover at a certain team position is reduced by a factor equal to the number of members. One might conclude that member programs grow faster the more members are varied. That this is

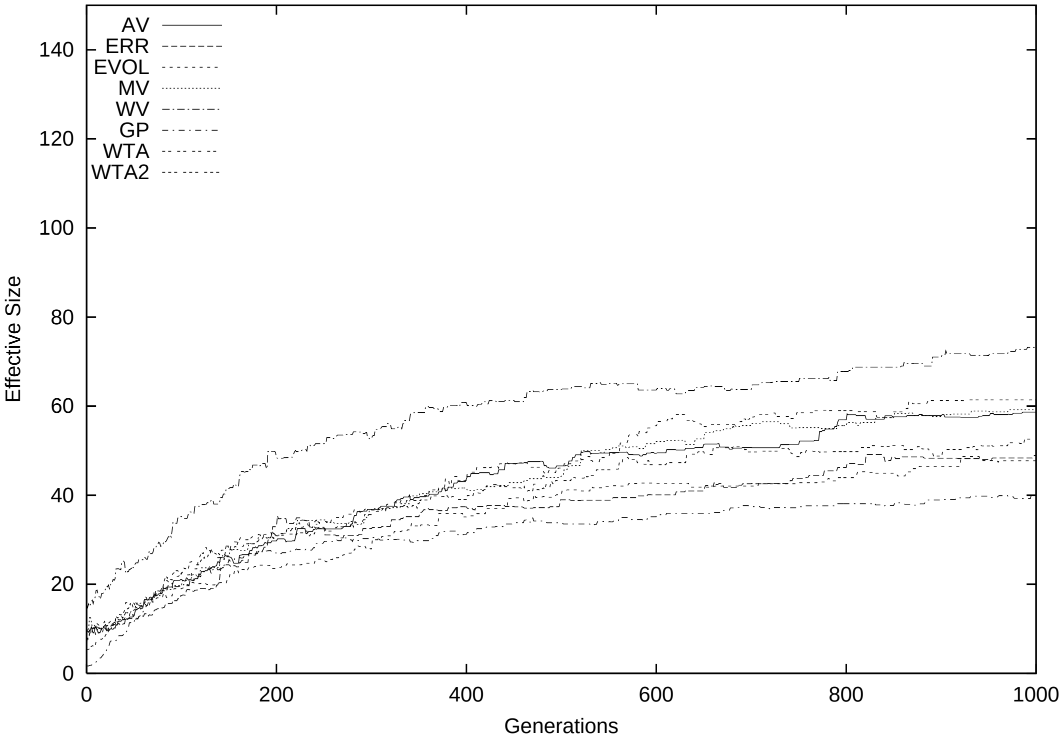

Figure 8: *Heart*: Effective code size of best teams with 4 members and standard GP. Teams are not even 50% bigger than standard individuals on average.

not true is demonstrated in the experiments documented in Table 8 and 9 further below. Members with the best prediction accuracy and the biggest effective length emerge with the *lowest* variation rate.

As a result there must be another explanation than variation speed for the relatively small effective size of teams. We have already seen in the last section that teams perform better than standard individuals after a sufficient number of generations even though single members are changed less frequently. In order to make team solutions more efficient there must be *cooperations* occuring between the members that specialize to solve certain subtasks. These subtasks can be expected to be less difficult than the main problem wherefore the respective subsolutions are most probably less complex in effective size than a full one-program solution.

6.3 Parameter analysis

In this section we analyze the influence of the two most relevant parameters when dealing with the evolution of program teams. Those are the total number of team members (team size) and the number of members that are selected from a team during a genetic operation. Both prediction errors and code sizes are compared for various settings of these parameters.

It would go under the scope of this paper to give a detailed analysis for each team variant and each problem. Instead, we restrict our experiments to the standard team approach (AV). Combination by simple average has the advantage that each member solution has exactly the same influence on the team decision. That makes teams with a single dom-

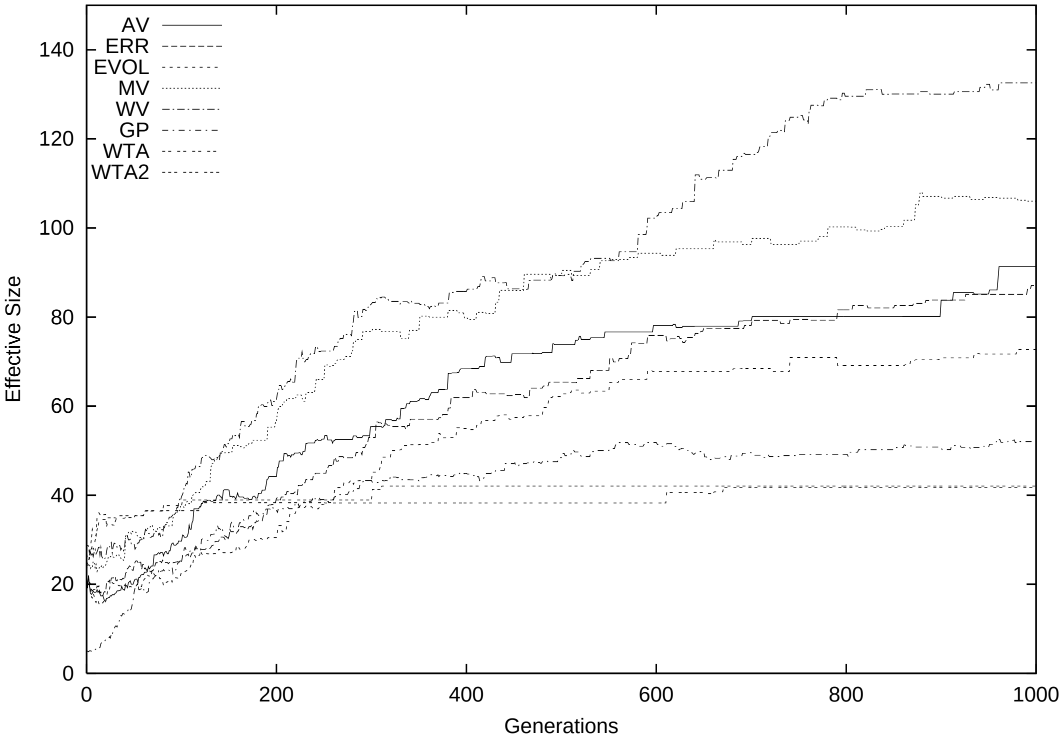

Figure 9: *Two chains*: Effective code size of best teams with 4 members and standard GP. Teams are about twice as big as standard individuals on average.

inating member less likely. Each of the two experiments is documented for one problem only. But similar results had been found with all three test problems.

Number of team members

Each team member is varied by crossover or mutation with a probabilty of 50% in order to guarantee a comparison as fair as possible. Modifying only one member at a time, for instance, would be unfair since then the variation speed of members reduces directly with their number. But, on the other hand, the more members are varied at the same time the more difficult it becomes to make small improvments to the combined team output.

Table 6 compares the classification errors (CE) for the *two chains* problem and different numbers of team members ranging from one (standard GP) to eight. Using teams with more individuals might be rather computationally unacceptable even though only effective instructions are executed in our GP system. Both prediction performance and generalization performance increase with the number of members. But from a team size of about four members significant improvements do not occur here any more.

The correlation between the number of members and the average code size of a member (in number of instructions) is shown in Table 7. The maximum code size of each member is restricted to 128 instructions. The absolute size and the effective size per member decrease until a certain number of team individuals only. Beyond that, both sizes stay almost the same. This corresponds directly to the development in prediction quality from Table 6. Note that the amount of genetic material of the whole team still increases with

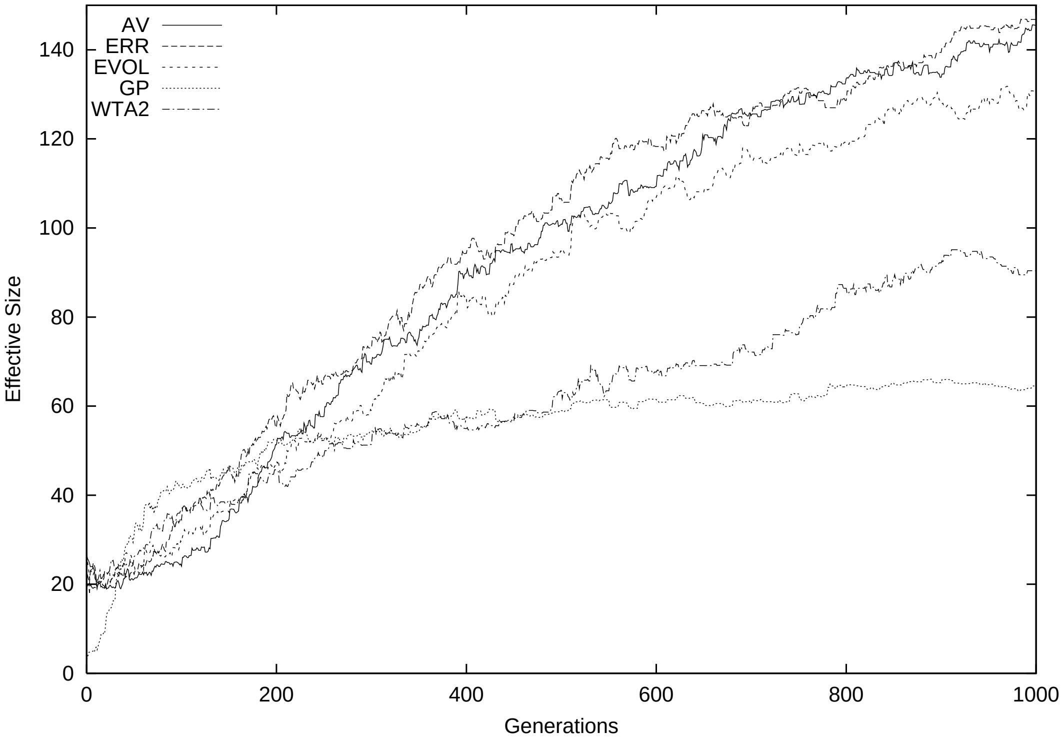

Figure 10: *Three function*: Effective code size of best teams with 4 members and standard GP.

#Members	Training CE (%)	Member CE (%)	Validation CE (%)	Test CE (%)
1	3.72	3.7	5.15	5.73
2	1.47	14.6	2.50	3.47
3	0.89	23.1	1.59	2.64
4	0.37	27.4	0.57	1.72
5	0.36	31.9	0.47	1.88
6	0.38	32.6	0.58	1.76
7	0.33	32.5	0.48	1.78
8	0.39	34.1	0.59	1.83

Table 6: *Two chains*: Classification error (CE) for different number of team members. Half of the team members are varied.

the number of members.

The reason for the reduction in effective member size can be seen in a distribution of the problem task among the team individuals whereby the subtask each member has to fulfill gets smaller and easier. A second indication for that might be the average member error that has been calculated for the full training set here. As shown in Table 6 the error increases respectively. Obviously, beyond a certain number of individuals the task can not be split more efficiently so that some members must fulfill more-or-less the same. As a result, members keep to a certain effective size and prediction quality.

The intron rate is not affected significantly even though genetic operators change more members (always 50%) simultaneously in bigger teams. Only with very few members

#Members	Member Size	Eff. Member Size	Introns (%)
1	128	47	63.1
2	127	38	70.1
3	100	27	73.0
4	95	22	76.8
5	83	19	77.1
6	88	22	75.0
7	81	19	76.5
8	80	20	75.0

Table 7: *Two chains*: Correlation between number of members and average member size (in number of instructions) in teams. Half of the team members are varied.

the rate is lower. But this is due to the maximum size limit that restricts mainly the growth of the intron code. The, otherwise, rather constant rate of non-effective code (and effective code respectively) can be explained by the influence of each member on the team output that decreases with the total number of members—especially if uniform member weights are used. As a result, the intervention of crossover should be almost the same for all configurations (in contrast to Table 8) and higher protection by more introns is not needed. Moreover, this is also an explanation why team errors in Table 6 do not get worse again from a certain number of individuals.

Number of varied members

As stated above best results occur when only a moderate number of team members, i.e. one or two, is varied simultaneously by crossover or mutation. This is demonstrated in Table 8 where the number of varied members ranges from one to a maximum of four while the team size stays fixed. This implies that the effect of crossover becomes the more destructive the more members participate in it. Prediction and generalization performance are found best if only one individual is varied at a time.

#Varied Members	Training MSE	Member MSE	Validation MSE	Test MSE
1	4.1	902.5	3.4	3.7
2	5.4	730.0	4.8	4.9
3	6.5	538.1	5.5	6.3
4	8.3	420.5	7.1	7.6

Table 8: *Three functions*: Mean square error (MSE $\times$ 100) with different numbers of varied members in teams with 4 mebers.

Table 9 demonstrates the relation between the number of varied team members and the code size of teams. Interestingly, the effective code size reduces with the variation strength. Although the variation probability per member is lowest if only one member is varied during a team operation the effective code is biggest. This reflects the results from Table 8 if we conclude that bigger program code reaches a higher prediction accuracy for this problem.

Obviously, the less variation the team members experience the higher becomes their effective length. Some reasons can be found to explain this phenomena:

#Varied Members	Code Size	Eff. Code Size	Introns (%)
1	440	148	66.4
2	424	125	70.5
3	388	113	70.9
4	320	99	69.1

Table 9: *Three functions*: Correlation between number of varied members and code size of teams. Number of team members is 4.

The main reason might be the fact that smaller steps in variation allow more directed improvements of the team programs and the combined (team) error than bigger steps. This is also reflected by the average error of the members that is highest with the lowest level of variation (see Table 8). Higher individual errors might correspond to a higher degree in specialization again as already observed in Section 6.1.

On the other hand, it is easier for smaller (effective) code to survive if the interferences of the variation operators increase. Decreasing effective size is the dominating protection mechanism here. The intron rate is not effected significantly and only slighly higher if more than one member is recombined.

7 Future research

First of all, it is interesting to fix problem classes for which the team approach is suitable in general or for which it cannot produce better results than the standard approach. Linear separability might be a key criterion in this context.

The exchange of information between the individuals of a team might help to evolve a better coordinated behaviour. One possiblity in linear GP is, for instance, to share some calculation variables between team members that together implement a *collective memory*. Values can be assigned to these variables by one individual and used by others that are executed afterwards. Note that with using a shared memory the evaluation order of the team members has to be observed. Another possible form of information sharing is the coevolution of submodules (ADFs) with each team that can be used by all its members in common (*shared submodules*).

Moreover, an implicit form of shared registers could be realized with linear GP if *single* program solutions themselves make multiple predictions in more than one output. These outputs can be combined by using the same methods as proposed for team solutions. If enough registers are provided complementary subsolutions may be computed in more-or-less independent sets of registers within the same program. As a result, the effective code can be expected longer than in solutions with a single output.

Teams offer the possibility for an alternative parallelization approach in genetic programming that is different from distributing subpopulations of individuals to multiple processors. The member programs of a team can be executed in parallel by assigning each member an own processing unit. If all members of the same position index ("member deme") belong to the same unit and interpositional recombination is not applied migration of programs between processing nodes is not necessary. The only communication overhead between the units would be the exchange of team identifier and team outputs.

Further research might be done to investigate the numerous alternatives in more detail that have been given in the text.

8 Conclusion

The team approach has been applied successfully to different prediction problems and found to improve both the training fitness and the generalization performance significantly. For different problem tasks different methods for combining the multiple decisions of the team members turned out to be the most successfull ones. The additional computational overhead of team evolution was found to be small if non-effective instructions are removed from the linear genetic programs before execution. Especially this property makes linear GP interesting for the evolution of program vectors. With linear GP the evolution of teams becomes efficient in solution quality as well as in runtime.

A downside of team solutions might be that they are probably more difficult to analyze than single genetic programs, thus compensating this weakness. However, a combination of subsolutions could be more simple than a one-program solution. We are convinced that team approaches suitable to harness the power of GP.

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 (1998) *Genetic Programming — An Introduction. On the automatic Evolution of Computer Programs and its Application.* dpunkt/Morgan Kaufmann, Heidelberg/San Francisco.

[2] M. Brameier and W. Banzhaf (2000) *A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining.* IEEE Transactions on Evolutionary Computation, in press.

[3] L.K. Hansen and P. Salamon (1990) *Neural network ensembles.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(10):993–1001.

[4] T. Haynes, S. Sen, D. Schoenefeld, and R. Wainwright (1995) *Evolving a team.* In *Working Notes for the AAAI Symposium on Genetic Programming*, MIT Press, Cambridge, MA.

[5] T. Haynes and S. Sen (1997) *Crossover operators for evolving a team.* In In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, 162–167, Morgan Kaufmann, San Francisco, CA.

[6] J. Koza (1992) *Genetic Programming.* MIT Press, Cambridge, MA.

[7] A. Krogh and J. Vedelsby (1995) *Neural network ensembles, cross validation, and active learning.* In G. Tesauro, D.S. Touretzky and T.K. Leen, editors, *Advances in Neural Information Processing Systems*, 7:231–238, MIT Press, Cambridge, MA.

[8] S. Luke and L. Spector (1996) *Evolving teamwork and coordination with genetic programming.* In J.R. Koza, D.E. Goldberg, David B. Fogel, and Rick L. Riolo (eds.) *Genetic Programming 1996: Proceedings of the First Annual Conference*, 150–156, MIT Press, Cambridge, MA.

[9] P. Nordin (1994) *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.

[10] M.P. Perrone and L.N. Cooper (1993) *When networks disagree: Ensemble methods for neural networks.* In R.J. Mammone, editor, *Neural Network for Speech and Image Processing*, 126–142, Chapman-Hall, London, 1993.

[11] M. Riedmiller and H. Braun (1993) *A direct adaptive method for faster backpropagation learning: the RPROP algorithm.* In *Proceedings of the IEEE International Conference on Neural Networks*, 586–591, San Francisco, CA.

[12] H.-P. Schwefel (1995) *Evolution and Optimum Seeking.* Wiley, New York.

[13] R.L. Somorjai, A.E. Nikulin, N. Pizzi, D. Jackson, G. Scarth, B. Dolenko, H. Gordon, P. Russell, C.L. Lean, L. Delbridge, C.E. Mountford and I.C.P. Smith (1995) *Computerized Consensus Diagnosis — A Classification Strategy for the Robust Analysis of MR Spectra. 1. Application to H-1 Spectra of Thyroid Neoplasma.* Magnetic Resonance in Medicine, 33:257–263.

[14] T. Soule (1999) *Voting Teams: A cooperative approach to non-typical problems using genetic programming.* In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela and R.E. Smith *Proceedings of the International Conference on Genetic and Evolutionary Computation (GECCO99)*, 916–922, Morgan Kaufmann, San Francisco, CA.

[15] W.A. Tackett (1994) *Recombination, Selection and the Genetic Construction of computer programs.* Ph.D. thesis, University of Southern California, Department of Electrical Engineering Systems.

[16] D.H. Wolpert (1992) *Stacked regression.* Neural Networks, 5(2):241–259.

[17] X. Yao and Y. Liu (1998) *Making use of population information in evolutionary artificial neural networks.* IEEE Transactions on Systems, Man and Cybernetics, 28B(3):417–425.

[18] B.-T. Zhang and J.-G. Joung (1996) *Enhancing Robustness of Genetic Programming at the Species Level.* In J.R. Koza, D.E. Goldberg, David B. Fogel, and Rick L. Riolo (eds.) *Genetic Programming 1996: Proceedings of the First Annual Conference*, 336–342, MIT Press, Cambridge, MA.