Coevolution in Cartesian Genetic Programming

Michaela Šikulová and Lukáš Sekanina

Brno University of Technology, Faculty of Information Technology,
IT4Innovations Centre of Excellence
Božetěchova 2, 612 66 Brno, Czech Republic
{isikulova,sekanina}@fit.vutbr.cz

Abstract. Cartesian genetic programming (CGP) is a branch of genetic programming which has been utilized in various applications. This paper proposes to introduce coevolution to CGP in order to accelerate the task of symbolic regression. In particular, fitness predictors which are small subsets of the training set are coevolved with CGP programs. It is shown using five symbolic regression problems that the (median) execution time can be reduced 2–5 times in comparison with the standard CGP.

Keywords: Cartesian genetic programming, Coevolution, Symbolic regression.

1 Introduction

Cartesian genetic programming (CGP) is a variant of *genetic programming* (GP) that uses a specific encoding in the form of directed acyclic graph and a mutation-based search [11, 10]. CGP has been successfully employed in many traditional application domains of genetic programming such as symbolic regression. It has, however, been predominantly applied in evolutionary design and optimization of logic networks.

The *fitness evaluation* is typically the most time consuming part of CGP in these applications. In the case of digital circuit evolution, it is necessary to verify whether a candidate n-input circuit generates correct responses for all possible input combinations (i.e., 2^n assignments). It was shown that testing just a subset of 2^n test vectors does not lead to correctly working circuits [6, 9]. Recent work has indicated that this problem can partially be eliminated in real-world applications by applying formal verification techniques [15].

In the case of *symbolic regression*, k fitness cases are evaluated during one fitness function call, where k typically goes from hundreds to ten thousands. The time needed for evaluating a single fitness case depends on a particular application. Usually, the goal of GP system design and GP parameters' tuning is to obtain a solution with predefined accuracy and robustness using a minimum number of evaluated fitness cases or fitness function calls. In order to reduce the evaluation time, *fitness approximation* techniques have been employed. One of them is *fitness modeling* which uses fitness models with different degrees of sophistication to reduce the fitness calculation time [7]. It is assumed that the fitness model can be constructed and updated in a reasonable time. The motivation for fitness modeling can be seen not only in reducing the complexity of fitness evaluation but also in avoiding the explicit fitness definitions, coping with noisy data, smoothing the fitness landscape and promoting diversity [14]. Fitness modeling is typically based on machine learning methods, subsampling of training data or partial evaluation.

Fitness prediction is a low cost adaptive procedure utilized to replace fitness evaluation. A framework for reducing the computation requirements of symbolic regression using fitness predictors has been introduced for standard genetic programming by Schmidt and Lipson [14]. Their method combines fitness prediction with coevolution to eliminate disadvantages of a classic fitness modeling, in particular the effort needed to train a fitness model and adapt the level of approximation and accuracy. The method utilizes a coevolutionary algorithm which exploits the fact that one individual can influence the relative fitness ranking between two other individuals in the same or a separate population [5]. Coevolving the training samples as the method of fitness modeling in GP has been studied in many aplication domains [2, 3, 4, 8] and in the symbolic regression problem [1, 12, 13, 14].

The goal of this paper is to introduce coevolving fitness predictors to CGP and show that by using them, the execution time of symbolic regression can significantly be reduced. The proposed coevolution of CGP programs and fitness predictors in the symbolic regression problem uses two populations evolving concurrently. Properties of individuals in the population of candidate programs change in response to properties of individuals in the population of fitness predictors and vice versa. It is expected that CGP which has been accelerated using coevolution will be implemented on a chip in our future work. Hence the proposed approach will also be useful for evolvable hardware purposes. Note that hardware implementation of CGP is straightforward which is not the case of tree-based GP [10].

The proposed coevolutionary CGP method is compared with a standard CGP on five symbolic regression problems. A brief comparison of CGP and tree-based GP is also performed on selected benchmark problems.

The rest of the paper is organized as follows. Section 2 introduces Cartesian genetic programming and its application to the symbolic regression problem. In Section 3, a new coevolutionary approach to CGP is presented. Section 4 compares the proposed coevolutionary algorithm with the standard CGP on five test problems. Experimental results are discussed in Section 5. Finally, conclusions are given in Section 6.

2 Cartesian Genetic Programming

In standard CGP (chapter 2 of [10]), a candidate program is modeled as an array of n_c (columns) $\times n_r$ (rows) of programmable elements (nodes). The number of primary inputs, n_i, and outputs, n_o, of the program is fixed. Each node input can be connected either to the output of a node placed in previous l columns or

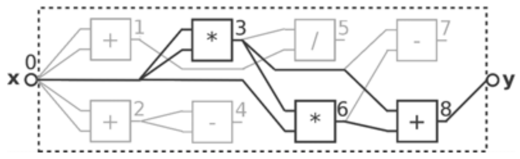


Fig. 1. A candidate program in CGP, where $l = 4$, $n_c = 4$, $n_r = 2$, $n_i = 1$, $n_o = 1$, $n_a = 2$, $\Gamma = \{+\ (1),\ -\ (2),\ *\ (3),\ /\ (4)\}$ and chromosome is: $0, 0, 1$; $0, 0, 1$; $0, 0, 3$; $2, 2, 2$; $3, 1, 4$; $3, 0, 3$; $3, 6, 2$; $3, 6, 1$; 8

to one of the program inputs. The l-back parameter, in fact, defines the level of connectivity and thus reduces/extends the search space. Feedback is not allowed. Each node is programmed to perform one of n_a-input functions defined in the set Γ. Each node is encoded using $n_a + 1$ integers where values $1 \ldots n_a$ are the indexes of the input connections and the last value is the function code. Every individual is encoded using $n_c \cdot n_r \cdot (n_a + 1) + n_o$ integers. Figure 1 shows an example of a candidate circuit. While the primary inputs are numbered $0 \ldots n_i - 1$ the nodes are indexed $n_i \ldots n_c n_r + n_i - 1$.

A simple (1+λ) evolutionary algorithm is used as a search mechanism. It means that CGP operates with the population of $1 + \lambda$ individuals (typically, λ is between 1 and 20). The initial population is constructed either randomly or by a heuristic procedure. Every new population consists of the best individual of the previous population (so-called parent) and its λ offspring. However, as a new parent an offspring is always chosen if it is equally as fit or has better fitness than the parent. The offspring individuals are created using a point mutation operator which modifies h randomly selected genes of the chromosome, where h is the user-defined value. The algorithm is terminated when the maximum number of generations is exhausted or a sufficiently working solution is obtained.

For symbolic regression problems, the goal of evolution is usually to minimize the *mean absolute error* of a candidate program response y and target response t. The fitness function (taking candidate program s as its argument) is then defined

$$\mathrm{f}\,(s) = \frac{1}{k} \sum_{j=1}^{k} |y(j) - t(j)| \tag{1}$$

where k is the number of fitness cases. Alternatively, the *number of hits* can represent the fitness value. The number of hits is defined

$$\mathrm{f}\,(s) = \sum_{j=1}^{k} \mathrm{g}\,(y\,(j))\,, \text{ where} \tag{2}$$

$$\mathrm{g}\,(y\,(j)) = \begin{cases} 0 \text{ if } |y\,(j) - t\,(j)| \geq \varepsilon \\ 1 \text{ if } |y\,(j) - t\,(j)| < \varepsilon \end{cases} \tag{3}$$

and ε is a user-defined acceptable error.

3 Coevolution of Fitness Predictors in CGP

The aim of coevolving fitness predictors and programs is to allow both solutions (programs) and fitness predictors to enhance each other automatically until a satisfactory problem solution is found. We propose to adopt Schmidt's and Lipson's approach [14] using CGP for the task of symbolic regression. Figure 2 shows the overall scheme of the proposed method. There are two concurrently working populations: (1) candidate programs (syntactic expressions) evolving using CGP and (2) fitness predictors evolving using a genetic algorithm.

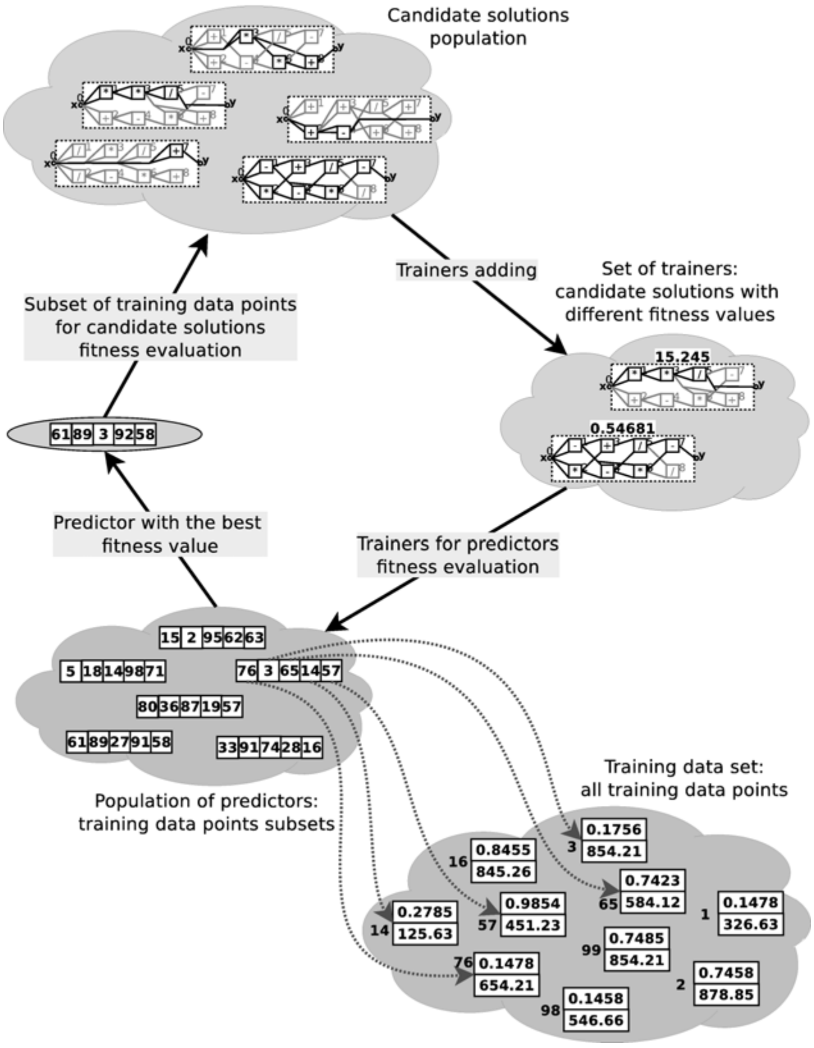

Candidate solutions population
Trainers adding
Set of trainers: candidate solutions with different fitness values
15.245
0.54681
Subset of training data points for candidate solutions fitness evaluation
61 89 3 92 58
Predictor with the best fitness value
Trainers for predictors fitness evaluation
15 2 95 62 63
5 18 14 98 71
76 3 65 14 57
80 36 87 19 57
61 89 27 91 58
33 91 74 28 16
Population of predictors: training data points subsets
Training data set: all training data points
3 0.1756 854.21
16 0.8455 845.26
65 0.7423 584.12
1 0.1478 326.63
14 0.2785 125.63
57 0.9854 451.23
99 0.7485 854.21
76 0.1478 654.21
2 0.7458 878.85
98 0.1458 546.66

Fig. 2. Coevolution of candidate programs and fitness predictors

3.1 Population of Candidate Programs

Evolution of candidate programs is based on principles of CGP as introduced in Section 2. The fitness function for CGP is defined as the relative number of hits. There are, in fact, two fitness functions for candidate program s. While the exact fitness function $f_{exact}(s)$ utilizes the complete training set, the predicted fitness function $f_{predicted}(s)$ employs only selected fitness cases. Formally,

$$\mathrm{f}_{\mathrm{exact}}(s) = \frac{1}{k}\sum_{j=1}^{k} g(y(j)) \tag{4}$$

$$\mathrm{f}_{\mathrm{predicted}}(s) = \frac{1}{m}\sum_{j=1}^{m} g(y(j)) \tag{5}$$

where k is the number of data points in the training set and m is the number of data points in the fitness predictor (i.e., m is the size of a subset of the training set).

3.2 Set of Trainers

The set of trainers which contains several candidate programs is used to evaluate fitness predictors. The proposed implementation differs from [14] in the organization and update strategy. In particular, the set of trainers is divided into two parts. The first part is periodically updated from the population of candidate programs (the best-scored candidate program is sent to the trainers set if its fitness value differs from the best-scored candidate program in the previous generation) and the second part is periodically and randomly generated to ensure genetic diversity of the set of trainers. The size of trainers set is kept constant during evolution. For every new selected or generated trainer, the exact fitness is calculated and the new trainer replaces the oldest one in the corresponding part of the trainers set.

3.3 Population of Fitness Predictors

Fitness predictor is a small subset of training data. An optimal fitness predictor is sought using a simple genetic algorithm (GA) which operates with a population of fitness predictors. Every predictor is encoded as a constant-size array of pointers to elements in the training data. In addition to one-point crossover and mutation, a randomly selected predictor replacing the worst-scored predictor in each generation has been introduced as a new genetic operator of GA. The fitness value of predictor p is calculated using the *mean absolute error* of the exact and predicted fitness values of trainers

$$\mathrm{f}(p) = \frac{1}{u}\sum_{i=1}^{u} \left|\mathrm{f}_{\mathrm{exact}}(i) - \mathrm{f}_{\mathrm{predicted}}(i)\right| \tag{6}$$

where u is the number of candidate programs in the trainers set. The predictor with the best fitness value is used to predict the fitness of candidate programs in the population of candidate programs.

3.4 Implementation

Two threads are used. The first one is responsible for candidate programs evolution using CGP. The second thread performs evolution of fitness predictors using a simple genetic algorithm. The coevolution is implemented as follows. The first thread randomly initializes both populations and also randomly creates the first individuals in the set of trainers. After the second thread is activated both populations are evaluated.

CGP evolution loop begins with loading the fittest training data sample from population of fitness predictors. This is performed periodically, but not in every iteration due to a slower rate of fitness predictors evolution. This results in a lower computational effort. It is not necessary to run the fitness predictor evolution as fast as the candidate program evolution, because fast changes of the best rated fitness predictor do not contribute to convergence.

The next step involves calculating the predicted fitness of all individuals in the candidate program population. The best rated individual is then selected and its number of hits is checked. If the predicted fitness value is not in the interval of acceptable fitness values, CGP will create a new population, eventually new trainer will be selected or generated. If predicted fitness value falls into the interval of acceptable fitness values, the exact fitness of candidate program is evaluated. If the exact fitness falls into the interval of acceptable fitness values, a solution is found, and coevolution is terminated. Otherwise, the update of the best rated fitness predictor is signaled and the coevolution has to continue.

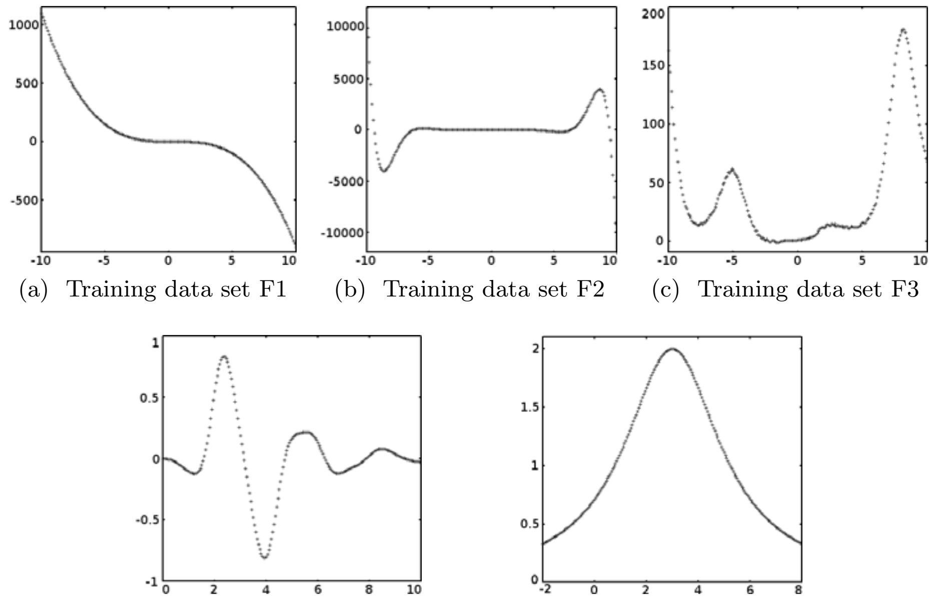
The second thread performs the evolution of fitness predictors. The fitness values of all fitness predictors are evaluated using trainers. The best rated predictor is selected and stored to shared memory. The next step involves creating of a new generation of fitness predictors by means of GA operators. Subsequently, the GA waits for a signal from the first thread. After receiving the signal, the GA loop continues with the next iteration, or if a solution is discovered, GA is terminated.

4 Results

This section presents benchmark problems, experimental setup, experimental evaluation of the proposed coevolutionary approach to CGP and its comparison with standard CGP.

4.1 Benchmark Problems

Five test functions (F1 – F5) were selected as data point sources for evaluation of the proposed method:



(a) Training data set F1 (b) Training data set F2 (c) Training data set F3

(d) Training data set F4 (e) Training data set F5

Fig. 3. Training data sets: x values on horizontal axes, $f(x)$ values on vertical axis

$$F1 : f(x) = x^2 - x^3, x \in \langle -10, 10\rangle \tag{7}$$
$$F2 : f(x) = e^{|x|}\sin(x), x \in \langle -10, 10\rangle \tag{8}$$
$$F3 : f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right), x \in \langle -10, 10\rangle \tag{9}$$
$$F4 : f(x) = e^{-x}x^3 \sin(x)\cos(x)\left(\sin^2(x)\cos(x) - 1\right), x \in \langle 0, 10\rangle \tag{10}$$
$$F5 : f(x) = \frac{10}{(x-3)^2 + 5}, x \in \langle -2, 8\rangle \tag{11}$$

In order to form a training set, 200 equidistant distributed samples were taken from each function (see Fig. 3). Functions F1, F2 and F3 are taken from [14] and functions F4 and F5 from [16]. Table 1 shows acceptable errors and the acceptable number of hits.

4.2 Experimental Setup

Table 1 shows that various settings of the components involved in the proposed coevolutionary method have been tested. Over 100,000 independent runs were performed to find the most advantageous setting which is presented in the right-most column and which is later used in all reported experiments.

Programs are evolved using CGP with the following setup: $l = n_c$, $n_r = 1$, $n_i = 1$, $n_o = 1$, every node has two inputs (i_1, i_2) and $\Gamma = \{\ i_1 + i_2,\ i_1 - i_2,\ i_1 \cdot i_2,\ \frac{i_1}{i_2},\ \sin(i_1),\ \cos(i_1),\ e^{i_1},\ \log(i_1)\}$. Table 1 shows various setting of n_c, λ and h considered during parameters tuning.

Fitness predictors evolution is conducted using a simple GA. Table 1 shows numerous setting of the chromosome length, population size and genetic operators.

Other parameters of coevolution, such as the size of trainers set, frequency of trainers substitutions and predictors evolution deceleration are also given in Table 1.

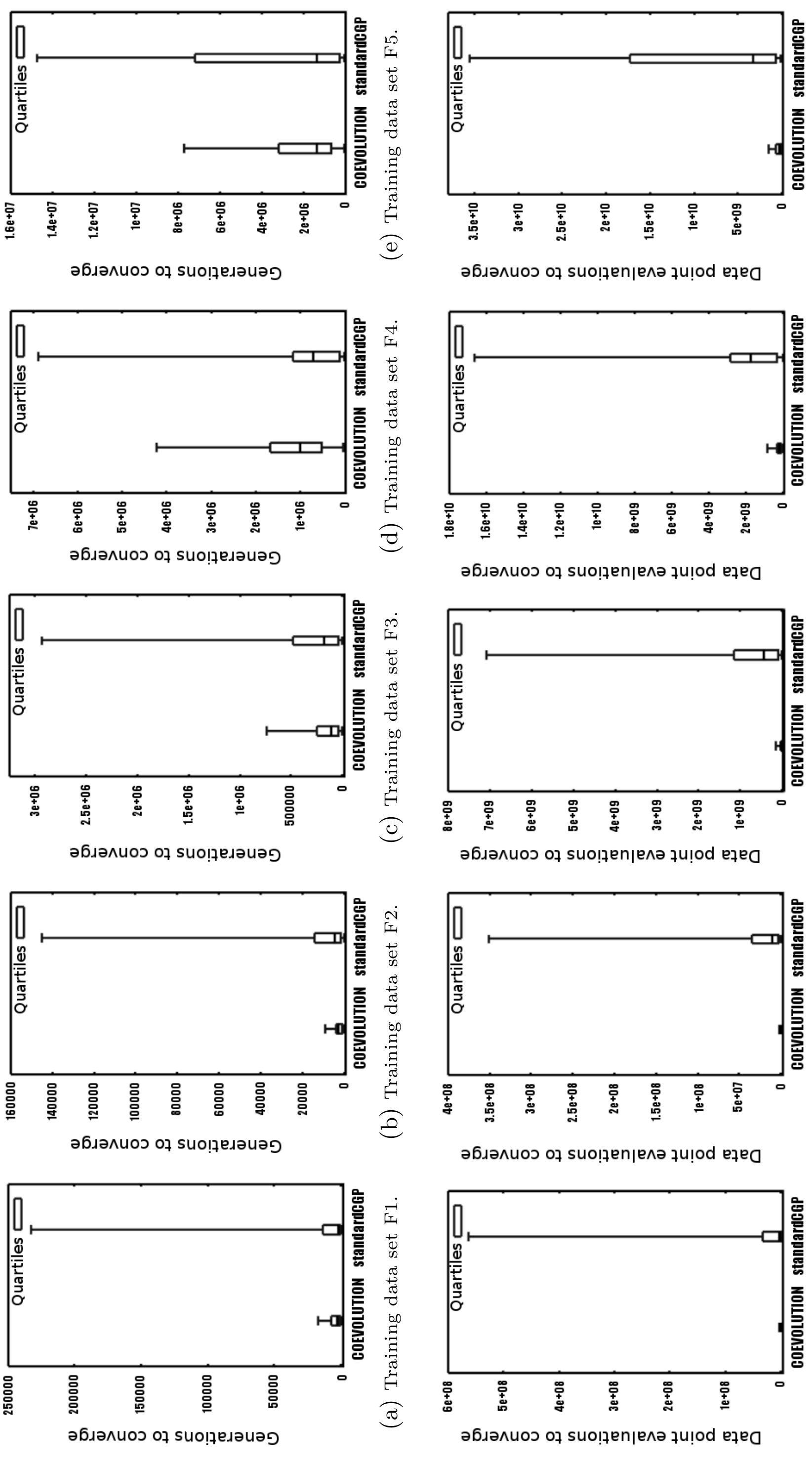
4.3 Comparison of Coevolving CGP with Standard CGP

The proposed coevolutionary algorithm was compared with standard CGP using test functions F1-F5. Parameters of both algorithms were chosen according to Table 1 and 50 independent runs were performed. Table 2 gives the resulting success rate (the number of runs giving a solution with predefined quality), the number of generations, the number of data point evaluations and time to converge calculated as median out of 50 independent runs. Figure 4 shows quartile graphs of the number of generations and data point evaluations for all five training data sets.

Figure 5 shows the progress of the best fitness value during a typical run on the F2 data set. It can be seen that while the progress is monotonic for the standard CGP, the coevolutionary algorithm produces very dynamic changes ending with a significant increase of the best fitness value at the end of evolution. The changes of the best fitness value are caused by updating of the best fitness predictor.

Table 1. Experimental setup

	Parameter	Tested values	Selected values
CGP	Chromosome length n_c	16, 24, 32, 64, 96, 128	32
	Population size λ	4, 8, 12, 16, 20	12
	Number of mutations h per individual	1-4, 1-8, 1-12, 1-16	1-8
Trainers, Coevolution	All trainers substitution	1 per 500 generations of CGP	500
	Trainers set size	8, 12, 16, 24, 32	8
	Predictor evolution deceleration	1 per 10, 25, 50, 100, 150, 200 generations of CGP	100
GA-Predictors	Chromosome length	4, 8, 12, 16, 24, 32, 64	12
	Population size	8, 12, 16, 24, 32, 48, 64, 96, 128	32
	Offspring creation	2-tournament selection, single point crossover	
	Mutation probability	0.2	0.2
Test functions	Acceptable error of data point	F1, F2: 0.5; F3: 1.5; F4, F5: 0.025	
	Acceptable number of hits	97 %	97 %



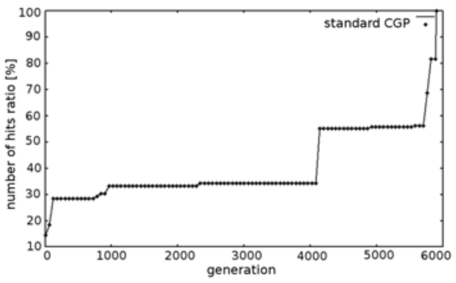
(a) Training data set F1. (b) Training data set F2. (c) Training data set F3. (d) Training data set F4. (e) Training data set F5.

(f) Training data set F1. (g) Training data set F2. (h) Training data set F3. (i) Training data set F4. (j) Training data set F5.

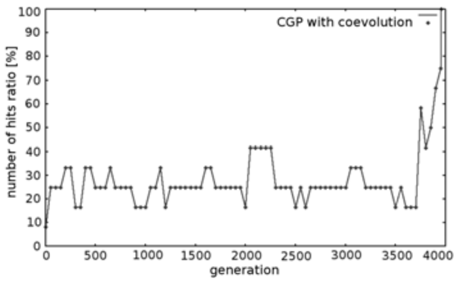
Fig. 4. Comparison of standard CGP and CGP with coevolution: Number of generations and number of data point evaluations to converge

Table 2. Comparison of standard CGP and CGP with coevolution for five training data sets

		F1	F2	F3	F4	F5
Success rate	stand. CGP	100 %	100 %	78 %	80 %	24 %
	coevolution	100 %	100 %	100 %	100 %	100 %
Generations to converge (median)	stand. CGP	$1.11 \cdot 10^3$	$4.46 \cdot 10^3$	$1.76 \cdot 10^5$	$7.15 \cdot 10^5$	$1.36 \cdot 10^6$
	coevolution	$2.62 \cdot 10^3$	$2.53 \cdot 10^3$	$1.10 \cdot 10^5$	$1.00 \cdot 10^6$	$1.34 \cdot 10^6$
Data point evaluations to converge (median)	stand. CGP	$2.68 \cdot 10^6$	$1.08 \cdot 10^7$	$4.24 \cdot 10^8$	$1.72 \cdot 10^9$	$3.28 \cdot 10^9$
	coevolution	$5.20 \cdot 10^5$	$5.01 \cdot 10^5$	$2.19 \cdot 10^7$	$2.00 \cdot 10^8$	$2.67 \cdot 10^8$
Time to converge (median) [s]	stand. CGP	35.4611	55.1476	98.8585	44.0388	104.6826
	coevolution	17.4588	21.1178	18.1257	17.1079	20.4529



(a) Standard CGP.



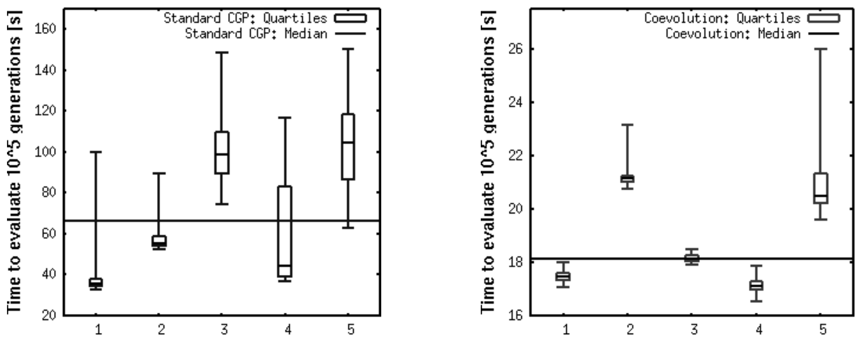
(b) CGP with coevolution.

Fig. 5. Progress of the best fitness value during a typical run for the F2 data set

5 Discussion

It can be seen from Table 2 that the proposed coevolutionary method has reached a satisfactory solution using much fewer data point evaluations than the standard CGP. The speedup measured on the Intel® Core$^{\text{TM}}$ i5-2500 machine is between 2.03 (F1) and 5.45 (F3). Detailed analysis of execution time is shown in Fig. 6 where quartile graphs are given for 50 independent runs. However, it should be pointed out that the standard CGP evaluates 200 fitness cases in every fitness function call while the coevolutionary algorithm evaluates only 12 fitness cases. The number of generations is similar for both methods. A notable observation is that while the standard CGP was not able to produce a satisfactory solution in 23.6% runs the proposed method reached a satisfactory solution in all cases. Moreover, the results generated by multiple runs of coevolutionary CGP are more stable than those produced by the standard CGP.

There is only one data set (F2) and corresponding results of the tree-based coevolutionary GP [14] which can serve for a direct comparison with our CGP-based coevolution. While tree-based GP requires $1 \cdot 10^3$ generations and $7 \cdot 10^6$



(a) Standard CGP.

(b) CGP using coevolution.

Fig. 6. Time of evolution

data point evaluations to converge, the proposed CGP-based approach requires $3 \cdot 10^3$ generations and only $5 \cdot 10^5$ data point evaluations to converge. The proposed method seems to be competitive with [14].

6 Conclusions

Symbolic regression has not been considered as a typical application domain for CGP. We have shown in this paper that CGP equipped with coevolution of fitness predictors can significantly be accelerated in this particular application. The speedup obtained for five test problems is $2.03 - 5.45$ over the standard CGP. Results are also very competitive with the tree-based GP.

Our future work will be devoted to utilization of the proposed coevolutionary algorithm in other applications domains where the standard CGP has been successful so far. Another goal will be to implement the coevolutionary CGP on a chip and use it in a real-world application.

Acknowledgments. This work was supported by the Czech science foundation project P103/10/1517, the research programme MSM 0021630528, the BUT project FIT-S-11-1 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

References

[1] Dolin, B., Bennett III, F.H., Reiffel, G.: Co-evolving an effective fitness sample: Experiments in symbolic regression and distributed robot control. In: Proc. of the 2002 ACM Symp. on Applied Computing, pp. 553–559. ACM, New York (2002)

[2] Dolinsky, J.U., Jenkinson, I.D., Colquhoun, G.J.: Aplication of genetic programming to the calibration of industrial robots. Computers in Industry 58(3), 255–264 (2007)

[3] Gagné, C., Parizeau, M.: Co-evolution of nearest neighbor classifiers. International Journal of Pattern Recognition and Artificial Inteligence 21(5), 921–946 (2007)

[4] Harrison, M.L., Foster, J.A.: Co-evolving Faults to Improve the Fault Tolerance of Sorting Networks. In: Keijzer, M., O'Reilly, U.-M., Lucas, S., Costa, E., Soule, T. (eds.) EuroGP 2004. LNCS, vol. 3003, pp. 57–66. Springer, Heidelberg (2004)

[5] Hillis, W.D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D 42(1), 228–234 (1990)

[6] Imamura, K., Foster, J.A., Krings, A.W.: The Test Vector Problem and Limitations to Evolving Digital Circuits. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, pp. 75–79. IEEE Computer Society (2000)

[7] Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Computing Journal 9(1), 3–12 (2005)

[8] Mendes, R.R.F., de Voznika, F.B., Freitas, A.A., Nievola, J.C.: Discovering Fuzzy Classification Rules with Genetic Programming and Co-evolution. In: Siebes, A., De Raedt, L. (eds.) PKDD 2001. LNCS (LNAI), vol. 2168, pp. 314–325. Springer, Heidelberg (2001)

[9] Miller, J.F., Thomson, P.: Aspects of Digital Evolution: Geometry and Learning. In: Sipper, M., Mange, D., Pérez-Uribe, A. (eds.) ICES 1998. LNCS, vol. 1478, pp. 25–35. Springer, Heidelberg (1998)

[10] Miller, J.F.: Cartesian Genetic Programming. Springer, Heidelberg (2011)

[11] Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)

[12] Pagie, L., Hogeweg, P.: Evolutionary consequences of coevolving targets. Evolutionary Computation 5(4), 401–418 (1997)

[13] Schmidt, M., Lipson, H.: Co-evolving fitness predictors for accelerating and reducing evaluations. In: Genetic Prog. Theory and Practice IV, vol. 5, pp. 113–130 (2006)

[14] Schmidt, M.D., Lipson, H.: Coevolution of Fitness Predictors. IEEE Transactions on Evolutionary Computation 12(6), 736–749 (2008)

[15] Vasicek, Z., Sekanina, L.: Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genetic Programming and Evolvable Machines 12(3), 305–327 (2011)

[16] Vladislavleva, E.: Symbolic Regression: Toy Problems for Symbolic Regression (2009-2010), `http://www.vanillamodeling.com/toyproblems.html`