BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

PHD THESIS

Brno, 2017 Ing. Michaela Drahošová

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

COEVOLUTION OF FITNESS PREDICTORS IN CARTESIAN GENETIC PROGRAMMING

KOEVOLUCE PREDIKTORŮ FITNESS V KARTÉZSKÉM GENETICKÉM PROGRAMOVÁNÍ

PHD THESIS
DIZERTAČNÍ PRÁCE

AUTHOR — **Ing. MICHAELA DRAHOŠOVÁ**
AUTOR PRÁCE

SUPERVISOR — **Prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2017

Abstract

Cartesian genetic programming (CGP) is an evolutionary based machine learning method which can automatically design computer programs or digital circuits. CGP has been successfully applied in a number of challenging real-world problem domains. However, the computational power that the design based on CGP needs for obtaining innovative results is enormous for most applications. In CGP, every candidate program is executed to dermine a fitness value, representing the degree to which it solves the problem. Typically, the most time consuming part of CGP is the fitness evaluation. This thesis proposes to introduce coevolution of fitness predictors to CGP in order to accelerate the evolutionary design performed by CGP. Fitness predictors are small subsets of the training data, which are used to estimate candidate program fitness instead of performing an expensive objective fitness evaluation. Coevolution of fitness predictors is an optimization method of the fitness modeling that reduces the fitness evaluation cost and frequency, while maintaining the evolutionary process. In this thesis, the coevolutionary algorithm is adapted for CGP and three approaches to fitness predictor encoding are introduced and examined. The proposed approach is evaluated using five symbolic regression benchmarks and in the image filter design problem. The method enabled us to significantly reduce the time of evolutionary design for considered class of problems.

Keywords

Evolutionary design, cartesian genetic programming, coevolutionary algorithms, fitness prediction.

Citation

Michaela Drahošová. *Coevolution of Fitness Predicotrs in Cartesian Genetic Programming*, PhD thesis, Department of Computer Systems, Faculty of Information Technology, Brno University of Technology, Brno, CZ, 2017

Abstrakt

Kartézské genetické programování (CGP) je evolucí inspirovaná metoda strojového učení, která je primárně určená pro automatizovaný návrh programů a číslicových obvodů. CGP je úspěšné v řešení mnoha úloh z reálného světa. Avšak k nalezení inovativních řešení obvykle potřebuje značný výpočetní výkon. Každý kandidátní program navržený pomocí CGP musí být spuštěn, aby se zjistilo, do jaké míry tento program řeší zadaný problém, a mohla mu být přiřazena fitness hodnota. Právě vyhodnocení fitness bývá výpočetně nejnáročnější částí návrhu pomocí CGP. Tato práce se zabývá využitím koevoluce prediktorů fitness v CGP za účelem zrychlení procesu evolučního návrhu prováděného pomocí CGP. Prediktor fitness je malá podmnožina trénovacích dat používaná pro rychlý odhad fitness hodnoty namísto náročného vyhodnocení objektivní fitness hodnoty. Koevoluce prediktorů fitness je optimalizační metoda modelování fitness, která snižuje náročnost a frekvenci výpočtu fitness. V této práci je koevoluční algoritmus přizpůsoben pro CGP a jsou představeny a zkoumány tři přístupy k zakódování prediktorů fitness. Představená metoda je experimentálně vyhodnocena v pěti úlohách symbolické regrese a v úloze návrhu obrazových filtrů. Výsledky experimentů ukazují, že pomocí této metody lze významně snížit výpočetní čas, který CGP potřebuje pro řešení zkoumané třídy úloh.

Klíčová slova

Evoluční návrh, kartézské genetické programování, koevoluční algoritmy, predikce fitness.

Citace

Michaela Drahošová. *Koevoluce prediktorů fitness v kartézském genetickém programování*, Dizertační práce, Ústav počítačových systémů, Fakulta informačních technologií, Vysoké učení technické v Brně, Brno, CZ, 2017

Coevolution of Fitness Predicotrs in Cartesian Genetic Programming

Prohlášení

Prohlašuji, že jsem tuto práci vypracovala samostatně pod vedením Prof. Ing. Lukáše Sekaniny, Ph.D. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Michaela Drahošová
26. června 2017

Poděkování

Děkuji svému školiteli profesoru Lukáši Sekaninovi za odborné vedení a také za cenné rady, náměty a nápady, které mi poskytl při řešení této disertační práce.

Chci poděkovat i kolegům z UPSY za odborné konzultace, podporu a možnost s nimi spolupracovat. Děkuji všem, kteří mi poskytli odbornou pomoc při vypracování této práce.

Děkuji svému manželovi Vojtovi, který mě po celou dobu doktorského studia bezmezně podporoval, povzbuzoval a umožnil mi tuto práci dokončit.

Contents

Chapter 1

Introduction

The computer aided design is a well established approach to engineering design. Many software tools are currently available and simplify designers job. The evolutionary design builds on these software tools by taking over a part of a design process. It allows designers to improve the performance of their designs and to explore numerous alternative solutions to problems automatically [4].

The automatic evolution of computer programs is known as genetic programming (GP). In some sense, GP represents an attempt to accomplish one of the most ambitious goals of computer science: being able to specify what one wants a program to do, but not how, and have the computer figure out an implementation.

In 1997, Julian Miller introduced a special form of GP, nowadays called Cartesian genetic programming (CGP) [41], which grew from a method of evolving digital circuits. Over the years, CGP has been successfully applied to a number of challenging real-world problem domains. However, the computational power that evolutionary design based on CGP (as well as on standard GP) needs for obtaining innovative results is enormous for many applications.

For most real-world applications of GP it is a well known fact that fitness evaluation is by far the most time consuming operation. For example, for 36 tasks solved using Koza's genetic programming, the average population size was 3,350,000 individuals, 128.7 generations were produced in average and the average time to reaching a solution was 81.9 hours [36].

The problem with expensive evaluations also holds for the evolutionary design performed by CGP. For example, consider the image filter design problem. The most time consuming procedure is the fitness calculation where tens of thousands of pixels of the training set (i.e. fitness cases) have to be evaluated in order to obtain one fitness value. A single run is typically finished after 200 thousands candidate filter evaluations [60].

Many researches are concerned with how to establish criteria that can help them to reduce the time spent in the fitness evaluation phase as much as possible without compromising results in terms of quality and, possibly, generalization capability.

The training set used for fitness evaluations is often a small sample of the entire problem domain space. One is thus confronted with two problems: how to select a sufficient number of fitness cases and how to choose such fitness cases that are effective in forcing the search process towards a desired solution.

The fitness evaluation time can be reduced by using well prepared training sets, fitness estimation techniques or, in some cases, formal verification algorithms (see an overview in

[71]). Moreover, various domain-specific hardware accelerators, including GPU-based or FPGA-based machines [74, 70, 23], are often employed.

One of fitness estimation techniques is a *fitness prediction*, which 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 GP by Schmidt and Lipson [57]. Their method combines fitness prediction with coevolution to eliminate disadvantages of 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 [24]. Coevolving the training samples (often called *tests*) – as the method of fitness modeling in standard GP – has been studied in many application domains. However, coevolution applied to reduce the computational cost of CGP has not been considered until the research leading in this thesis.

1.1 Goals of the Thesis

On the basis of the previous survey of problems relevant for CGP, the following hypothesis has been formulated:

Research hypothesis: A properly designed mechanism employing coevolution of cartesian programs and tests is able to reduce the computational cost of CGP for considered applications in comparison with the standard CGP.

In order to confirm the hypothesis, more specific research goals have been defined:

Goal 1: To study the literature to get an overview of the state of the art in GP, CGP and coevolutionary principles.

Goal 2: To propose and implement CGP that uses coevolutionary principles.

- **2.1:** To solve how to choose such fitness cases that are effective in driving the search process towards a solution.
- **2.2:** To solve how to select a sufficient number of fitness cases.
- **2.3:** To evaluate the proposed approach using selected case studies, especially symbolic regression tasks.

Goal 3: To propose coevolutionary CGP suitable for FPGAs.

1.2 The Thesis Outline

The thesis is a collection of papers. The main ideas of this thesis and all relevant research results are, therefore, available in the papers. The papers were written with this thesis in mind and, therefore, build naturally on each other, leading towards the conclusion. Chapter 2 presents necessary background material. Chapter 3 provides an overview of the research process and the papers. Chapter 4 concludes the thesis, summarizes research contributions and provides suggestions for future work.

Chapter 2

Background

This chapter briefly surveys theoretical basis relevant to this work and gives a background to this thesis. Chapter 2.1 starts with a survey of the GP, including the graph-based GP approach, benchmark problems and real-world problems successfully solved by GP. Selected open issues – the so called *scalability problems* – are also introduced. Chapter 2.2 presents fitness approximation techniques as a possible way to overcome the *scalability of evaluation* problem. Chapter 2.3 gives a brief overview of the coevolutionary principles in GP. A summary of hardware acceleration techniques used in the evolutionary design is given in Chapter 2.4.

2.1 Genetic Programming

Genetic programming (GP) is an evolutionary approach that allows the exploration and exploitation of the space of computer programs. GP works by defining a goal in the form of a quality criterion (a *fitness function*) and then using this criterion to evolve a set (a *population*) of candidate solutions (*individuals*) by mimicking the basic principles of Darwinian evolution. GP tries to solve the problem using an iterative process involving the probabilistic selection of the fittest solutions and their variation by means of a set of genetic operators, usually crossover and mutation. GP became more widely known after the publication of John Koza's book in 1992 [31], but its roots can be traced back to Cramer's work [7].

In particular, GP genetically breeds a population of computer programs to solve a problem. Individuals of the population are hierarchical variable-size structures that represent computer programs. To code the GP implementation, Koza chose the LISP language where both programs and data are in the form of the list. Lists can be nested and can easily represent hierarchical structures such as *syntax trees*. While modern GP implementations are based on C, C++, Java or Python and syntax trees are often represented using flattened array-based representation rather than linked lists, programs are still treated and manipulated as trees as Koza did. This form of GP is called the *tree-based GP*.

While the tree-based representation of individuals is the oldest and most common one, there are other forms of GP. A growing attention has been dedicated by researchers to linear genomes [5] and graph-based genomes [43]. Many other forms of GP, based on, e.g., grammars or the estimation of probability distributions algorithms, have also been proposed. A more comprehensive review is provided in [50].

2.1.1 Cartesian Genetic Programming

This thesis deals with *Cartesian Genetic Programming* (CGP). The state of the art of CGP has been summarized in monograph [41]. CGP uses a specific encoding in the form of directed acyclic graph and a mutation-based search. CGP has been successfully employed in many traditional application domains of genetic programming such as symbolic regression, but has been predominantly applied in evolutionary design and optimization of logic networks.

A candidate program in CGP is modelled as a Cartesian grid of programmable elements (nodes) and defined using the following parameters (according to [43]):

- The problem-specific parameters:

  n_i: the number of program inputs,
  n_o: the number of program outputs,
  n_n: the number of node inputs,
  n_f: the number of node functions,
  Γ: the function look-up table, which contains n_f (up to) n_n-input functions of nodes.

- The user-defined parameters:

  n_r: the number of grid rows,
  n_c: the number of grid columns,
  l: the levels-back parameter, which controls the connectivity of the graph encoded, i.e. constraints which columns a node can get its inputs from.

The candidate program genotype G is then in the form of a string of $n_r n_c (n_n + 1) + n_o$ integers. The program data inputs are given the absolute data address $\langle 0, n_i - 1 \rangle$. The data outputs of nodes in genotype are given the address sequentially, column by column, particularly $\langle n_i, n_i + n_r n_c - 1 \rangle$. Nodes take their inputs in a feed-forward manner from the output of nodes in l previous columns or program inputs. Each node is encoded by $n_n + 1$ integers; where one integer is the address of the node function in Γ. The n_o integers, which specify where the program outputs are taken from, are added to the end of genotype. Figure 2.1 shows an example of a candidate program encoding.

This type of candidate program genotype encoding has some important features:

- some nodes may be ignored (the so-called *non-coding nodes*),
- some nodes can be multiply used,

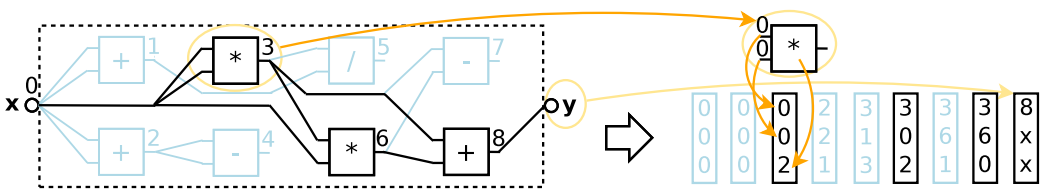


Figure 2.1: A candidate program (left) in CGP, where $l = 4$, $n_c = 4$, $n_r = 2$, $n_i = 1$, $n_o = 1$, $n_a = 2$, $\Gamma = \{+ (0), - (1), * (2), / (3)\}$ and its genotype (right): $0, 0, \underline{0}$; $0, 0, \underline{0}$; $0, 0, \underline{2}$; $2, 2, \underline{1}$; $3, 1, \underline{3}$; $3, 0, \underline{2}$; $3, 6, \underline{1}$; $3, 6, \underline{0}$; 8. Function genes are underlined.

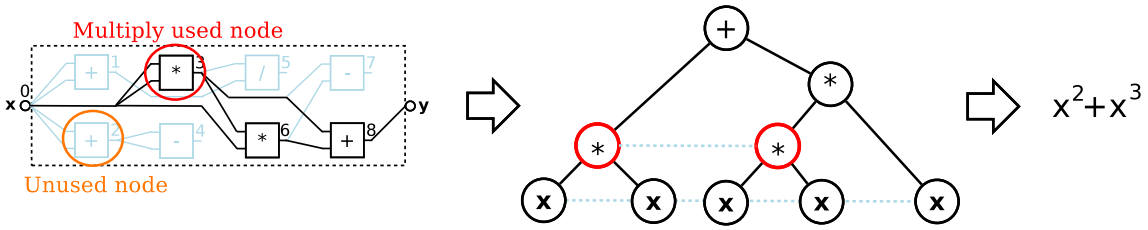


Figure 2.2: Example of CGP phenotype and its interpretation.

- some node inputs may be ignored,
- some primary inputs may be ignored.

While candidate program genotype has a fixed length, this representation has an unusual form of redundancy in which nodes may be switched on or off, under the control of evolution. The phenotype may consist of 0 up to $n_r n_c$ nodes. The role of redundancy and its utility in evolutionary search was investigated in [42]. Figure 2.2 shows an example of a candidate program genotype and its interpretation in the form of a syntax tree and a functional expression.

In CGP, a variant of a simple $(1 + \lambda)$ evolutionary algorithm is used as a search mechanism as follows:

1. The initial population is constructed either randomly, by a heuristic procedure or uses an existing solution.
2. All individuals in the population are evaluated.
3. The fittest individual is promoted as a new parent. If more than one individual show the best fitness, the new parent is selected randomly from the set of best individuals; however the previous parent is never taken to promote diversity.
4. The λ offspring are created from the parent using a point mutation operator. The mutation operator modifies h randomly selected genes to other randomly generated (but valid) values.
5. Every new population consists of the parent and its λ offspring.
6. If a solution is not found or the generation limit is not reached, continue to 2.

Recent work [21] analyzed the aspects of the CGP evolutionary mechanisms. It examined parameter values tuning, the search quality of CGP variants at different problem difficulties, node behavior, or offspring replacement. While a really useful crossover operator has not been proposed for CGP so far, several promising mutation techniques are investigated in [21].

While this chapter introduces CGP in its basic form (as it is used in this thesis), several more complex forms were proposed, such as embedded CGP, modular CGP, self-modifying CGP, developmental CGP, and others. The interested reader is referred to [41].

2.1.2 Fitness Evaluation in GP

Each program in the population is assigned a fitness value, representing the degree to which it solves the problem of interest. This value is calculated by means of a well-defined

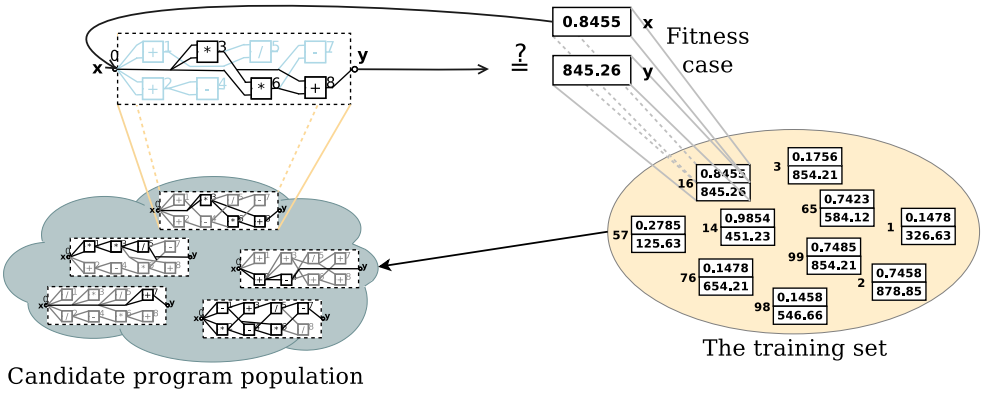


Figure 2.3: Fitness evaluation in the standard CGP.

procedure. The fitness in GP is often calculated over a set of *fitness cases* [69]. A fitness case corresponds to a representative situation in which the ability of the program to solve a problem can be evaluated. The fitness case consists of potential program inputs and target values expected from a perfect solution as a response for these program inputs. Figure 2.3 shows the scheme of fitness evaluation.

The set of fitness cases is typically a small sample of the entire domain space. The choice of how many fitness cases (and which ones) to use is often crucial since whether or not an evolved solution will generalize over the entire domain depends on this choice. In practice, the decision is made on the basis of knowledge about the problem and practical performance considerations – the bigger the training set, the longer the time required to evaluate fitness. The first theoretical study on the suitable number of fitness cases to be used has been presented in [20].

2.1.3 GP Benchmarks

Koza [31] introduced a set of problems that can be considered as *typical GP problems*. These problems have been adopted by GP research community as an agreed-upon set of benchmarks, to be used in experimental studies. They include the following:

- *The Even k bit Parity Problem*, whose goal is to find a Boolean function of k arguments that returns *true* if an even number of its arguments evaluates to *true* and *false* otherwise.
- *The h bit Multiplexer Problem*, where the goal is to design a Boolean function with h inputs that operates as a multiplexer function.
- *The Symbolic Regression Problem*, that aims at finding a mathematical expression that matches a given dataset.
- *The Intertwined Spirals Problem*, the goal of which is to find a program to classify points in the $x - y$ plane as belonging to one of two spirals.
- *The Artificial Ant on the Santa Fe Trail*, whose goal is to find a navigation strategy for an agent on a limited toroidal grid.

With the development of the field, the GP community analyzed the use of GP benchmarks and suggested that several problems in common use should be blacklisted. Possible

replacements were proposed [75]. Among others, they offered more complex symbolic regression benchmarks; the *multiple output multiplier* replaced the multiplexer problem, or the artificial ant was replaced by the *Mario gameplay*.

2.1.4 Examples of Real-World Applications of GP

Since the nineties, GP has produced a plenty of results. The literature reports an enormous number of applications where GP has been successfully used as an automatic programming tool, a machine learning tool or an automatic problem-solving engine. The interested reader is referred to [33] for more detailed analysis. Examples of real-world applications of GP that are relevant for this thesis are listed bellow.

Curve Fitting, Data Modeling and Symbolic Regression

A large number of GP applications are in curve fitting, data modeling and symbolic regression. GP can be used as a stand-alone tool or coupled with one of numerous existing feature selection methods, or even GP itself can be used for the feature selection [37]. Furthermore, CGP (or linear GP with multiple output registers) can also be used in cases where more than one output is required.

Evolvable Hardware

GP has now been successfully used to automatically synthesize designs in a number of fields, many of them were classified as *human-competitive* [35]. GP can be applied to create novel topologies for optical systems [1], antennas [39] or electronic circuits [61]. As CGP originally developed from a method for evolving digital circuits, it is often applied to challenging problems in digital circuit design.

Image and Signal Processing

GP can be used for motorway traffic jams prediction from subsurface traffic speed measurements [25], preprocessing images of human faces to find regions of interest for subsequent analysis [68], classification of objects, and speech classification [80, 77]. Furthermore, evolutionary design of low level image filters by means of CGP resulted in novel designs showing very good quality of processing and low implementation cost in comparison with conventional image filters [60].

Genetic Improvement of Software

Nowadays, the genetic improvement of software is significantly growing. It uses automated search to find improved versions of an existing software. This field of research has been recently surveyed in [49].

2.1.5 Open Issues

The scalability problem is usually understood as a problem in which the evolutionary algorithm (EA) is able to provide a very good solution to a small problem instance; however, only unsatisfactory solutions can be generated for larger problem instances. Because a kind of evolutionary algorithm is employed, the user is supposed to design a suitable problem

encoding, search strategy (i.e. the search algorithm including genetic operators) and fitness function. All these components can suffer from the scalability problems.

Scalability of Evaluation

In order to evaluate a complex candidate solution, a time consuming fitness evaluation procedure has to be undertaken in which the evaluation time typically grows exponentially with the size of the problem. This difficulty is known as the *scalability of the fitness calculation*.

Fitness evaluation is often the most time consuming activity in GP applications and it is thus useful to find techniques that can help in reducing the evaluation time without compromising the quality of the results. The problem of determining adequate training samples of data is common in the machine learning field and it is shared by GP.

In GP, the training set can be small and complete according to the solved problem. Considering the design of a one-bit full adder, the training set consists of 8 training vectors, which fully represent the functionality of the desired circuit. Each candidate program is then executed 8 times to evaluate its fitness. In the case of more complex circuits such as the 8b×8b multiplier, the complete training set consists of 2^{16} training vectors – the bigger the training set, the longer the program evaluation process.

However, in the case of digital circuit evolution, it is necessary to verify whether a candidate n-input circuit generates correct responses for all possible fitness cases (input combinations, i.e. 2^n assignments). It was shown that testing just a subset of 2^n fitness cases does not lead to correctly working circuits [26]. This problem can partially be eliminated in real-world applications by applying formal verification techniques [71].

In many GP applications, it is sufficient just to approximate the desired functionality. The goal is then to minimize the error for the training set, which can be large and incomplete. Considering symbolic regression, the experimentally obtained training data are often noisy (containing measurement errors) and thus it is not necessary to find the exact representation of the training set (which often contains from hundreds to ten thousands fitness cases). The desired expression should often be the simplest discovered form, that provides the predefined accuracy.

The time needed for evaluating a single fitness case depends on a particular application. However, the computation time that the evolutionary design based on GP requires for obtaining innovative results is enormous for complex real-world applications. Usually, the goal of GP system design is to obtain a solution with predefined accuracy and robustness using a minimum number of evaluated fitness cases. There are two basic problems:

I. How to select the sufficient number of fitness cases.

II. How to choose fitness cases in such a way that they are effective in driving the learning process towards a solution.

A commonly accepted principle, called *minimum description length*, states that the best model for explaining training set is the one that minimizes the amount of information needed to encode it. The minimum description length idea has been incorporated into GP fitness functions to avoid overfitting and *bloat*[1] [79]. Another theoretical contribution showed that, under certain conditions, the number of fitness cases needed in GP fitness evaluation

[1]The average size of the programs in a GP population often starts growing during the evolution flow, but the increase in program size is not accompanied by any corresponding increase in fitness [50].

can be upper-bounded via statistical and information-theoretic considerations [20]. Results indicated that the number of fitness cases needed for reliable function reconstruction in GP is well approximated by the entropy of the target function.

Other works approached the second problem (II.) from a heuristical point of view with the aim of somehow sorting, cutting, or reordering the fitness cases in such a way that they are more useful in driving the learning process towards a solution [18, 19]. A brief overview of more complex heuristical approaches to selecting proper fitness cases is given in Chapter 2.2.

Scalability of Representations

As complex solutions are usually represented by long chromosomes, it is difficult to establish a fast and accurate search method which will be capable of finding good solutions in the corresponding complex search spaces. This problem is referred to as the *scalability of representations* problem. Hence, the third problem related to open issues in GP is:

III. How to eliminate the scalability of representation problem.

The most important approaches will be surveyed in the following paragraphs.

At the level of the search algorithm, modularization has been introduced for genetic programming (e.g. in the form of automatically defined functions [32]) as well as Cartesian genetic programming [73]. In the framework of *co-evolutionary algorithms*, several methods have been developed to reduce the computational requirements. A survey will be given in Chapter 2.3.

There are also problem specific approaches to get over the scalability of representations problem. Approaches developed in the circuit design area which is the most relevant for CGP are listed bellow.

A *divide and conquer* approach firstly (either deterministically or heuristically) divides the target circuit into modules (or subcircuits) and then evolves a solution for each module separately [67]. The benefits are twofold: reducing the search space and simplifying the fitness calculation. The approach can be applied iteratively [66].

Functional level evolution allows utilizing complex circuit elements (such as adders, multipliers or comparators) instead of elementary gates [45]. Relatively complex circuits can then be encoded using a shorter chromosome and the search space is thus restricted. The functional level evolution is often combined with a suitable decomposition scheme [62].

Another class of approaches, inspired by the biological development, employs an *indirect problem encoding* [34, 22]. A program is encoded in the chromosome rather than a circuit. When executed, the program is capable of constructing a complete phenotype from an initial solution, which is usually called the embryo. While designing a suitable developmental encoding is tricky, main advantages are shortening the genotype and obtaining a natural support for modularity and scalability in resulting circuits.

2.2 Fitness Approximation

The goal of the GP software design and GP parameters tuning is to automatically obtain a program (solving the target problem with predefined accuracy and robustness) in as short time as possible. In practice, this time is measured as the number of evaluated fitness cases or fitness function calls. In order to reduce the computational complexity of expensive fitness evaluation, *fitness approximation* techniques have been developed.

2.2.1 Fitness Modeling

One of the fitness approximation technique is *fitness modeling* which employs fitness models with different degrees of sophistication to reduce the evaluation time [27]. A predefined model or coarse-grained simulation has been used to approximate the fitness value in the cases in which obtaining the exact fitness requires an expensive simulation or a physical experiment.

The concept of fitness modeling falls in the field of machine learning. Neural networks, support vector machines, decision trees and other machine learning methods can be used in order to efficiently approximate the exact fitness [28]. Sub-sampling of training data such as random subset selection [18], stochastic sampling [46] or dynamic topology-based selection [38] have also been studied in order to evaluate an individual on a smaller subset of fitness cases.

Fitness modeling reduces the execution time in high-complexity or interactive domains, where the resolution provided by the exact fitness is unnecessary for the design process [28, 53]. Fitness models have also been applied in domains without computable fitness, such as evolution of art and music [29], or with very noisy fitness functions [2]. Fitness approximation can also reduce the frequency and severity of local optima in multimodal landscapes. Moreover, the motivation for fitness modeling can also be seen in smoothing the fitness landscape and promoting diversity [57]. However, it is not always clear when the benefits of fitness modeling can outweigh the cost.

The use of fitness modeling techniques is often connected with three fundamental difficulties [57]:

- Significant computational effort is required to train the desired model.
- It is often unclear what level of approximation is accurate enough to achieve desired solution – the higher-quality approximations provide greater accuracy, but require more computation, and vice versa.
- Approximations are bound with loss of accuracy, which, in the worse case, can lead to losing the global optimum.

2.2.2 Fitness Prediction

A closely related concept to fitness modeling is *fitness prediction*, which is a technique used to replace fitness evaluations by a lightweight approximation that adapts with the solution evolution. Fitness predictors cannot approximate the entire fitness landscape, but they are instead shifting their focus throughout the evolution. An algorithm that coevolves fitness predictors, optimized for the solution population, has been introduced for tree-based GP [57]. Results indicated that the fitness evaluation cost and also the bloated solutions can be reduced.

2.3 Coevolutionary Algorithms

Coevolutionary algorithms (CoEAs) are characterized by comparing individuals on the basis of their outcomes from interactions with other individuals.

Considering EA fitness function of the form $f : G \to \mathbb{R}$, a real value is assigned to each possible genotype in G. Then, the relationship between any two genotypes $g_1, g_2 \in G$ is

clear – $f(g_1)$ is compared to $f(g_2)$ to find out which one is more fit. The objectively given fitness function used in typical EAs has been denoted as *objective fitness*. Contrary to the traditional EAs, in CoEAs individuals can be evaluated by interacting with other individuals from the same population or individuals in one population interact with individuals in one or several other populations. Thus, the fitness ranking of two individuals can change over time. The fitness in CoEAs has been called *subjective fitness*.

CoEAs are traditionally used to evolve interactive behavior which is difficult to evolve with an absolute fitness function. The state of the art of coevolutionary algorithms has been summarized in the Chapter 31 of Handbook of Natural Computing [51].

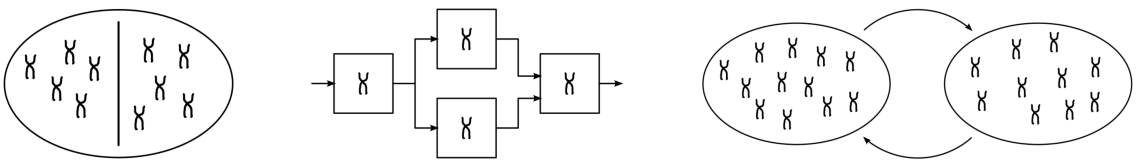
Historically, the terms *cooperative* and *competitive* have been used to classify the domains in which coevolution is often applied. These terms appear from the game theory, but they have been appropriate neither for classifying problems over which CoEA operates nor algorithms themselves. According to [51], problems are primarily divided into classes based on what constitutes a solution. Two types of problems are distinguished – *compositional* problems and *test-based* problems.

In CoEAs, relationship between terms such as *individuals*, *populations* and *solutions* can be more complicated than in traditional EAs. Depending on the problem, an individual can represent a candidate solution, but also a *component* (see Chapter 2.3.1) of a solution or a *test* (see Chapter 2.3.2) for solution evaluation.

The population is a collection of individuals (as well as in traditional EAs). In CoEAs, two simple algorithms are distinguished: the *single-population CoEA*, and the *multi-population CoEA*. An example of the single-population CoEA are the neuro-developmental programs that play checkers [30]. Each individual in population is evaluated in the way, that it plays checkers against the other individuals from the same population (see Figure 2.4a). Multi-population CoEAs are characterized by using populations, where different types of individuals are evolved in each population. Then, each population can contain one component of a problem (see Figure 2.4b), or one population contains candidate solutions and the other contains the test for candidate solution evaluations (see Figure 2.4c).

However, coevolutionary methods often employ another kind of collection than population – an *archive*. In general, the archives contain individuals that span generations. The archives often contain the end solution (i.e. the best solution obtained during the evolution flow) [54]; or when the solution is the aggregation of components, the composition of archives is the solution [8, 47]. Likewise elitism in traditional EAs, archives can also influence the search direction.

To evaluate individuals according to interactions with other individuals, decisions need to be made as to what *interactions* should be assessed and how the outcomes of those interactions should be *aggregated* to give individuals fitness. When using multiple populations, also the issue of *communication* between these populations should be considered.



(a) Single population. (b) Compositional problem. (c) Test-based problem.

Figure 2.4: Basic types of CoEAs.

The simplest choice is to enable all possible interactions (i.e. *complete mixing*). However, this approach is computationally very expensive. Only some of all interactions are often allowed and assessed. In order to aggregate the values coming from multiple interactions into the fitness score, two basic approaches are considered:

- to aggregate values from multiple interactions into a single value;
- to define the fitness with a *tuple* and select parents with a *multi-objective* EA.

2.3.1 Coevolution Applied to Compositional Problems

Coevolution applied to compositional problems sprang from the cooperative coevolutionary algorithms, wherein the originally stated aim was to attack the problem of evolving complicated objects by explicitly breaking them into parts, evolving parts separately and then assembling the parts into a working whole [52]. There is a number of successful applications of CoEAs to compositional problems. For example, in neuro-evolutionary algorithms, weights and structure of artificial neural networks are often coevolved [65, 44, 64, 63].

Coevolution applied to compositional problems has also been proposed as a promising framework for solving high-dimensional optimization problems. However, it is not always clear how to decompose a problem into single variables. A new problem decomposition strategy (the grouping based strategy), in order to better capture the variable interdependencies for complex non-separable problems, has been proposed in [78].

2.3.2 Coevolution Applied to Test-based Problems

In coevolution applied to test-based problems, the quality of a potential solution is determined by its performance when interacting with a set of tests. Hillis [24] introduced an approach that can automatically evolve subsets of fitness cases concurrently with a problem solution. Hillis employed a two-population coevolutionary algorithm in the task of a minimal sorting network design. Subsets of fitness cases composed to evaluate candidate sorting networks were evolved simultaneously with the sorting networks. Evolved sorting networks were applied to evaluate the fitness cases subsets. The fitness of each sorting network was measured by its ability to correctly solve fitness cases while the fitness of the fitness cases subsets was better for those that could not be solved well by currently evolved sorting networks.

The test-based problems are discussed in [10] and analyzed in connection with a multi-objective optimization in [9]. Coevolving the fitness cases as the method of fitness modeling in GP has been studied in many application domains [13, 17, 40] as well as in symbolic regression problems [12, 48, 56, 57].

2.3.3 Open Issues

The successful use of any EA is connected with many key decisions that have to be decided before executing it. For instance, choosing the candidate solution representation, population size, the genetic operators used to explore the search space and the fitness function(s) are surely the most important ones. Compared to traditional EAs, the CoEAs use more components, which are expected to interact, and thus the success of a particular CoEA is dependent on many more parameters.

For instance, the occurrence of two pathological coevolutionary behaviours (disengagement and cycling) in simple genetic programming systems has been investigated in [58].

The disengagement occurs when an element of the system has entered a state for which no search gradient can be induced by reference to the other coevolving elements. Cycling occurs when previously visited interactions recur so that the search process is led to repeat a previous progress of evolution. Next, under certain conditions, it is theoretically possible for some CoEAs to prefer components later in the run, which will tend to require larger supporting tests [6]. This feature may cause a kind of bloating.

Although many theoretical works are concerned with analytical description of CoEAs, it is still an open research issue how to utilize all benefits that CoEAs bring to real algorithms and for real problems.

2.4 Hardware Acceleration of CGP

As the design based on CGP (as well as on other forms of GP) is computationally a very intensive method, it has been usually accompanied by application-specific acceleration techniques. Among others, many hardware accelerators have been designed, such as *field programmable gate array* (FPGA) based acceleration platforms.

FPGAs are pre-fabricated silicon devices that can be electrically programmed in the field to become almost any kind of digital circuit or system [16]. Modern FPGAs provide cheap, flexible and powerful platform, often outperforming common workstations or even clusters of workstations in particular applications. Normally FPGAs comprise of programmable logic blocks which implement logic functions, programmable routing that connects these logic functions, and I/O blocks that are connected to logic blocks through routing interconnect and that make off-chip connections. A generalized example of an FPGA is shown in Figure 2.5 where configurable logic blocks (CLBs) are arranged in two-dimensional grid and are interconnected by programmable routing resources. I/O blocks (IOB) are arranged at the periphery of the grid and they are also connected to the programmable routing interconnect. A CLB consists of so-called slices. Each slice contains the function generators implemented using 3-,4-, or 6-input look-up tables (LUTs), flip-flops (FFs), and some additional logic. The configuration bitstream is stored in the configuration SRAM memory.

Hardware implementations of CGP are typically developed with the aim of either enabling autonomous system adaptation at the hardware level and in a real environment, or as accelerators reducing the execution time in comparison with a pure software implementation [55, 11, 60]. Vasicek and Sekanina [71] introduced a new FPGA accelerator of CGP with the aim to provide high performance together with low power. The architecture contains multiple instances of *virtual reconfigurable circuit* (VRC, [59]) to evaluate several candidate solutions in parallel. Dobai [11] showed that when properly accelerated in an FPGA, CGP can evolve a unique image filter for every frame of a video played with a resolution of 427×240 pixels. In this task, CGP can generate and evaluate over 9,200 candidate filters per second, each of them is evaluated using a 128×128 pixel image taken from the previous frame.

It has to be noticed that only the standard CGP was accelerated in FPGAs. No advanced CGP versions (such as coevolutionary CGP) are available.

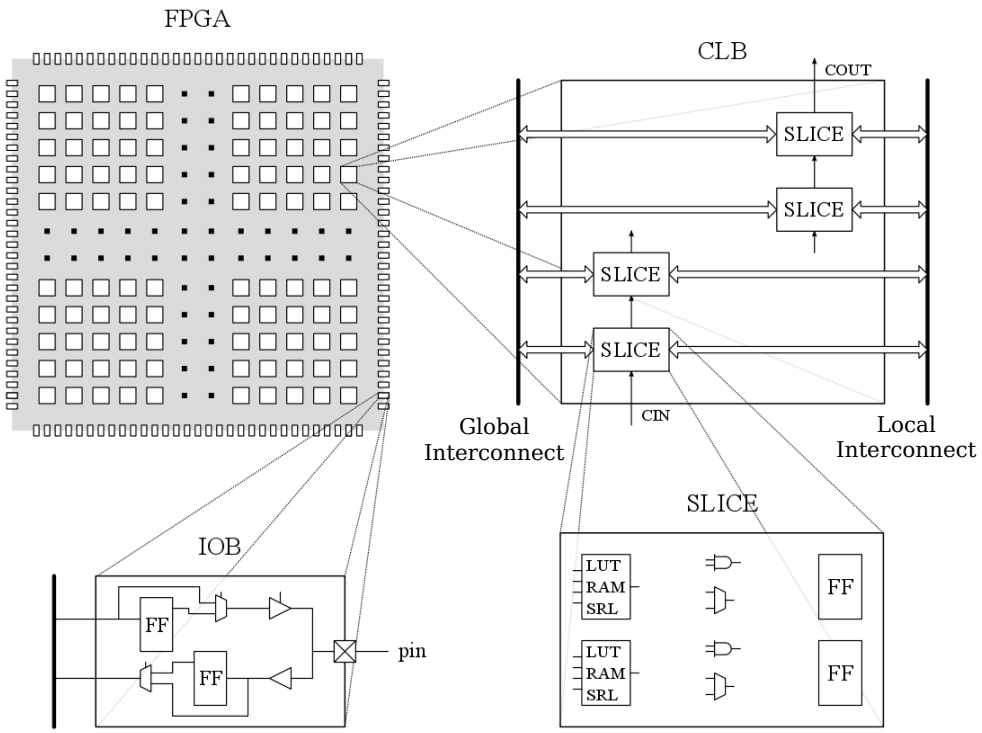


Figure 2.5: Overview of FPGA architecture.

Chapter 3

Research Summary

This chapter gives a summary of the research behind this thesis. Chapter 3.1 starts with a description of the research process that led to this thesis. Chapter 3.2 lists five papers [A,B,C,D,E] included in this thesis, representing the research path taken towards finding answers to the research hypothesis of this work. The reader is supposed to consult these papers when reading this summary.

3.1 Research Process

This chapter presents the research process that led to this thesis. The CGP acceleration at the level of the search process inspired by coevolution of fitness predictors and tree-based GP introduced in [56, 57] was chosen as the main topic from the very beginning. The method combines fitness prediction with coevolution to eliminate disadvantages of the classic fitness modeling, in particular the effort needed to train the fitness model and adapt the level of accuracy. This approach had to lead to reduction of expensive fitness evaluations and thus to help to eliminate the scalability of evaluation problem in CGP.

After the initial study of literature, the coevolutionary algorithm was adapted for CGP as presented in paper [A]. CGP has some important differences compared to the tree-based GP. CGP uses very small populations (usually 1+4 individuals) which implies many generations have to be performed (compared to tree-based GP) to obtain a solution. This feature is the main aspect to consider while designing the coevolutionary interactions in CGP. Together with the adaptation of the coevolutionary algorithm, the experimental evaluation process was specified, which is summarized in Chapter 3.1.4. The adaptation was accompanied by a large number of experiments in order to find proper settings of the components involved in the coevolutionary algorithm.

The proposed method employs four collections of individuals. There are two populations:

- population of candidate programs,
- population of fitness predictors.

Fitness predictors are surveyed in Chapter 3.1.1. In order to enable interactions between the populations, two archives are used:

- archive of *fitness trainers*,
- archive containing the current best evolved fitness predictor.

The archive of fitness trainers is used by the fitness predictors population for the evaluation of evolved fitness predictors, which is summarized in Chapter 3.1.3. Chapter 3.1.2 gives the overall preview of the proposed algorithm in terms of how each collection of individuals interact to each other.

Although several possible encodings were mentioned in [56], both [56] and [57] used fitness predictors encoded as the constant-size array of fitness cases. In the proposed co-evolutionary CGP, this type of fitness predictor encoding was considered as the first choice (see Chapter 3.1.5). We decided to evaluate proposed approach on five symbolic regression benchmarks. We obtained a significant reduction in the number of fitness evaluations – comparable to the original approach involving tree-based GP. Results were summarized in paper [A]. The problem of how to choose fitness cases in such a way that they are effective in forcing the CGP towards a solution was solved by the proposed approach.

Next, we employed this approach in a real-world problem which is more typical for CGP – the evolutionary design of low-level image filters. We also obtained a significant speedup [B] compared to the standard CGP, but many experiments were accomplished in order to find the most advantageous number of fitness cases in the fitness predictor (i.e. the *predictor size*), for this new task.

This disadvantage focused our research on the second problem: how to automatically select a sufficient number of fitness cases directly by the coevolutionary process. In order to solve this problem, new approaches to fitness predictor encoding were applied (papers [C] and [D]). Therefore, this thesis proposes and analyzes three types of fitness predictor encoding:

- a constant-size fitness predictor (Chapter 3.1.5),
- an indirectly encoded fitness predictor (Chapter 3.1.6),
- an adaptive-size fitness predictor (Chapter 3.1.7).

As many problem-specific hardware accelerators have been introduced in order to accelerate fitness evaluations in CGP, the proposed coevolutionary algorithm was also implemented into hardware, as summarized in Chapter 3.1.8.

3.1.1 Fitness Predictor

For purposes of this thesis, the fitness predictor is a small subset of the training data. In context of coevolution applied to a test-based problem, the fitness predictor is a special form of a *test*. Fitness predictor is represented as an array of pointers to elements in the training data. Figure 3.1 illustrates how every pointer addresses one selected fitness case.

A good fitness predictor provides a fitness prediction which is robust enough to differentiate the fitness between any pair of candidate programs, along with a significantly faster outcome than the objective fitness calculation.

3.1.2 Proposed Coevolutionary Algorithm

There are two concurrently evolved populations in the proposed coevolutionary algorithm. Figure 3.2 shows the overall interaction scheme as introduced in paper [A].

The first population is the population of candidate programs evolved using CGP. The goal of the program evolution, formalized in the fitness function, is to minimize the difference between responses produced by a candidate solution and desired responses. While the

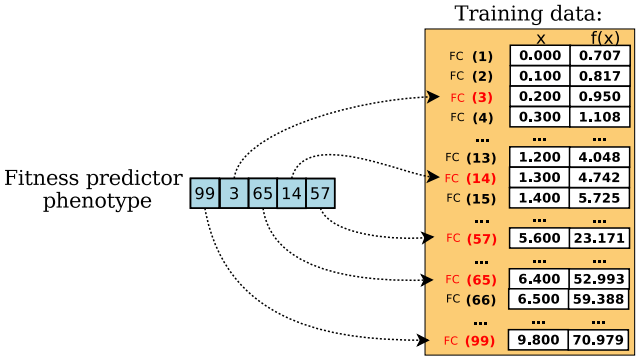


Figure 3.1: Fitness predictor addresses selected fitness cases (FCs).

standard CGP uses the complete training data to evaluate candidate programs, the fitness prediction-based method uses only a selected subset of the training data (i.e. the fitness predictor) to estimate the program fitness.

In the second population, fitness predictors have to be coevolved with the programs in order to adapt them to the solved task. The fitness predictor training data consists of fitness trainers that are selected copies of candidate programs occurred during the program evolution. The fitness predictor evaluation process is described in Chapter 3.1.3. The detailed implementation and pseudo-code for coevolution of the population of programs and the population of fitness predictors is shown in paper [B], Section 3.3.

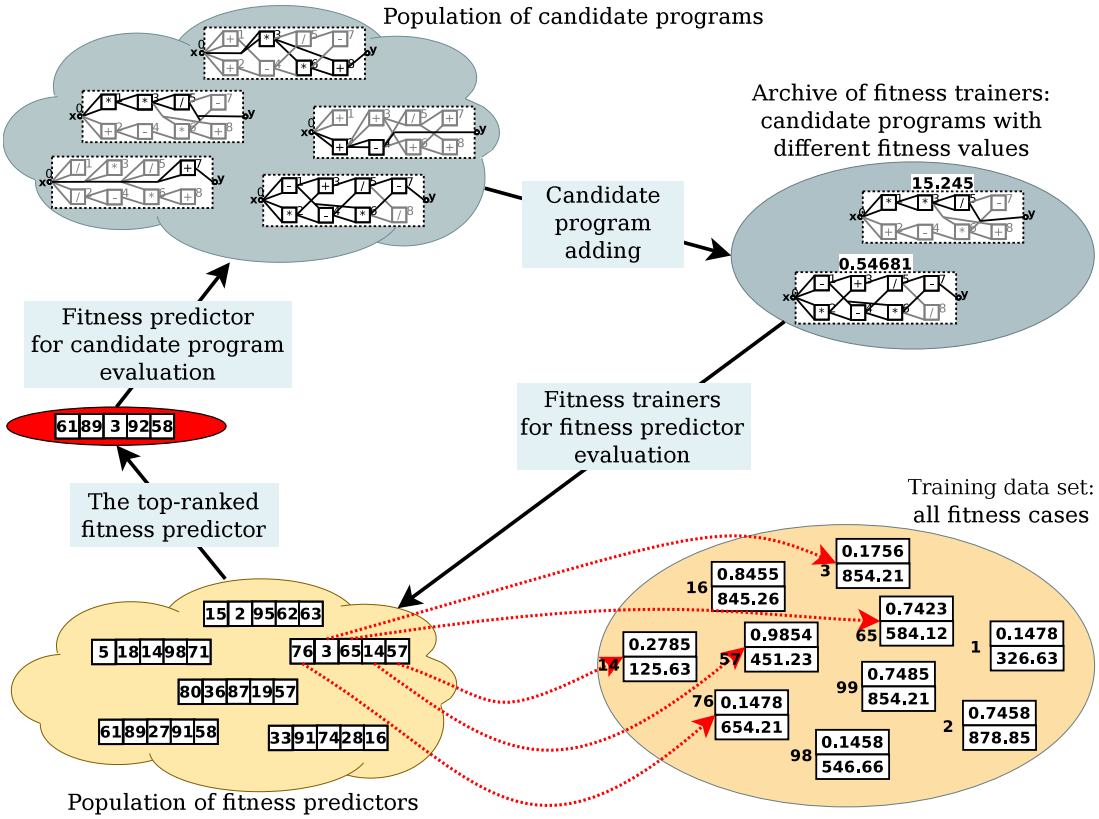


Figure 3.2: The overall coevolutionary scheme.

While using coevolutionary algorithms, there are many settings of involved components to make coevolution produce satisfactory solutions. In addition to a correct setting of the search algorithm in each population, the proper setup of interactions, communication and synchronization has to be decided. In general, the time necessary to perform one generation is different in each population. The evolution loops can be executed in parallel in separate computing threads. In this case, the archives can be updated immediately after a new top-ranked individual is evolved. However, the experiments showed that it is not necessary to update the fitness predictor archive each time a new top-ranked fitness predictor is found. For this reason, it is possible to specify the relative speed between the populations (i.e. how many generations of the faster algorithm are performed per one generation of the slower algorithm).

Another option is to execute both evolutions in a single-threaded environment by interleaving the evolution loops. In this case, there is in fact only one evolution loop, which performs a specified number of generations in the program evolution in each iteration, followed by one or more generations of the fitness predictors evolution.

Our experiments had shown that frequent interactions between populations do not lead to correctly working solutions, because of very fast changes in the fitness calculation procedures. As CGP uses small populations, the search process needs more generations to adapt to the changes of the selective pressure. The most suitable setup of coevolution of fitness predictors and CGP is presented in paper [A] and used in all other papers included in this thesis.

3.1.3 Fitness Predictor Evaluation

The fitness predictors are evaluated using *fitness trainers* (trainers for short). Trainers are selected copies of candidate programs occurred during the program evolution. Trainers in the archive are updated periodically – the top-ranked candidate program is copied to the trainers archive if its predicted fitness value differs from the top-ranked candidate program in the previous generation. A new (randomly generated) trainer t replaces the oldest one in the trainers archive and the objective fitness of the new trainer $f_{objective}(t)$ is evaluated. This approach to the fitness predictor training leads to maintaining a representative sample of the current program population (due to the copies of top-ranked candidate programs) as well as increasing fitness diversity of programs in the trainers archive (due to the randomly generated trainers).

The fitness value of the fitness predictor is calculated using the mean absolute error of the objective fitness and predicted fitness of fitness trainers. Let us consider a symbolic regression problem where the candidate program fitness is represented as the number of hits. In common (non-coevolutionary) GP, the program fitness function (taking candidate program s as its argument) is then defined

$$f(s) = \sum_{j=1}^{k} \mathrm{g}(y(j)), \text{ and} \tag{3.1}$$

$$\mathrm{g}(y(j)) = \begin{cases} 0 & \text{if } |y(j) - t(j)| \geq \varepsilon \\ 1 & \text{if } |y(j) - t(j)| < \varepsilon \end{cases} \tag{3.2}$$

where $y(j)$ is the response of candidate program s for the j-th fitness case, t is the target response, k is the number of fitness cases in the training data, and ε is a user-defined maximum error.

When the coevolutionary GP with the fitness predictor is employed, there are, in fact, two fitness functions for candidate program s. While the objective fitness function $f^{(s)}_{objective}$ uses the complete training data, the predicted fitness function $f^{(s)}_{predicted}$ employs only selected fitness cases. Formally,

$$f^{(s)}_{objective} = \frac{1}{k}\sum_{j=1}^{k} g\left(y\left(j\right)\right), \tag{3.3}$$

$$f^{(s)}_{predicted} = \frac{1}{m}\sum_{j=1}^{m} g\left(y\left(j\right)\right) \tag{3.4}$$

where k is the number of fitness cases in the training data and m is the number of fitness cases in the fitness predictor (i.e. m is the size of the fitness predictor).

Fitness value of the fitness predictor (taking fitness predictor p as its argument) is then expressed as

$$f\left(p\right) = \frac{1}{u}\sum_{i=1}^{u} \left|f_{objective}\left(i\right) - f_{predicted}\left(i\right)\right| \tag{3.5}$$

where u is the number of candidate programs (trainers) in the trainers archive, and $f_{objective}$ and $f_{predicted}$ are their objective and predicted fitness values.

3.1.4 Experimental Evaluation of Coevolutionary CGP

Proposed approaches were compared with standard CGP on selected applications and with each other. The selected applications are as follows:

- the symbolic regression problem and
- the low-level image filter design.

Symbolic regression can be considered as a typical GP benchmark [31]. Hence, proposed approaches were evaluated on five symbolic regression benchmarks. Three of them were also used to evaluate the original fitness predictors based on the tree-based GP [57]. It enabled us to compare the use of fitness predictors with tree-based GP and proposed fitness predictors with CGP, among others. The other two benchmarks were taken from [72] (which are also mentioned in [75] as *better GP benchmarks*). These benchmarks were selected to easily observe the behaviour of fitness predictors. The benchmarks are described in paper [A].

The low-level image filter design problem was taken from [60] and described in paper [B]. The main difference compared to the symbolic regression benchmarks, in context of fitness predictors, is in the structure of the training data. In symbolic regression benchmarks, the training data are often sorted and, when visualized, one can recognize fitness cases placed in peaks, valleys or flexes. The filter design benchmarks employ much wider training set and the particular selection of fitness cases in the fitness predictor is not so easy to explain. The fitness cases are not sorted as in the case of symbolic regression benchmarks, but can be classified on the basis of their features – for example, a fitness case can represent the neighbourhood of corrupted pixel, while designing noise-suppressing filters, or the neighbourhood of a pixel representing the edge, while designing the edge detector.

In order to evaluate the impact of proposed approaches, the following attributes were observed:

- the quality of evolved solutions,
- the number of generations of program evolution needed to obtain a satisfactory solution,
- the number of fitness case evaluations during the whole coevolutionary process,
- the execution time (as the the sum of execution times of all used resources during the whole coevolutionary process, i.e. sequential time of all processes even if running in parallel).

All proposed coevolutionary approaches were compared with the standard CGP. While evaluating standard CGP and CGP employing proposed coevolutionary approach, the identical CGP implementation (and CGP setup), running on the same machine, was employed. Various parameter setting of the fitness predictor evolution and coevolutionary interactions were investigated and compared while searching for the most suitable parameters. All experiments were repeated at least 50 times and results were statistically evaluated.

The proposed approaches are then compared with each other, which is summarized in Chapter 4.1 of this thesis. Finally, the coevolutionary CGP implemented in hardware was compared with the highly optimized software implementation of the coevolutionary CGP [E].

3.1.5 Constant-Size Fitness Predictor

The first approach we proposed, introduced the fitness predictor encoding in the form of a constant-size array of pointers to elements in the training data [A]. Constant-size fitness predictors (CSFPs) are operated using a simple genetic algorithm (GA). The interested reader is referred to [76] for a survey of models and methods of GAs. In addition to one-point crossover and mutation, a randomly generated fitness predictor replacing the worst-scored fitness predictor of the generation was introduced as a new genetic operator of GA. The fitness value of fitness predictor is then calculated as the mean absolute error of the objective and predicted fitness values of fitness trainers according to formula 3.5.

Results

The CSFP approach was evaluated with the symbolic regression benchmarks [A]. It was shown in paper [A] that CGP equipped with coevolution of CSFPs can significantly be accelerated in this particular application. The speedup obtained for five benchmarks is 2.03 – 5.45 over the standard CGP. It should be pointed out, that standard CGP evaluated 200 fitness cases in every fitness function call, while the coevolutionary CGP evaluated only 12 fitness cases. Results are also very competitive with the tree-based GP introduced in [57].

While symbolic regression has not been considered as a typical application domain for CGP, the following research was devoted to utilization of the proposed approach in another application domain, where the standard CGP has been successful so far – the low-level image filter design. We investigated the coevolution of CSFPs in the evolutionary design of filters suppressing the so-called salt-and-pepper type of noise [B]. The median time of evolution was reduced 2.99 times in comparison with the standard CGP. However, this work revealed that the recommended size of the fitness predictor differs from task to task. In this case, 15 – 20 % pixels of original training image (i.e. thousands fitness cases) were needed

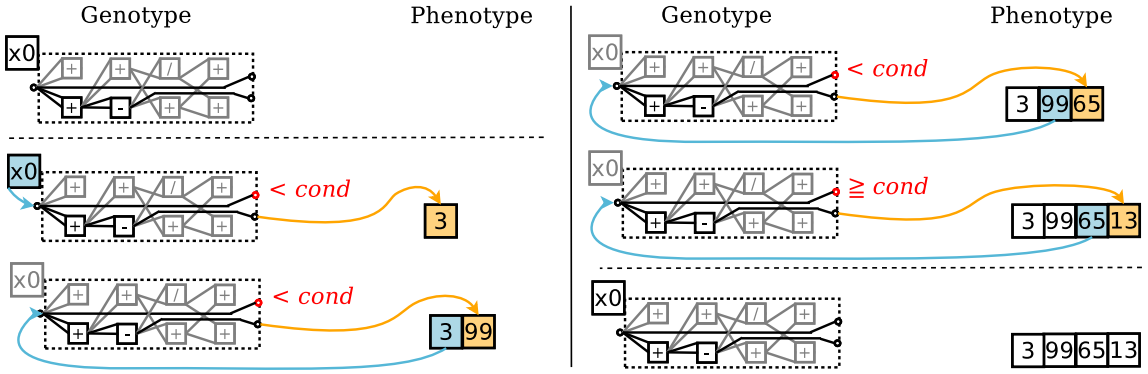


Figure 3.3: Phenotype construction with IEFP.

to find an image filter which provided the same quality of filtering as the best filter evolved using the standard CGP which utilized the whole 256×256 pixel image. It means, that in order to find the most advantageous size of the fitness predictor for a particular task, a large number of experiments have to be accomplished.

3.1.6 Indirectly Encoded Fitness Predictor

In order to reduce a large number of experiments needed to find the most advantageous size of fitness predictor for a new unknown task, a new type of fitness predictor whose size is changing dynamically during the coevolution, was developed. Indirectly encoded fitness predictors (IEFPs) with a variable number of fitness cases are represented in the form of functional expressions. This functional expression, evolved by means of CGP, generates a certain number of indexes into the training data (see Figure 3.3). Indexes then address specific fitness cases from the original training data which are selected for program fitness prediction. Details of IEFP encoding, evaluation and search process are given in Section 4 of paper [C].

Results

The IEFPs were evaluated in five symbolic regression benchmarks [C] and were found to be comparable with the original approach presented in paper [A] (which used only 12 fitness cases). The detailed comparisons of these algorithms are given in paper [C]. Section 5.3 and Section 5.4 of paper [C] analyzed the behaviour of IEFP. It was shown that during the evolution of IEFPs, also large fitness predictors had to be evaluated (and then refused for their larger size), and thus plenty of fitness case evaluations were wasted. In order to prevent this overhead, the upper bound of fitness predictor size was considered, and the large fitness predictors have been refused before evaluation, to get closer to the evaluation cost of the CSFPs approach. However, this hard constraint prevented creating larger fitness predictors even if they were needed in the particular task.

3.1.7 Adaptive-Size Fitness Predictor

The adaptive-size fitness predictor (ASFP) was inspired by the principle of *phenotypic plasticity*, which is the ability of an individual to learn how to utilize its genotype in order to adapt to the environment [3]. It was shown that a proper rate of environmental change may reduce the learning cost while evolving, for example, neural networks [14, 15].

The ASFP genotype is a constant-size circular array of pointers to fitness cases. Its size is equal to the total number of fitness cases in training data. In order to generate the phenotype, the genes are read sequentially from a specified position. The reading stops after it has processed a specified number of genes. Figure 4 in paper [D] illustrates the genotype-phenotype mapping. The number of genes in the phenotype is adjusted in response to the current state of program evolution.

It can be observed that the evolution, in general, goes through various phases. At first, the overall fitness increases towards better solutions, until the evolution reaches a local optima. In our approach, the number of fitness cases in the fitness predictor is reduced in this phase, which increases the difference between predicted and objective fitness of candidate programs. This leads to higher population diversity and higher probability that a program situated farther from the local optima becomes the parent of the next generation. When the evolution leaves the local optima, the predictor size is increased in order to predict the fitness more accurately and thus evolve better programs. For the detailed survey of how the ASFP is encoded, and its size adapted, see Section 3 in paper [D].

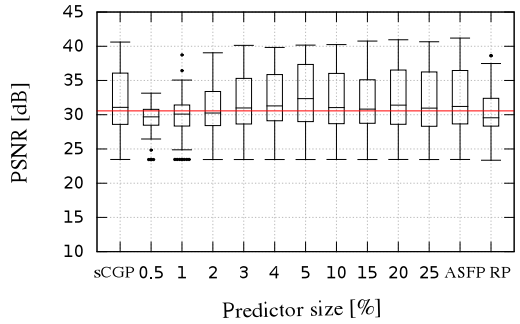
Results

The ASFPs were applied to the five symbolic regression tasks [D]. It was shown that the proposed approach outperforms the original CSFPs [A], which used only 12 fitness cases, in terms of success rate and computational cost, expressed as the number of fitness case evaluations to converge. The detailed comparisons of these algorithms are given in paper [D], Section 4.6. Observing the ASFP behaviour revealed that the proposed algorithm is able to adapt the predictor size to the solved problem in response to the development in the candidate program evolution.

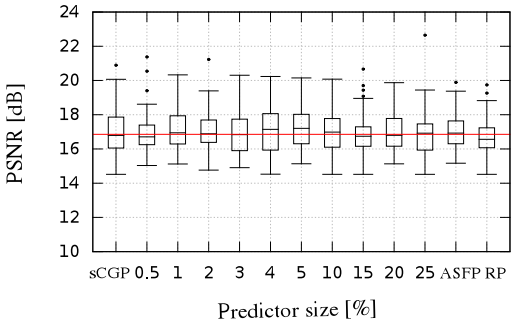
The proposed method was then evaluated in the task of evolutionary design of image filters of various types[1]. Figure 3.4 summarizes the results obtained for salt-and-pepper noise filters (noise intensity 5 % and 50 %). In the case of 5 % noise intensity (see Figures 3.4a and 3.4c), the quality of filtering provided by the standard CGP, CGP with the CSFP (using 5 % pixels) and CGP with the ASFP is almost identical. CGP with the CSFP is 6.4 times faster than the standard CGP while CGP with the ASFP is only 2.4 times faster. For 50 % noise intensity (see Figures 3.4b and 3.4d), the proper predictor size is only 1 % of all fitness cases and CGP with this setting of the CSFP is 10.7 times faster than the standard CGP. CGP with the ASFP is 10.2 times faster.

In order to confirm that the proposed approach is able to adapt the predictor size to a given task, we plot the progress of the average number (out of 100 independent runs) of fitness cases in the top-ranked ASFP during the course of evolution with respect to the initial predictor size. It can be seen in Figures 3.4e and 3.4f that the average size of the fitness predictor converges to a specific value independently of its initial size. Furthermore, the converged average predictor size differs for each benchmark. In general, it is advantageous to begin with a lower number of fitness cases in the fitness predictor, which in some cases leads to a lower number of fitness cases evaluations and thus shortening the design time. On the other hand, if the initial size is too small to find an acceptable solution, it will be automatically increased without a significant impact on the run time.

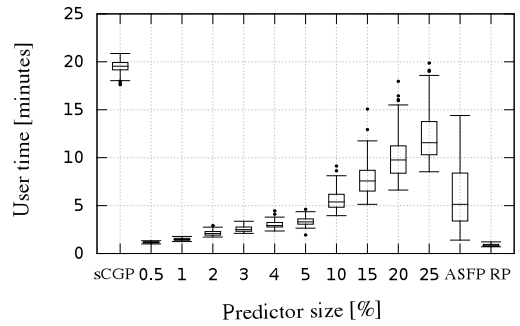
[1]The corresponding paper [G] is under review in a journal and it was not included to the thesis.



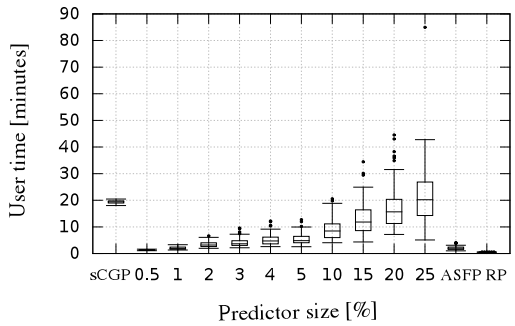
(a) Salt-and-pepper noise 5 %.



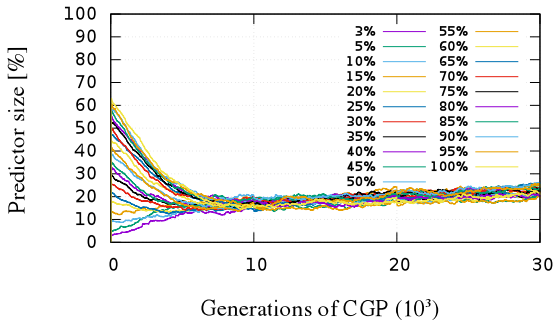
(b) Salt-and-pepper noise 50 %.



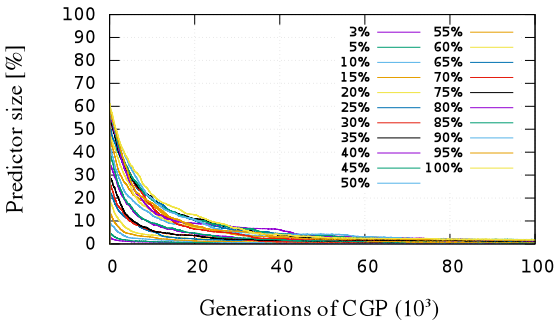
(c) Salt-and-pepper noise 5 %.



(d) Salt-and-pepper noise 50 %.



(e) Salt-and-pepper noise 5 %.



(f) Salt-and-pepper noise 50 %.

Figure 3.4: The quality of filtering (PSNR) (3.4a and 3.4b) and the time of evolution (3.4c and 3.4d) for filters evolved using the standard CGP (sCGP), CGP with the CSFP (predictor size $0.5 - 25$ % of the complete training set), CGP with ASFP, and CGP with random constant fitness predictor (RP) – the boxplots created from 100 runs with $3 \cdot 10^4$ generations each. Figures 3.4e and 3.4f are the convergence curves for various initial ASFP sizes averaged from 100 independent runs.

3.1.8 HW Implementation

Despite the acceleration provided by the fitness predictor coevolution, the CGP design is still computationally very expensive method. Inspired by the FPGA accelerator of CGP, a hardware platform for parallel coevolutionary CGP has been proposed. A two-population coevolutionary algorithm running on dual MicroBlaze soft processor system was accelerated using custom peripheral based on the VRC approach. The full pipelined VRC along with a special fitness cases memory enables very efficient fitness calculation. The implementation is described in paper [E].

Results

The performance of the hardware accelerator was experimentally evaluated in the task of evolutionary image filter design. The proposed hardware-accelerated coevolutionary CGP was compared with hardware-accelerated standard CGP and with a highly optimized software implementation of coevolutionary CGP. It was shown that by using this platform, the execution time of evolutionary design using CGP can be significantly reduced (up to 58 times). The detailed comparisons are given in paper [E].

3.2 List of Publications with Abstracts

This chapter presents the list of papers with abstracts for all papers included in this thesis. The list of other papers of the author relevant to this thesis is also given.

Papers Included in Thesis

[A] Michaela Sikulova and Lukas Sekanina. Coevolution in cartesian genetic programming. In *Genetic Programming - 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings*, volume 7244 of *Lecture Notes in Computer Science*, pages 182–193. Springer, 2012.

The author participation: 60 %

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.

[B] Michaela Sikulova and Lukas Sekanina. Acceleration of evolutionary image filter design using coevolution in cartesian GP. In *Parallel Problem Solving from Nature - PPSN XII - 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I*, volume 7491 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 2012.

The author participation: 60 %

Abstract: The aim of this paper is to accelerate the task of evolutionary image filter design using coevolution of candidate filters and training vectors subsets. Two

coevolutionary methods are implemented and compared for this task in the framework of Cartesian Genetic Programming (CGP). Experimental results show that only 15 – 20 % of original test vectors are needed to find an image filter which provides the same quality of filtering as the best filter evolved using the standard CGP which utilizes the whole training set. Moreover, the median time of evolution was reduced 2.99 times in comparison with the standard CGP.

[C] Michaela Sikulova, Jiri Hulva, and Lukas Sekanina. Indirectly encoded fitness predictors coevolved with cartesian programs. In *Genetic Programming - 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings*, volume 9025 of *Lecture Notes in Computer Science*, pages 113–125. Springer, 2015.

The author participation: 60 %

Abstract: We investigate coevolutionary Cartesian genetic programming that coevolves fitness predictors in order to diminish the number of target objective vector (fitness case) evaluations, needed to obtain a satisfactory solution, to reduce the computational cost of evolution. This paper introduces the use of coevolution of fitness predictors in CGP with a new type of indirectly encoded predictors. Indirectly encoded predictors are operated using the CGP and provide a variable number of fitness cases used for solution evaluation during the coevolution. It is shown in 5 symbolic regression problems that the proposed predictors are able to adapt the size of fitness cases array in response to a particular training data set.

[D] Michal Wiglasz and Michaela Drahosova. Plastic fitness predictors coevolved with cartesian programs. In *Genetic Programming - 19th European Conference, EuroGP 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings*, volume 9594 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2016.

The author participation: 50 %

Abstract: Coevolution of fitness predictors, which are a small sample of all training data for a particular task, was successfully used to reduce the computational cost of the design performed by Cartesian genetic programming. However, it is necessary to specify the most advantageous number of fitness cases in predictors, which differs from task to task. This paper proposes to introduce a new type of directly encoded fitness predictors inspired by the principles of phenotypic plasticity. The size of the coevolved fitness predictor is adapted in response to the phase of learning that the program evolution goes through. It is shown in 5 symbolic regression tasks that the proposed algorithm is able to adapt the number of fitness cases in predictors in response to the solved task and the program evolution flow.

[E] Radek Hrbacek and Michaela Sikulova. Coevolutionary cartesian genetic programming in FPGA. In *Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems: Advances in Artificial Life, ECAL 2013, Sicily, Italy, September 2-6, 2013*, pages 431–438. MIT Press, 2013.

The author participation: 40 %

Abstract: In this paper, a hardware platform for coevolutionary cartesian genetic programming is proposed. The proposed two-population coevolutionary algorithm involves the implementation of search algorithms in two MicroBlaze soft processors (one for each population) interconnected by the AXI bus in Xilinx Virtex

6 FPGA. Candidate programs are evaluated in a domain-specific virtual reconfigurable circuit incorporated into custom MicroBlaze peripheral. Experimental results in the task of evolutionary image filter design show that we can achieve a significant speed-up (up to 58) in comparison with a highly optimized software implementation.

Other Relevant Papers

[F] Michaela Sikulova, Gergely Komjathy, and Lukas Sekanina. Towards compositional coevolution in evolutionary circuit design. In *2014 IEEE International Conference on Evolvable Systems, ICES 2014, Orlando, FL, USA, December 9-12, 2014*, pages 157–164. IEEE, 2014.

The author participation: 55 %

[G] Michaela Drahosova, Lukas Sekanina, and Michal Wiglasz. On Coevolved Fitness Predictors in Cartesian Genetic Programming. [Under review in *Evolutionary Computation*, 2017.]

The author participation: 50 %

Chapter 4

Conclusions

The research presented in this thesis has addressed the problem of the scalability of evaluation in Cartesian genetic programming. We proposed to apply coevolution of fitness predictors in the evolutionary design performed by CGP in order to reduce the expensive fitness evaluations. We investigated several scenarios how to adapt the concept of coevolution for CGP.

Chapter 1.1 formulated the following hypothesis:

Research hypothesis: A properly designed mechanism employing coevolution of cartesian programs and tests is able to reduce the computational cost of CGP for considered applications in comparison with the standard CGP.

We believe that this thesis experimentally confirmed the hypothesis through 5 symbolic regression benchmarks and the low-level image filter design, which is considered as a real-world problem often approached by CGP. Table 4.1 shows the acceleration obtained using proposed approaches in these particular tasks.

Chapter 1.1 also formulated three research goals. They are given bellow together with a short comment how they were fulfilled.

Goal 1: To study the literature to get an overview of the state of the art in GP, CGP and coevolutionary principles.

The theoretical basis relevant to this thesis has been briefly surveyed in Chapter 2.

Goal 2: To propose and implement CGP that uses coevolutionary principles.

- **2.1:** To solve how to choose such fitness cases that are effective in driving the search process towards a solution.
- **2.2:** To solve how to select a sufficient number of fitness cases.
- **2.3:** To evaluate the proposed approach using selected case studies, especially symbolic regression tasks.

Of the three goals, Goal 2 has been the most challenging one to find a solution and was directly addressed in all papers, except paper [E]. Paper [A] introduced our adaptation of coevolutionary principles to CGP and presented its ability to choose those fitness cases that enable CGP to find a good solution. Papers [C] and [D] are very important contributions to fulfill this goal because they introduced approaches capable of (together with choosing

proper fitness cases) determining a sufficient number of fitness cases automatically during the evolution. The proposed approaches have been evaluated in five symbolic regression benchmarks [A,C,D] and the design of low-level image filters [B].

Goal 3: To propose coevolutionary CGP suitable for FPGAs.

The basic approach presented in paper [A] and later simplified in paper [B] in terms of population synchronization and communication can be considered as a suitable solution for implementation in FPGAs because of simple fitness predictor encoding and fitness predictor evolution via basic GA. Paper [E] addressed the challenge of applying coevolutionary CGP in resource-constrained devices, in particular in FPGA. Coevolutionary CGP implemented into FPGA was evaluated in the design of image filters and provided a significant speedup with respect to a highly-optimized software implementation.

4.1 Summary of Main Contributions

This chapter summarizes the main contributions of this thesis.

Contribution 1: The use of coevolution applied to reduce the computational cost of CGP has been a novel approach.

As the coevolution applied to reduce the computational cost of CGP has not been considered until the research presented in this thesis, paper [A] has been the first published coevolutionary method reducing the cost of CGP. This paper has provided a blueprint of how to set up the coevolutionary interactions to make coevolutionary CGP produce sufficient results.

Contribution 2: Using coevolution of fitness predictors in CGP reduces the execution time in comparison with standard CGP.

We have consistently shown in all papers included in this thesis that using coevolution of fitness predictors, the number of fitness cases evaluations necessary to produce a satisfactory solution has been significantly reduced compared to the standard CGP (without coevolution). The summary of the number of fitness case evaluations for investigated benchmarks and all compared approaches is shown in Table 4.1. The speedup compared to the standard CGP is not as significant as the fitness cases evaluation reduction because of the overhead associated with coevolution.

In paper [A], where fitness predictors were represented as a constant-size array of pointers to fitness cases, speedup of 2.0 – 5.4 was reported in comparison with the standard CGP for 5 symbolic regression benchmarks and the results were quite competitive with the tree-based GP [57] in terms of the number of fitness evaluations needed to obtain a satisfactory solution. While this approach required to set up the number of fitness cases before executing it, the predictor size is fixed during the whole coevolutionary process. This approach required a large number of experiments in order to find the most suitable predictor size for a particular task. This limitation doesn't matter in hardware implementation, because it used to be problem-specific.

While using coevolutionary CGP to a new task, the approaches proposed in papers [C,D] are able to automatically adapt to the solved task. The number of fitness case evaluations needed to obtain satisfactory solutions for symbolic regression benchmarks (F1 – F5) and

Table 4.1: Comparison of standard CGP (sCGP), coevolutionary CGP with constant-size fitness predictor (CSFP, papers [A,B]), adaptive-size fitness predictor (ASFP, paper [D]) and coevolutionary CGP with indirectly-encoded fitness predictor (IEFP, paper [C]). For each benchmark, the best result is marked in bold font.

(a) Symbolic regression benchmarks F1 – F5 (according to [A]).

	Approach	F1	F2	F3	F4	F5
Success rate	sCGP	100 %	100 %	91 %	5 %	27 %
	CSFP	100 %	100 %	100 %	100 %	92 %
	ASFP	100 %	100 %	100 %	99 %	87 %
	IEFP	100 %	100 %	100 %	100 %	90 %
Fitness case evaluations to converge (median)	sCGP	$2.0 \cdot 10^7$	$7.4 \cdot 10^7$	$2.8 \cdot 10^8$	$9.9 \cdot 10^9$	$7.8 \cdot 10^9$
	CSFP	$\mathbf{4.4 \cdot 10^5}$	$3.0 \cdot 10^6$	$7.4 \cdot 10^6$	$2.3 \cdot 10^9$	$2.1 \cdot 10^9$
	ASFP	$6.2 \cdot 10^5$	$\mathbf{1.2 \cdot 10^6}$	$\mathbf{4.6 \cdot 10^6}$	$1.4 \cdot 10^9$	$1.5 \cdot 10^9$
	IEFP	$7.4 \cdot 10^5$	$1.6 \cdot 10^6$	$1.9 \cdot 10^7$	$\mathbf{8.0 \cdot 10^8}$	$\mathbf{8.7 \cdot 10^8}$
Speedup compared to CGP_{STD} (median)	CSFP	2.0	**2.6**	5.4	2.5	5.1
	ASFP	**2.8**	2.4	3.8	9.7	3.8
	IEFP	1.9	1.1	**8.3**	**16.3**	**11.9**

(b) Salt-and-pepper noise-suppressing filter design (noise intensity 10 – 50 %).

	Approach	S&P 10%	S&P 20%	S&P 30%	S&P 40%	S&P 50%
PSNR [dB] (median)	sCGP	27.9	24.8	22.3	20.1	18.2
	CSFP	**28.6**	**25.1**	**22.5**	**20.2**	**18.3**
	ASFP	28.2	24.8	22.3	19.8	18.0
Fitness case evaluations to converge (median)	sCGP	$4.5 \cdot 10^{10}$	$4.5 \cdot 10^{10}$	$4.5 \cdot 10^{10}$	$4.5 \cdot 10^{10}$	$4.5 \cdot 10^{10}$
	CSFP	$\mathbf{2.2 \cdot 10^{10}}$	$2.3 \cdot 10^{10}$	$2.5 \cdot 10^{10}$	$2.6 \cdot 10^{10}$	$2.9 \cdot 10^{10}$
	ASFP	$2.3 \cdot 10^{10}$	$\mathbf{1.1 \cdot 10^{10}}$	$\mathbf{3.5 \cdot 10^9}$	$\mathbf{2.2 \cdot 10^9}$	$\mathbf{1.5 \cdot 10^9}$
Speedup comp. to CGP_{STD} (median)	CSFP	4.1	2.0	2.0	2.9	4.3
	ASFP	**6.2**	**7.3**	**7.7**	**7.8**	**7.3**

the speedup obtained for each proposed approach (compared to the standard CGP) are shown in Table 4.1a.

The same coevolutionary CGP has been used in the evolutionary design of salt-and-pepper noise-suppressing filters. The number of fitness case evaluations needed to obtain satisfactory solutions and the speedup obtained in comparison to the standard CGP for this application is given in Table 4.1b.

The approach IEFP presented in paper [C] seems to overcome other approaches under our experimental setup, because of low cost of coevolution. However, it should be pointed out, that during the evolution of the IEFPs, large fitness predictors can occur and must be evaluated (and then refused for a larger size), and thus plenty of fitness case evaluations are wasted. In order to prevent this overhead, the limitation of fitness predictor size was established to reduce the evaluation cost. Hence, the approach ASFP presented in [D] should be recommended, while applying coevolutionary CGP to a new, unknown, task.

Contribution 3: Using coevolution of fitness predictors in CGP, the severity of local optima can be reduced.

Papers [A,C,D] have also shown that standard CGP in some runs inclined to reach only a local optimum and wasn't able to leave it. The use of proposed approaches have reduced this problem. Table 4.1a summarizes the ratio of successfully reached solutions in the case of symbolic regression benchmarks.

4.2 Possibilities of Future Research

This chapter suggests some directions for the research following on this thesis.

- The CGP has been applied in many different problem domains, predominantly in evolutionary design and optimization of logic networks. Hence the proposed method should also be useful for evolvable hardware purposes.
- Proposed approach is designed with respect to a changing environment in which both populations entail each other. It would be interesting to employ proposed approach in the task with outwardly changing environment, i.e. using a variable training set, such as in systems enabling autonomous adaptation, in order to speedup the adaptation process.
- With small modifications, the hardware platform introduced in paper [E], can be used to effectively evolve other digital circuits using coevolutionary CGP.
- As hardware implementations of CGP are available, the methods proposed in paper [C] or [D] can be combined with a hardware accelerator of fitness evaluation in an embedded HW/SW system, for instance, implementing adaptive video filtering.
- As mentioned in Chapter 3.1, CGP has some important differences, in terms of the use of very small populations and many more generations to obtain a solution, compared to the tree-based GP. This feature is the main aspect to consider while designing the coevolutionary interactions in CGP. Proposed approach has shown to be a relevant guidance to apply other type of coevolution in CGP, in particular the coevolution applied to compositional problems. Paper [F], which is related to this thesis, has offered a blueprint of how to apply the coevolutionary approach to divide and conquer method in order to solve the scalability of representations, addressed in section 2.1.5. The design of more complex circuits using compositional coevolution in CGP would also be interesting.
- In the context of approximate computing, the proposed coevolutionary CGP could co-evolve approximate components of a given system in such a way that errors produced by one components are compensated by the other component.

Bibliography

[1] Sameer H. Al-Sakran, John R. Koza, and Lee W. Jones. Automated re-invention of a previously patented optical lens system using genetic programming. In *Genetic Programming, 8th European Conference, EuroGP2005, Lausanne, Switzerland, March 30 - April 1, 2005, Proceedings*, volume 3447 of *Lecture Notes in Computer Science*, pages 25–37. Springer, 2005.

[2] Dirk V. Arnold and Hans-Georg Beyer. A comparison of evolution strategies with other direct search methods in the presence of noise. *Computational Optimization and Applications*, 24(1):135–159, 2003.

[3] Mark J. Baldwin. A new factor in evolution. *The American Naturalist*, 30(354):441–451, 1896.

[4] Peter Bentley. *Evolutionary design by computers*. Morgan Kaufmann, 1999.

[5] Markus Brameier and Wolfgang Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, 5(1):17–26, 2001.

[6] Anthony Bucci. *Emergent geometric organization and informative dimensions in coevolutionary algorithms*. PhD thesis, Brandeis University, 2007.

[7] Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985*, pages 183–187. Lawrence Erlbaum Associates, 1985.

[8] Edwin D. de Jong. The maxsolve algorithm for coevolution. In *Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, 2005*, pages 483–489. ACM, 2005.

[9] Edwin D. de Jong and Anthony Bucci. Objective set compression. In *Multiobjective Problem Solving from Nature: From Concepts to Applications*, Natural Computing Series, pages 357–376. Springer Berlin Heidelberg, 2008.

[10] Edwin D. de Jong and Jordan B. Pollack. Ideal evaluation from coevolution. *Evolutionary Computation*, 12(2):159–192, 2004.

[11] Roland Dobai. Evolutionary on-line synthesis of hardware accelerators for software modules in reconfigurable embedded systems. In *24th International Conference on Field Programmable Logic and Applications, FPL 2014, Munich, Germany, 2-4 September, 2014*, pages 1–6. IEEE, 2014.

[12] Brad Dolin, Forrest H. Bennett III, and Eleanor G. Rieffel. Co-evolving an effective fitness sample: experiments in symbolic regression and distributed robot control. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC), March 10-14, 2002, Madrid, Spain*, pages 553–559. ACM, 2002.

[13] Jens-Uwe Dolinsky, Ian Jenkinson, and Gary Colquhoun. Application of genetic programming to the calibration of industrial robots. *Computers in Industry*, 58(3):255–264, 2007.

[14] Kai Olav Ellefsen. Balancing the costs and benefits of learning ability. In *Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems: Advances in Artificial Life, ECAL 2013, Sicily, Italy, September 2-6, 2013*, pages 292–299. MIT Press, 2013.

[15] Kai Olav Ellefsen. Evolved sensitive periods in learning. In *Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems: Advances in Artificial Life, ECAL 2013, Sicily, Italy, September 2-6, 2013*, pages 409–416. MIT Press, 2013.

[16] Umer Farooq, Zied Marrakchi, and Habib Mehrez. FPGA architectures: An overview. In *Tree-based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*, pages 7–48. Springer New York, 2012.

[17] Christian Gagné and Marc Parizeau. Coevolution of nearest neighbor classifiers. *International Journal of Pattern Recognition and Artificial Inteligence*, 21(5):921–946, 2007.

[18] Chris Gathercole and Peter Ross. Dynamic training subset selection for supervised learning in genetic programming. In *Parallel Problem Solving from Nature - PPSN III, International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature, Jerusalem, Israel, October 9-14, 1994, Proceedings*, volume 866 of *Lecture Notes in Computer Science*, pages 312–321. Springer, 1994.

[19] Chris Gathercole and Peter Ross. Tackling the Boolean even N parity problem with genetic programming and limited-error fitness. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 119–127. Morgan Kaufmann, 1997.

[20] Mario Giacobini, Marco Tomassini, and Leonardo Vanneschi. Limiting the number of fitness cases in genetic programming using statistics. In *Parallel Problem Solving from Nature - PPSN VII, 7th International Conference, Granada, Spain, September 7-11, 2002, Proceedings*, volume 2439 of *Lecture Notes in Computer Science*, pages 371–380. Springer, 2002.

[21] Brian W. Goldman and William F. Punch. Analysis of cartesian genetic programming's evolutionary mechanisms. *IEEE Transactions on Evolutionary Computation*, 19(3):359–373, 2015.

[22] Timothy G. W. Gordon and Peter J. Bentley. Towards development in evolvable hardware. In *4th NASA / DoD Workshop on Evolvable Hardware (EH 2002), 15-18 July 2002, Alexandria, VA, USA*, pages 241–250. IEEE Computer Society, 2002.

[23] Simon L. Harding and Wolfgang Banzhaf. Hardware acceleration for CGP: graphics processing units. In *Cartesian Genetic Programming*, Natural Computing Series, pages 231–253. Springer, 2011.

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

[25] Daniel Howard and Simon C. Roberts. Incident detection on highways. In *Genetic Programming Theory and Practice II*, volume 8 of *Genetic Programming*, pages 263–282. Springer US, 2005.

[26] Kosuke Imamura, James A. Foster, and Axel W. Krings. The test vector problem and limitations to evolving digital circuits. In *2nd NASA / DoD Workshop on Evolvable Hardware (EH 2000), 13-15 July 2000, Palo Alto, CA, USA*, pages 75–80. IEEE Computer Society, 2000.

[27] Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Comput.*, 9(1):3–12, 2005.

[28] Yaochu Jin and Bernhard Sendhoff. Reducing fitness evaluations using clustering techniques and neural network ensembles. In *Genetic and Evolutionary Computation - GECCO 2004, Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004, Proceedings, Part I*, volume 3102 of *Lecture Notes in Computer Science*, pages 688–699. Springer, 2004.

[29] Brad Johanson and Riccardo Poli. Gp-music: An interactive genetic programming system for music generation with automated fitness raters. Technical report, University of Birmingham, Cognitive Science Research Centre, 1998.

[30] Gul Muhammad Khan, Julian Francis Miller, and David M. Halliday. Coevolution of neuro-developmental programs that play checkers. In *Evolvable Systems: From Biology to Hardware, 8th International Conference, ICES 2008, Prague, Czech Republic, September 21-24, 2008. Proceedings*, volume 5216 of *Lecture Notes in Computer Science*, pages 352–361. Springer, 2008.

[31] John R. Koza. *Genetic programming - on the programming of computers by means of natural selection.* Complex adaptive systems. MIT Press, 1993.

[32] John R. Koza. *Genetic programming 2 - automatic discovery of reusable programs.* Complex adaptive systems. MIT Press, 1994.

[33] John R. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284, 2010.

[34] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann Publishers, 1999.

[35] John R. Koza, Martin A. Keane, Matthew J. Streeter, Sameer H. Al-Sakran, and Lee W. Jones. *Human-Competitive Evolvable Hardware Created by Means of Genetic Programming*, pages 173–197. Springer US, 2006.

[36] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[37] William B. Langdon and Bernard F. Buxton. Genetic programming for mining DNA chip data from cancer patients. *Genetic Programming and Evolvable Machines*, 5(3):251–257, 2004.

[38] Christian Lasarczyk, Peter Dittrich, and Wolfgang Banzhaf. Dynamic subset selection based on a fitness case topology. *Evolutionary Computation*, 12(2):223–242, 2004.

[39] Jason D. Lohn, Gregory S. Hornby, and Derek S. Linden. *An Evolved Antenna for Deployment on Nasa's Space Technology 5 Mission*, volume 8 of *Genetic Programming*, pages 301–315. Springer US, 2005.

[40] Roberto R. F. Mendes, Fabricio de B. Voznika, Alex Alves Freitas, and Júlio C. Nievola. Discovering fuzzy classification rules with genetic programming and co-evolution. In *Principles of Data Mining and Knowledge Discovery, 5th European Conference, PKDD 2001, Freiburg, Germany, September 3-5, 2001, Proceedings*, volume 2168 of *Lecture Notes in Computer Science*, pages 314–325. Springer, 2001.

[41] Julian F. Miller, editor. *Cartesian Genetic Programming*. Natural Computing Series. Springer, 2011.

[42] Julian F. Miller and Stephen L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.

[43] Julian F. Miller and Peter Thomson. Cartesian genetic programming. In *Genetic Programming, European Conference, Edinburgh, Scotland, UK, April 15-16, 2000, Proceedings*, volume 1802 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2000.

[44] German A. Monroy, Kenneth O. Stanley, and Risto Miikkulainen. Coevolution of neural networks using a layered pareto archive. In *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006*, pages 329–336. ACM, 2006.

[45] Masahiro Murakawa, Shuji Yoshizawa, Isamu Kajitani, Tatsumi Furuya, Masaya Iwata, and Tetsuya Higuchi. Hardware evolution at function level. In *Parallel Problem Solving from Nature - PPSN IV, International Conference on Evolutionary Computation. The 4th International Conference on Parallel Problem Solving from Nature, Berlin, Germany, September 22-26, 1996, Proceedings*, volume 1141 of *Lecture Notes in Computer Science*, pages 62–71. Springer, 1996.

[46] Peter Nordin and Wolfgang Banzhaf. 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, 1997.

[47] Frans A. Oliehoek, Edwin D. de Jong, and Nikos A. Vlassis. The parallel nash memory for asymmetric games. In *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006*, pages 337–344. ACM, 2006.

[48] Ludo Pagie and Paulien Hogeweg. Evolutionary consequences of coevolving targets. *Evolutionary Computation*, 5(4):401–418, 1997.

[49] Justyna Petke, Saemundur Haraldsson, Mark Harman, William B. Langdon, David White, and John Woodward. Genetic improvement of software: a comprehensive survey. *IEEE Transactions on Evolutionary Computation*, PP(99):1–18, 2017.

[50] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. lulu.com, 2008.

[51] Elena Popovici, Anthony Bucci, R. Paul Wiegand, and Edwin D. de Jong. Coevolutionary principles. In *Handbook of Natural Computing*, pages 987–1033. Springer, 2012.

[52] Mitchell A. Potter and Kenneth A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[53] Rommel G. Regis and Christine A. Shoemaker. Constrained global optimization of expensive black box functions using radial basis functions. *Journal of Global Optimization*, 31(1):153–171, 2005.

[54] Christopher D. Rosin and Richard K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

[55] Ruben Salvador, Andrés Otero, Javier Mora, Eduardo de la Torre, Teresa Riesgo, and Lukas Sekanina. Self-reconfigurable evolvable hardware system for adaptive image processing. *IEEE Trans. Computers*, 62(8):1481–1493, 2013.

[56] Michael D. Schmidt and Hod Lipson. Co-evolving fitness predictors for accelerating and reducing evaluations. In *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, pages 113–130. Springer, 2006.

[57] Michael D. Schmidt and Hod Lipson. Coevolution of fitness predictors. *IEEE Transactions on Evolutionary Computation*, 12(6):736–749, 2008.

[58] Tom Seaton, Julian F. Miller, and Tim Clarke. Semantic bias in program coevolution. In *Genetic Programming - 16th European Conference, EuroGP 2013, Vienna, Austria, April 3-5, 2013. Proceedings*, volume 7831 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 2013.

[59] Lukas Sekanina. *Evolvable Components - From Theory to Hardware Implementations*. Natural Computing Series. Springer Verlag, 2003.

[60] Lukas Sekanina, Simon L. Harding, Wolfgang Banzhaf, and Taras Kowaliw. Image processing and CGP. In *Cartesian Genetic Programming*, Natural Computing Series, pages 181–215. Springer, 2011.

[61] Lukas Sekanina, James Alfred Walker, Paul Kaufmann, and Marco Platzner. Evolution of electronic circuits. In *Cartesian Genetic Programming*, Natural Computing Series, pages 125–179. Springer, 2011.

[62] A. P. Shanthi and Ranjani Parthasarathi. Practical and scalable evolution of digital circuits. *Applied Soft Computing*, 9(2):618–624, 2009.

[63] Min Shi. Empirical analysis of cooperative coevolution using blind decomposition. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Companion Material Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 141–142. ACM, 2011.

[64] Min Shi and Haifeng Wu. Pareto cooperative coevolutionary genetic algorithm using reference sharing collaboration. In *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*, pages 867–874. ACM, 2009.

[65] Kenneth Owen Stanley. *Efficient Evolution of Neural Networks Through Complexification.* PhD thesis, The University of Texas at Austin, 2004.

[66] Emanuele Stomeo, Tatiana Kalganova, and Cyrille Lambert. Generalized disjunction decomposition for the evolution of programmable logic array structures. In *First NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2006), 15-18 June 2006, Istanbul, Turkey*, pages 179–185. IEEE Computer Society, 2006.

[67] Jim Tørresen. A divide-and-conquer approach to evolvable hardware. In *Evolvable Systems: From Biology to Hardware, Second International Conference, ICES 98, Lausanne, Switzerland, September 23-25, 1998, Proceedings*, volume 1478 of *Lecture Notes in Computer Science*, pages 57–65. Springer, 1998.

[68] Leonardo Trujillo and Gustavo Olague. Using evolution to learn how to perform interest point detection. In *18th International Conference on Pattern Recognition (ICPR 2006), 20-24 August 2006, Hong Kong, China*, pages 211–214. IEEE Computer Society, 2006.

[69] Leonardo Vanneschi and Riccardo Poli. Genetic programming - introduction, applications, theory and open issues. In *Handbook of Natural Computing*, pages 709–739. Springer, 2012.

[70] Zdenek Vasicek and Lukas Sekanina. Hardware accelerator of cartesian genetic programming with multiple fitness units. *Computing and Informatics*, 29(6):1359–1371, 2010.

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

[72] Ekaterina Vladislavleva, Guido Smits, and Dick den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, 2009.

[73] James Alfred Walker and Julian Francis Miller. The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 12(4):397–417, 2008.

[74] Jin Wang, Q. S. Chen, and Chong Ho Lee. Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware. *IET Computers and Digital Techniques*, 2(5):386–400, 2008.

[75] David Robert White, James McDermott, Mauro Castelli, Luca Manzoni, Brian W. Goldman, Gabriel Kronberger, Wojciech Jaskowski, Una-May O'Reilly, and Sean Luke. Better GP benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29, 2013.

[76] Darrell Whitley and Andrew M. Sutton. Genetic algorithms - A survey of models and methods. In *Handbook of Natural Computing*, pages 637–671. Springer, 2012.

[77] Huayang Xie, Mengjie Zhang, and Peter Andreae. Genetic programming for automatic stress detection in spoken english. In *Applications of Evolutionary Computing, EvoWorkshops 2006, Budapest, Hungary, April 10-12, 2006, Proceedings*, volume 3907 of *Lecture Notes in Computer Science*, pages 460–471. Springer, 2006.

[78] Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008.

[79] Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

[80] Mengjie Zhang and William D. Smart. Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification. *Pattern Recognition Letters*, 27(11):1266–1274, 2006.

Papers

Paper A

Coevolution in Cartesian Genetic Programming

Michaela Sikulova and Lukas Sekanina

In *Genetic Programming - 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings*, volume 7244 of *Lecture Notes in Computer Science*, pages 182–193. Springer, 2012.

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

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 182–193, 2012.

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

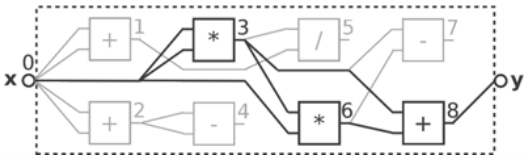


Fig. 1. A candidate program in CGP, where $l = 4$, $n_c = 4$, $n_r = 2$, $n_i = 1$, $n_o = 1$, $n_a = 2$, $\Gamma = \{+ (1), - (2), * (3), / (4)\}$ and chromosome is: $0, 0, 1;\ 0, 0, 1;\ 0, 0, 3;\ 2, 2, 2;\ 3, 1, 4;\ 3, 0, 3;\ 3, 6, 2;\ 3, 6, 1;\ 8$

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.

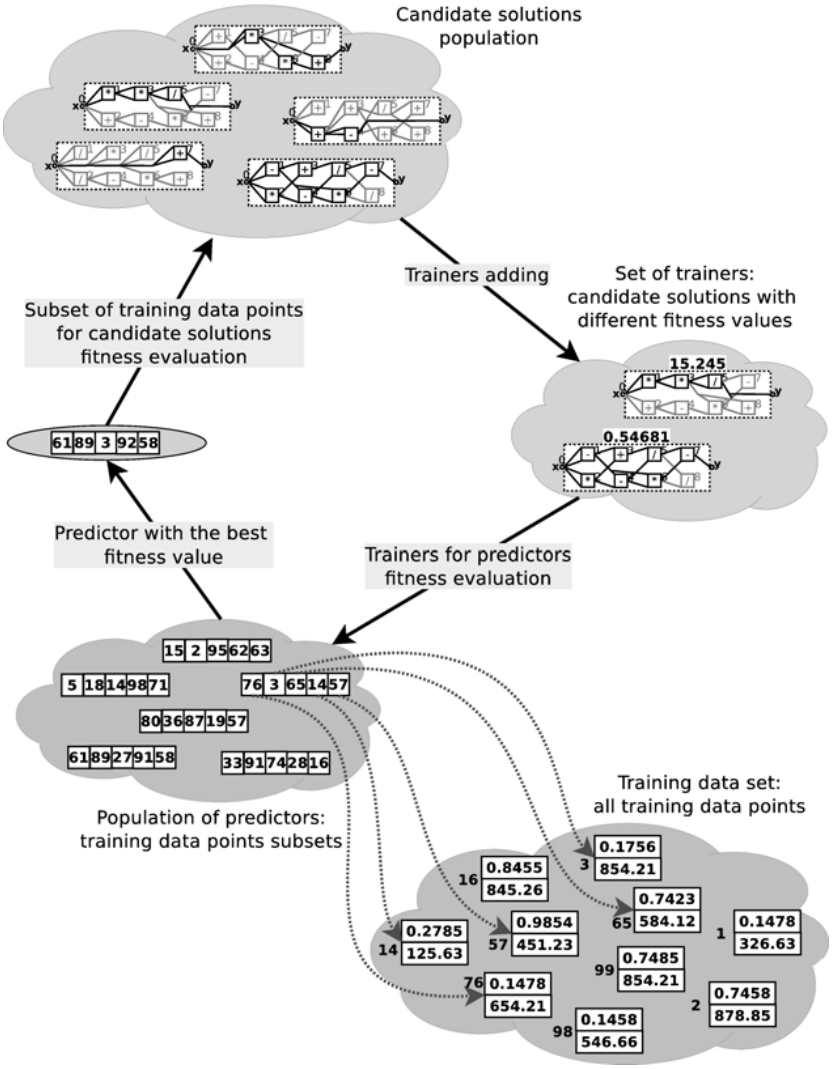


Fig. 2. Coevolution of candidate programs and fitness predictors

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,

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

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

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

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

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.

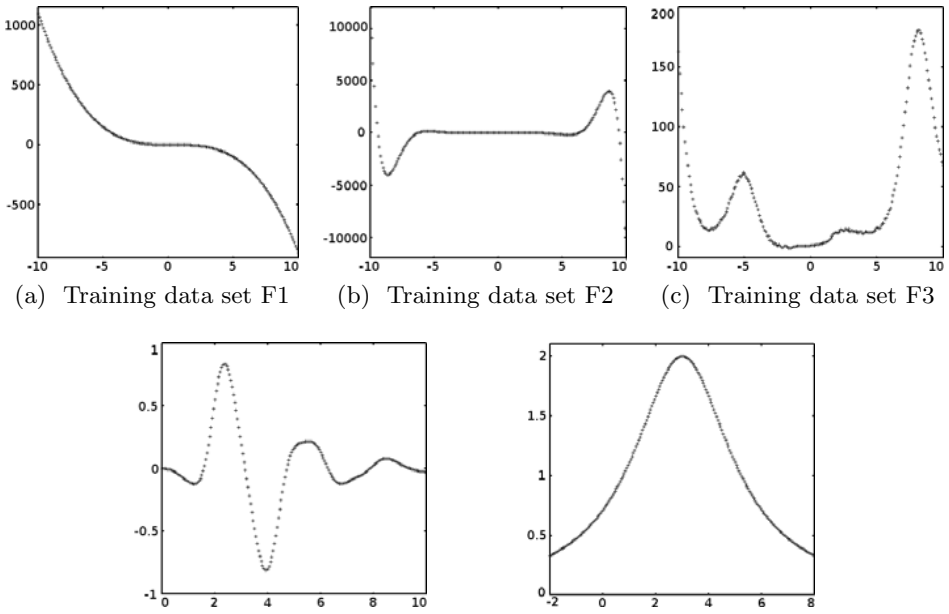
The second thread performs the evolution of fitness predictors. The fitness values of all fitness predictors are evaluated using trainers. The best rated predictor is selected and stored to shared memory. The next step involves creating of a new generation of fitness predictors by means of GA operators. 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 rightmost 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.

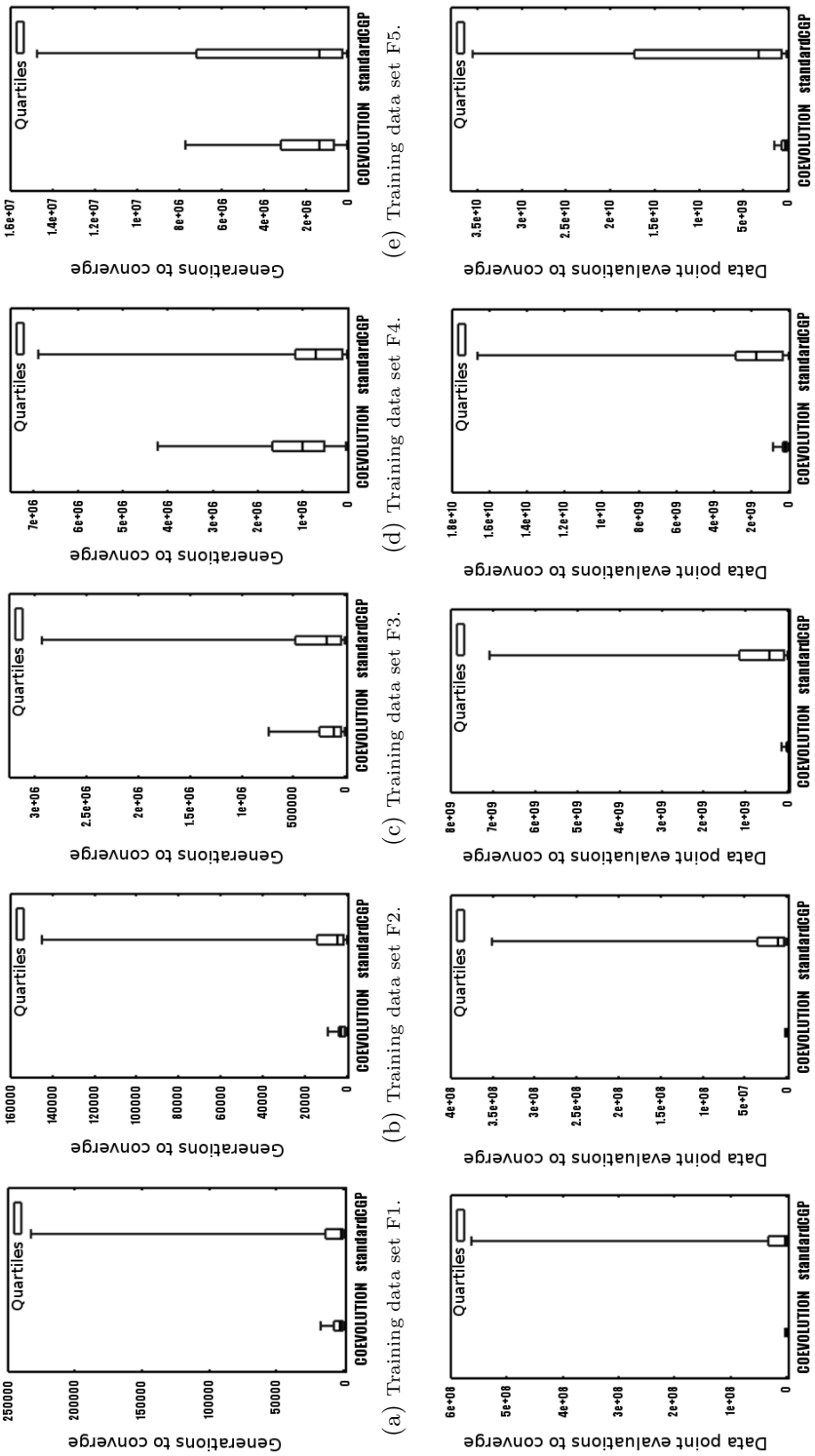
4.3 Comparison of Coevolving CGP with Standard CGP

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 %



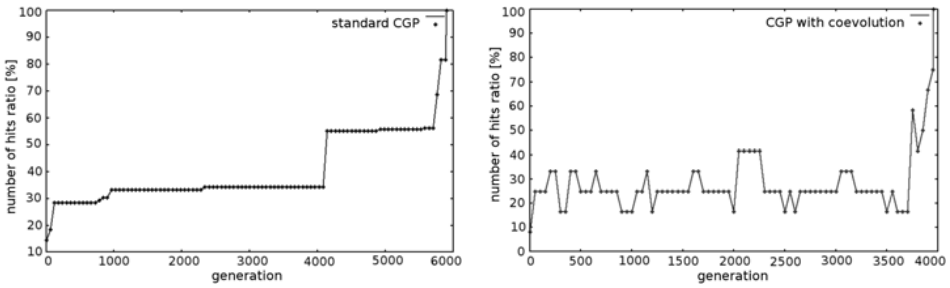
(a) Training data set F1. (b) Training data set F2. (c) Training data set F3. (d) Training data set F4. (e) Training data set F5.

(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



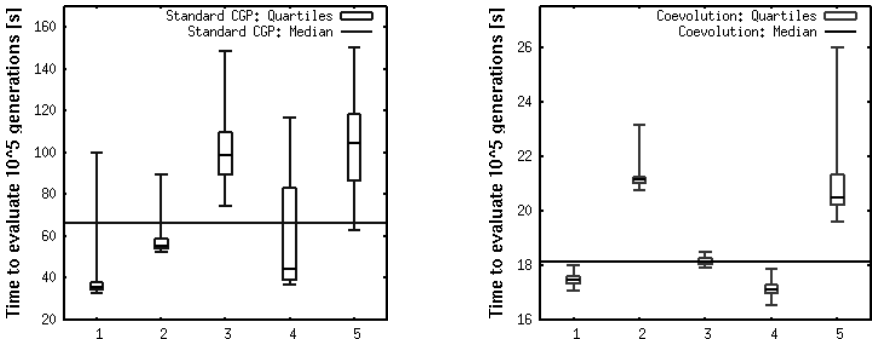
(a) Standard CGP. (b) CGP with coevolution.

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™ 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`

Paper B

Acceleration of Evolutionary Image Filter Design Using Coevolution in Cartesian GP

Michaela Sikulova and Lukas Sekanina

In *Parallel Problem Solving from Nature - PPSN XII - 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I*, volume 7491 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 2012.

Acceleration of Evolutionary Image Filter Design Using Coevolution in Cartesian GP

Michaela Sikulova and Lukas 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. The aim of this work is to accelerate the task of evolutionary image filter design using coevolution of candidate filters and training vectors subsets. Two coevolutionary methods are implemented and compared for this task in the framework of Cartesian Genetic Programming (CGP). Experimental results show that only 15–20% of original training vectors are needed to find an image filter which provides the same quality of filtering as the best filter evolved using the standard CGP which utilizes the whole training set. Moreover, the median time of evolution was reduced 2.99 times in comparison with the standard CGP.

1 Introduction

Evolutionary design based on genetic programming is a very computationally-intensive design method. For example, for 36 tasks solved using Koza's genetic programming, the average population size was 3,350,000 individuals, 128.7 generations were produced in average and the average time to reaching a solution was 81.9 hours [3]. It also holds for the evolutionary design of image filters which has been performed by Cartesian Genetic Programming (CGP). The most time consuming procedure is the fitness calculation where tens of thousands of pixels in training set (the so-called target objective vectors, TOVs) have to be evaluated in order to obtain a single fitness value. A single run is typically finished after 200 thousands candidate filter evaluations. However, for the cost of runtime, very efficient image filters were evolved, often beating conventional designs in terms of the filtering quality as well as the area required on a chip [9]. In order to reduce the time of evolution, various CGP accelerators have been introduced including GPU-based and FPGA-based machines [12, 11, 1].

In this paper, we propose to employ a coevolutionary algorithm running on an ordinary processor to accelerate the image filter evolution. Various coevolutionary methods have been introduced that can evolve suitable subsets of TOVs for evaluation of candidate solutions. The aim of this type of coevolution is to allow both candidate programs and TOVs subsets to improve each other automatically until a satisfactory problem solution is found. Coevolutionary algorithms with interactions between two independently evolving populations, in the "hosts" and "parasites" type relationships, were studied in many application domains [2, 7, 6, 4].

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 163–172, 2012.

In our previous work, inspired by *coevolution of fitness predictors* (CFP) [8], we applied coevolution of TOVs in CGP in order to accelerate the task of symbolic regression [10]. In this paper, we will show that the subsets of TOVs can be (substantially) smaller than original training sets in the task of image filter evolution. Consequently, the overall time of filter evolution can be significantly reduced. This paper also proposes and evaluates two strategies for top-ranked TOVs subset selection. The proposed coevolutionary algorithms will be compared with the standard CGP in the task of evolutionary design of image filters suppressing a salt-and-pepper noise.

2 CGP for Image Filter Design

The state of the art of CGP has recently been summarized in a monograph [5] which also surveys the evolutionary image filter design using CGP [9]. In CGP, candidate programs are represented in the form of directed acyclic graph, which is modeled as a matrix of $n_c \times n_r$ programmable elements (nodes). The number of primary inputs, n_i, and outputs, n_o, of the program is defined for a particular task. Each node input can be connected either to the output of a node placed in previous l columns or to one of the program inputs. 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.

As the considered filters operate over a filtering window consisting of 3×3 pixels, each candidate filter can utilize up to nine 8-bit inputs, i.e. $n_i = 9$. The filters produce a single pixel, i.e. $n_o = 1$. Table 1 gives a set of functions working over two pixels i_1 and i_2 that are typically used for image filter evolution. Figure 1 shows an example of a candidate filter and its encoding in CGP.

In this task, the search is usually performed using a simple $(1+\lambda)$ evolutionary algorithm, where $\lambda = 7$. Every new population consists of the best individual of the previous population and its λ offspring created using a mutation operator which modifies up to h integers of the chromosome. The initial population is generated randomly. The algorithm is terminated when the maximum number

Table 1. List of node functions

Code	Function	Description	Code	Function	Description
0	255	constant	8	$i_1 \gg 1$	right shift by 1
1	i_1	identity	9	$i_1 \gg 2$	right shift by 2
2	$255 - i_1$	inversion	A	$swap\,(i_1, i_2)$	swap nibbles
3	$i_1 \vee i_2$	bitwise OR	B	$i_1 + i_2$	+ (addition)
4	$\overline{i_1} \vee i_2$	bitwise $\overline{i_1}$ OR i_2	C	$\overline{i_1} +^S i_2$	+ with saturation
5	$i_1 \wedge i_2$	bitwise AND	D	$(i_1 + i_2) \gg 1$	average
6	$\overline{i_1 \wedge i_2}$	bitwise NAND	E	$max\,(i_1, i_2)$	maximum
7	$i_1 \oplus i_2$	bitwise XOR	F	$min\,(i_1, i_2)$	minimum

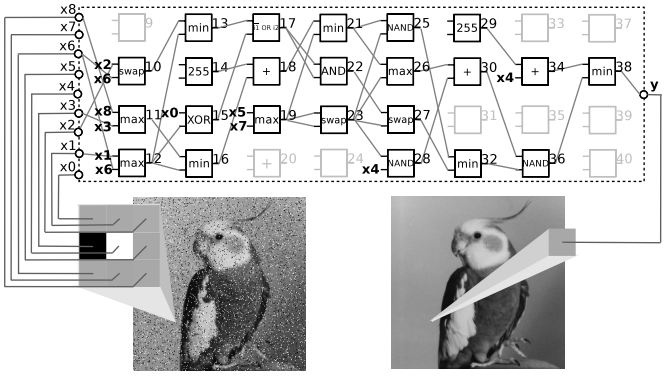


Fig. 1. A candidate filter in CGP, where $l = 1$, $n_c = 8$, $n_r = 4$, $n_i = 9$, $n_o = 1$, $n_a = 2$, Γ is according to Table 1 and the chromosome is: $6, 7, 14;\ 2, 6, 10;\ 8, 3, 14;\ 1, 6, 14;\ 10, 12, 15;\ 12, 8, 0;\ 0, 12, 7;\ 11, 12, 15;\ 13, 15, 4;\ 14, 16, 11;\ 5, 7, 14;\ 16, 5, 10;\ 18, 19, 15;\ 17, 17, 5;\ 19, 19, 10;\ 19, 18, 7;\ 21, 23, 6;\ 21, 23, 14;\ 22, 23, 10;\ 4, 23, 6;\ 25, 28, 0;\ 28, 26, 11;\ 1, 25, 11;\ 27, 25, 15;\ 31, 30, 9;\ 4, 29, 11;\ 6, 30, 8;\ 30, 32, 6;\ 33, 33, 2;\ 34, 36, 15;\ 33, 35, 11;\ 33, 34, 1;\ 38.$

of generations is exhausted, typically after 30,000 generations [9]. In the fitness function, the goal is to minimize the mean absolute error between uncorrupted version of the training image and the result of filtering of a candidate filter. This can be expressed in terms of the mean difference per pixel (MDPP) as

$$\mathrm{MDPP} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} |v\,(i,j) - w\,(i,j)|\,. \tag{1}$$

where $M \times N$ is the image size, $v(i,j)$ is a pixel value in the filtered image and $w(i,j)$ is a pixel value in the uncorrupted image.

3 Coevolution of TOVs in CGP

In the proposed coevolutionary algorithm, there are two concurrently working populations: (1) candidate programs (filters) evolving using CGP and (2) TOVs subsets evolving using a genetic algorithm. Figure 2 shows that both populations evolve simultaneously, interacting through the fitness function (using top-ranked individuals).

3.1 Population of Candidate Filters

Evolution of candidate filters is based on principles of CGP as introduced in Section 2. The fitness function for CGP is defined in terms of MDPP. There are, in fact, two fitness functions for a candidate filter. While the exact fitness function $\mathrm{MDPP}_{\mathrm{exact}}$ utilizes the complete training set (i.e. all training vectors from the training images), the partial fitness function $\mathrm{MDPP}_{\mathrm{partial}}$ uses only a selected subset. Formally,

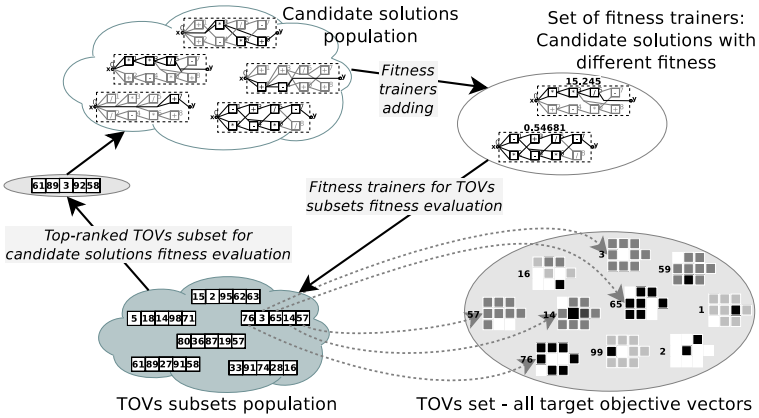


Fig. 2. Populations in coevolution of TOVs in CGP

$$\mathrm{MDPP}_{\mathrm{exact}} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} |v \ \ (i,j) - w\,(i,j)| \tag{2}$$

$$\mathrm{MDPP}_{\mathrm{partial}} = \frac{1}{K} \sum_{l=1}^{K} |v\,(l) - w\,(l)| \tag{3}$$

where $M \times N$ is the total number of TOVs in the training set and K is the number of TOVs in a training subset and l is index in the list of pointers to pixel at possition (i, j).

The set of *fitness trainers* contains several candidate filters and is used to evaluate the fitness of TOVs subsets. If the top-ranked candidate filter has a different fitness value than the top-ranked candidate filter in the previous generation, the top-ranked candidate filter replaces the oldest trainer in a circular list of trainers.

3.2 Population of TOVs Subsets

The most useful subset of TOVs is sought using a simple genetic algorithm (GA) which operates with a population of TOVs subsets. Every TOVs subset is encoded as a constant-size array of pointers to elements (i, j) in the training set. In addition to one-point crossover and mutation, a randomly generated TOVs subset replacing the worst-scored TOVs subset in each generation has been introduced as a new genetic operator of GA.

We will compare two approaches to fitness calculation. In the first one, the fitness value of a TOVs subset (i.e. the fitness predictor) is calculated using the mean absolute error of the exact fitness and partial fitness of fitness trainers s. This fitness value, which is based on the CFP approach [8], can be expressed as

$$\mathrm{f}_{\mathrm{CFP}} = \frac{1}{T} \sum_{t=1}^{T} \left|\mathrm{MDPP}_{\mathrm{partial}}\,(s(t)) - \mathrm{MDPP}_{\mathrm{exact}}\,(s(t))\right|, \tag{4}$$

where T is the number of trainers in the trainers set. The goal of TOVs subsets evolution is to minimize this f_{CFP} value, i.e. to ensure that the TOVs subset can determine (predict) the solutions' fitness values as exactly as possible.

Another approach exploits the *competitive coevolution* (CC) scheme [2]. The population of candidate filters can be viewed as "hosts" and the population of test cases as "parasites". The fitness of each candidate filter is measured by its ability to correctly solve training cases (therefore, the eq. 3 is applicable) while the fitness of the training cases is higher for those that cannot be solved well by currently evolved filters. Then the fitness value of a TOVs subset derived from Eq. 3 is defined as

$$f_{CC} = \frac{1}{T} \sum_{t=1}^{T} \frac{1}{K} \sum_{l=1}^{K} |v(l) - w(l)| \tag{5}$$

and the goal of evolution is to maximize the f_{CC} value. This type of fitness function should ensure that the TOVs subset includes TOVs that cannot be solved exactly by currently evolved filters.

3.3 Implementation

There are two threads in the coevolution implementation. One thread is responsible for candidate filters evolution using CGP, the other one for TOVs subsets evolution. This two thread model is described in Figure 3.

In the first step, the candidate filters, the trainers and the TOVs subsets are randomly initialized. Then the CGP thread waits for the first evolved TOVs subset to load it from shared memory, which is done at the beginning of every iteration of the CGP main loop. This loop continues with evaluating fitness values ($MDPP_{partial}$) of each candidate filter in a current population and selecting the top-ranked candidate filter. Next, there is a possibility of storing a new trainer in trainers circular list. After deciding whether to store the new trainer or not, a new generation is created and the evolution loop continues with the next iteration. The evolution loop terminates when the predefined count of generations is reached.

The TOVs subsets evolution loop begins with loading the trainers from shared memory. Depending on the selected method the exact fitness is or is not evaluated. In CFP, the exact fitness $MDPP_{exact}$ is calculated for each trainer. Then the TOVs subset fitness is evaluated using f_{CFP} (Eq. 4). In CC, the exact fitness values of trainers are not evaluated, and the TOVs subset fitness is calculated using f_{CC} (Eq. 5). As the top-ranked TOVs subset is then taken the subset with the maximal f_{CC} value, which means that this TOVs subset filtered using trainers has the worst (maximal) mean quality measured by $MDPP_{predicted}$ over trainers. This can help the candidate filters to improve filtering those training cases, which cannot be solved yet.

```
BEGIN Filters Thread
    Randomize trainers-circular-list
    Randomize filters population
    FOR 1 TO generation-total-count DO BEGIN
        Load top-ranked-TOVs-subset from shared memory
        Evaluate partial fitnesses for filters using top-ranked-TOVs-subset
        Select top-ranked-filter
        IF actual-parent-fitness <> previous-parent-fitness THEN BEGIN
            Store parent to trainers-circular-list to shared memory
        END
        Create new generation of filters using top-ranked-filter
    END
    Set terminating-flag
    Evaluate exact fitness of last-top-ranked-filter
    RETURN last-top-ranked-filter
END

BEGIN TOVs Thread
    Randomize TOVs population
    REPEAT forever
        Load trainers-circular-list from shared memory
        [OPTIONALLY] Evaluate exact fitnesses of trainers     // depending on selected method
        Evaluate fitnesses for TOVs-subsets using trainers
        Select top-ranked-TOVs-subset
        Store top-ranked-TOVs-subset to shared memory
        Create new generation of TOVs-subsets using 2-tournament selection
                                              and single point crossover
        IF terminating-flag THEN BEGIN
            EXIT thread
        END
    END
END
```

Fig. 3. Pseudocode for coevolution of the population of filters and the TOVs subsets population

After trainers processing, the fitness values of TOVs subsets are evaluated and the top-ranked TOVs subset is selected and stored to shared memory. The worst-ranked TOVs subset is replaced by a new randomly generated one, which is involved in TOVs subset reproduction to ensure TOVs subsets population diversity. A new generation is then created and the evolution loop continues with the next iteration, while the terminating flag is not set up.

4 Results

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

4.1 Benchmark Problems

In order to evaluate the proposed approach, salt and pepper noise filters will be designed using CGP. This type of noise is characterized by noisy pixels with the value of either 0 or 255 (for the 8-bit gray-scaled images) typically caused by

errors in data transmission, faulty memory locations in hardware or malfunctioning pixels in camera sensors. Conventionally designed filters for this type of noise are based on the median function. Two noise intensities have been applied, i.e. the Lena training image with resolution 256×256 pixels was corrupted by 5 % and 10 % salt-and-pepper type of noise. Evolved filters are tested using five different images containing the same type of noise.

4.2 Experimental Setup

CGP is used according to literature [5], i.e. $n_c = 8$, $n_r = 4$, $l = 1$, $n_i = 9$, $n_o = 1$, $\lambda = 7$, every node has two inputs, the number of mutations per new individual is $h = 5$ and Γ contains the functions from Table 1. The trainers set is represented as a circular list with eigth elements.

TOVs subsets are evolved using a simple GA, where 2-tournament selection, single point crossover (with the probability 100%, but the top-ranked individual is always involved in next generation) and mutation up to 2% of chromosome are used. For the GA, various chromosome lengths are tested, particularly, 2.5 %, 5 %, 10 %, 15 %, 20 % and 25 % of total size of TOVs in the training set (which contains 64,516 TOVs because boundary pixels are not considered in the 256×256 training images). For each TOVs subset size, 25 independent runs were performed and the evolution/coevolution was terminated after 30,000 generations of CGP.

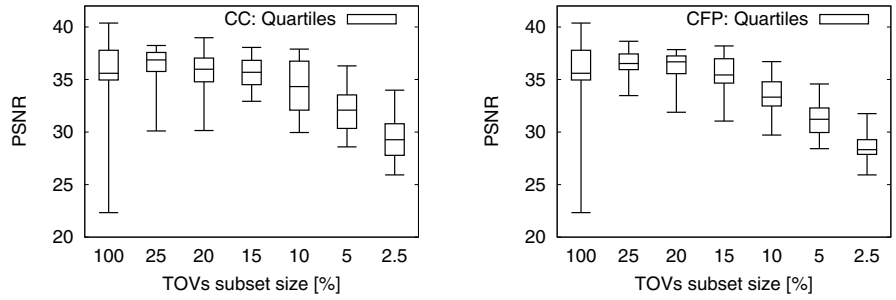
4.3 Comparison of Coevolving CGP with Standard CGP

The proposed coevolutionary algorithms were compared with the standard CGP in terms of filtering quality of evolved filters and the execution time.

The quality of filtering is expressed as a peak signal-to-noise ratio (PSNR) which is a measure typically used in the image processing community. It can be seen in Figure 4 that the proposed coevolutionary algorithm is capable of evolving image filters of satisfactory quality even if only 15% of TOVs are utilized.

Table 2 gives PSNR for five test images filtered by the best filters evolved using standard CGP, coevolutionary CGP (CC, the TOVs subset size is 20 %) and conventional median filter. The PNSR results of coevolutionary CGP are comparable or better with respect to the standard CGP for both noise intensities. The PSNR results for the median filter are not as good, however, we expected this on the basis of our previous work [9]. The best filter for the 5% noise evolved using CGP with coevolution is shown in Figure 1.

Figure 5 shows one of test images corrupted with the 5 % salt-and-pepper noise and then filtered using the median filter, the best filter evolved using the standard CGP and the best filter evolved using CGP with coevolution. The median filter provides smudged images in comparison to evolved filters.



(a) CC PSNR **(b)** CFP PSNR

Fig. 4. PSNR statistics calculated from 25 independent runs of CC and CFP for the Lena image (5% noise). The 100 % result is for the standard CGP.

It can be seen in Figure 6 that for the 15% TOVs subset size, the evolutionary design is accelerated 2.99-times in comparison to the standard CGP. Note that the execution time is given as the sum of execution times of both threads. The speedup was measured on the 2× 8-thread Intel® Xeon® E5640 machine.

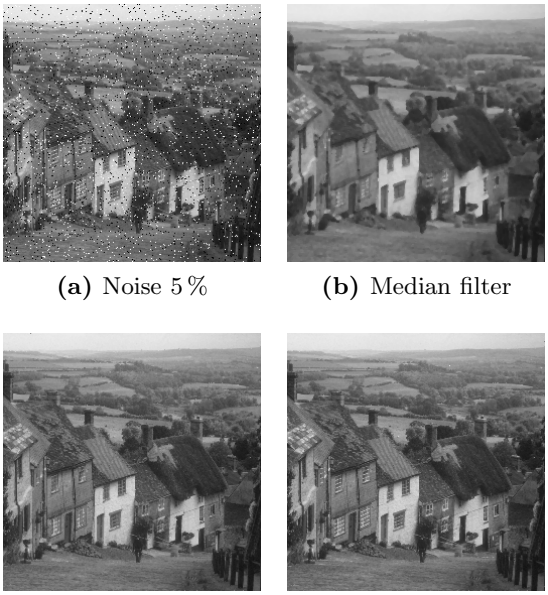
Figures 6 and 4 show that the CC and CFP fitness interacting strategies are similar in terms of filtering quality of evolved filters and execution time. However, profiles of evolved TOVs subsets differ. The CFP fitness interacting strategy leads to TOVs subsets that have a similar ratio of corrupted pixels as the total TOVs set, while the CC fitness interacting strategy leads to a little higher ratio of corrupted pixels in TOVs subsets (Table 3).

Table 2. PSNR for test images filtered by the best filters evolved using standard CGP, coevolutionary CGP (CC, the TOVs subset size is 20 %) and conventional median filter.

Test image	5 % noise			10 % noise		
	std CGP	coevolution	median 3×3	std CGP	coevolution	median 3×3
Airplane	38.008	37.747	29.303	32.370	34.053	28.557
Bird	46.113	44.706	38.242	35.735	40.054	36.990
Bridge	35.117	33.707	26.040	30.051	30.246	25.662
Camera	35.299	36.075	26.823	30.590	32.800	26.245
Goldhill	37.799	37.906	27.927	32.284	34.012	27.524
Lena	38.233	38.357	30.381	33.332	35.301	29.739

Table 3. Comparison of a mean ratio of corrupted pixels in the top-ranked TOVs subsets in the CFP and CC strategy (5 % noise).

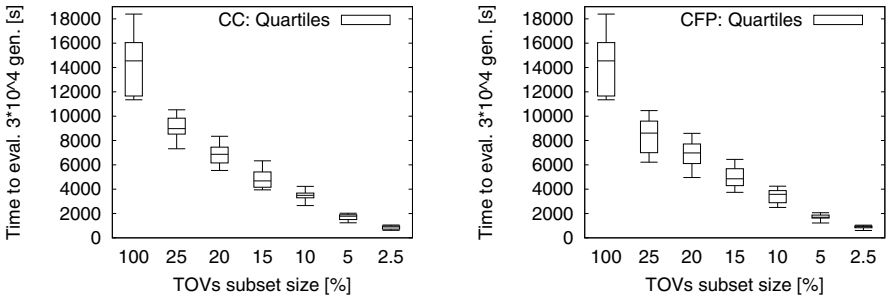
Subset size	25 %	20 %	15 %	10 %	5 %	2.5 %
CFP	4.973 %	5.009 %	4.971 %	4.960 %	4.961 %	5.087 %
CC	5.328 %	5.391 %	5.456 %	5.658 %	5.519 %	6.114 %



(a) Noise 5 % **(b)** Median filter

(c) Std CGP filter **(d)** Coe CGP filter

Fig. 5. Comparison of images filtered by median filter (b), the best filter evolved using the standard CGP (c), and CGP with coevolution (d)



(a) CC speed **(b)** CFP speed

Fig. 6. Execution time statistics calculated from 25 independent runs for CC and CFP. The 100% result is for the standard CGP running in one thread. Other values are for coevolution using two threads (the execution time is the sum of both threads).

5 Conclusions

In this paper, we proposed two coevolutionary methods to CGP in order to accelerate the evolutionary design of image filters. No significant differences were observed between the cooperative coevolution and coevolution of fitness predictors. It was shown that only 15–20% of original test vectors are needed to find

an image filter which provides the same quality of filtering as the best filter evolved using the standard CGP which utilizes the whole training set. The median time of evolution was reduced 2.99 times in comparison with the standard CGP. Future work will be devoted to further parallelization of the whole concept.

Acknowledgments. This work was supported by the Czech science foundation projects P103/10/1517 and GD102/09/H042, 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] Harding, S.L., Banzhaf, W.: Hardware acceleration for cgp: Graphics processing units. In: Cartesian Genetic Programming, pp. 231–253. Springer (2011)
[2] Hillis, D.W.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D: Nonlinear Phenomena 42(1–3), 228–234 (1990)
[3] Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers (2003)
[4] Lohn, J., Kraus, W., Haith, G.: Comparing a coevolutionary genetic algorithm for multiobjective optimization. In: Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002, vol. 2, pp. 1157–1162 (2002)
[5] Miller, J.F.: Cartesian Genetic Programming. Springer (2011)
[6] Pagie, L., Hogeweg, P.: Evolutionary consequences of coevolving targets. Evolutionary Computation 5(4), 401–418 (1997)
[7] Rosin, C.D., Bellew, R.K.: New methods for competitive coevolution. Tech. Rep. CS96-491, Department of Computer Science and Engineering, University of California, San Diego (1996)
[8] Schmidt, M.D., Lipson, H.: Coevolution of Fitness Predictors. IEEE Transactions on Evolutionary Computation 12(6), 736–749 (2008)
[9] Sekanina, L., Harding, L.S., Banzhaf, W., Kowaliw, T.: Image processing and cgp. In: Cartesian Genetic Programming, pp. 181–215. Springer (2011)
[10] Šikulová, M., Sekanina, L.: Coevolution in Cartesian Genetic Programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 182–193. Springer, Heidelberg (2012)
[11] Vasicek, Z., Sekanina, L.: Hardware accelerator of cartesian genetic programming with multiple fitness units. Computing and Informatics 29(6), 1359–1371 (2010)
[12] Wang, J., Chen, Q.S., Lee, C.H.: Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware. IET Computers and Digital Techniques 2(5), 386–400 (2008)

Paper C

Indirectly Encoded Fitness Predictors Coevolved with Cartesian Programs

Michaela Sikulova, Jiri Hulva, and Lukas Sekanina

In *Genetic Programming - 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings*, volume 9025 of *Lecture Notes in Computer Science*, pages 113–125. Springer, 2015.

Indirectly Encoded Fitness Predictors Coevolved with Cartesian Programs

Michaela Sikulova( ), Jiri Hulva, and Lukas Sekanina

Faculty of Information Technology, IT4Innovations Centre of Excellence,
Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic
{isikulova,sekanina}@fit.vutbr.cz, xhulva00@stud.fit.vutbr.cz

Abstract. We investigate coevolutionary Cartesian genetic programming that coevolves fitness predictors in order to diminish the number of target objective vector (TOV) evaluations, needed to obtain a satisfactory solution, to reduce the computational cost of evolution. This paper introduces the use of coevolution of fitness predictors in CGP with a new type of indirectly encoded predictors. Indirectly encoded predictors are operated using the CGP and provide a variable number of TOVs used for solution evaluation during the coevolution. It is shown in 5 symbolic regression problems that the proposed predictors are able to adapt the size of TOVs array in response to a particular training data set.

Keywords: Coevolution · Cartesian genetic programming · Fitness prediction

1 Introduction

The development of *Genetic Programming* (GP) is mainly driven by the increasing demand to solve complex problems which cannot be solved directly or systematically using informed methods. In many real-world applications, the fitness evaluation of a candidate program is computationally very expensive. Often, the fitness in GP is calculated over a set of *fitness cases* [11]. A fitness case corresponds to a representative situation in which the ability of a program to solve a problem can be evaluated. A fitness case consists of potential program inputs and target values expected from a perfect solution as a response to these program inputs. Potential program inputs and the corresponding target values are ordered in a sequence called *target objective vector* (TOV).

A set of TOVs (*training data*) is typically a small sample of the entire domain space. The choice of how many TOVs (and which ones) to use is often a crucial decision since whether or not an evolved solution will generalize over the entire domain depends on this choice. It also holds for the evolutionary design which has been performed by *Cartesian Genetic Programming* (CGP). In the case of symbolic regression or the evolutionary image filter design (which is one of the typical application domains for CGP [8]), from hundreds to tens of thousands TOVs have to be evaluated in order to obtain a single fitness value. In order to

P. Machado et al. (Eds.): EuroGP 2015, LNCS 9025, pp. 113–125, 2015.
DOI: 10.1007/978-3-319-16501-1_10

find a robust and acceptable solution a large number of fitness evaluations has to be performed.

Fitness modeling methods have been used to reduce the computational complexity of expensive fitness evaluations [2]. A predefined model or coarse-grained simulation has been used to approximate the fitness value in cases in which obtaining the exact fitness requires an expensive simulation or a physical experiment. Machine learning methods or a subsampling of training data can be used in order to approximate the fitness efficiently. However, it is not always clear when the benefits of fitness modeling can outweigh the cost.

A closely related concept to fitness modeling is *fitness prediction*, which is a technique used to replace fitness evaluations by a lightweight approximation that adapts with the solution evolution. Fitness predictors cannot approximate the entire fitness landscape, but they are instead shifting their focus throughout the evolution. An algorithm that coevolves fitness predictors, optimized for the solution population, has been introduced for standard (tree-based) genetic programming in order to reduce the fitness evaluation cost and frequency by Schmidt and Lipson [7].

In our previous work, inspired by *coevolution of fitness predictors* [7] and the *coevolutionary principles* which have been summarized in [4], we applied a coevolution of TOVs in order to accelerate fitness evaluations in CGP. We adopted the fitness predictor encoding in the form of a subset of training data. Fitness predictors have been represented as a constant-size array of pointers to elements in the training data and operated using a simple genetic algorithm. Coevolutionary algorithm has been adapted for CGP. We have obtained a significant speedup (2.03–5.45) over the standard CGP for 5 symbolic regression problems [10] and the results have been very competitive with tree-based GP. The same coevolutionary CGP and Hillis' *competitive coevolution* approach [1] adapted for CGP have been used in the evolutionary image filter design [9]. Although the median time of evolution has been reduced 2.99 times in comparison with standard CGP, a large number of experiments had to be accomplished in order to find the most advantageous size of the fitness predictor (the number of TOVs in predictor) for this particular task. An open problem is how to reduce this overhead.

This paper deals with a new type of fitness predictors whose size is changing dynamically during the coevolution. These fitness predictors with a variable number of TOVs are represented in the form of functional expressions. This functional expression generates a certain number of indexes into the training data. Indexes then address specific TOVs from the original training data which are selected for solution fitness prediction. The proposed method is evaluated using 5 symbolic regression problems and compared with the original approach.

The paper is organized as follows. Section 2 introduces Cartesian genetic programming, Sect. 3 summarizes our previous work on Coevolution of Fitness predictors in the CGP and outlines an open issue. In Sect. 4, a new approach to fitness predictor encoding is presented. Experimental results are discussed in Sect. 5. Finally, conclusions are given in Sect. 6.

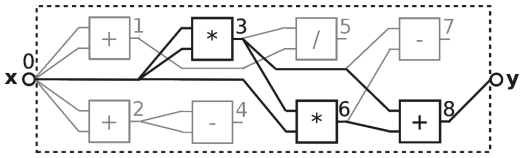


Fig. 1. A candidate program in CGP, where $l = 4, n_c = 4, n_r = 2, n_i = 1, n_o = 1, n_a = 2, \Gamma = \{+(1), -(2), *(3), /(4)\}$ and chromosome is: 0, 0, 1; 0, 0, 1; 0, 0, 3; 2, 2, 2; 3, 1, 4; 3, 0, 3; 3, 6, 2; 3, 6, 1; 8.

2 Cartesian Genetic Programming

The state of the art of Cartesian genetic programming has been summarized in a monograph [3]. CGP is a variant of genetic programming that uses a specific encoding in the form of directed acyclic graph and a mutation-based search. CGP has been successfully employed in many traditional application domains of genetic programming such as symbolic regression, but has been predominantly applied in evolutionary design and optimization of logic networks.

A candidate program in CGP is modelled as a Cartesian grid of $n_{\mathrm{c}} \times n_{\mathrm{r}}$ (columns $\times$ rows) programmable elements (nodes). The number of primary inputs, n_{i}, and outputs, n_{o}, of the program are defined for a particular task. Each node input can be connected to the output of a node placed in previous l columns or to one of the program primary inputs. The types of n_a-input node functions are decided by user and defined in the set Γ. Each node of the directed graph represents a particular function and is encoded by $n_{\mathrm{a}} + 1$ genes. One gene is the code of node function, the remaining genes are the indexes of the node input connections. Figure 1 shows an example of a candidate program and its encoding in the chromosome.

In CGP, a variant of a simple $(1 + \lambda)$ evolutionary algorithm is used as a search mechanism. The initial population is constructed either randomly, by a heuristic procedure or uses an existing solution. Every new population consists of the best individual of the previous generation (so-called parent) and its λ offspring. To create the offspring individuals from the parent, a point mutation operator is used. Mutation modifies h randomly selected genes to another randomly generated (but valid) values.

3 Fitness Prediction in CGP

In our previous work, fitness predictors were small subsets of the training data and coevolved with CGP programs [10]. An optimal fitness predictor was sought using a simple genetic algorithm (GA) which operated a population of fitness predictors. Fitness predictor was directly encoded as a constant-size array of pointers to the elements (TOVs) in the training data. It was shown in 5 symbolic regression benchmarks that only 12 TOVs for fitness prediction were needed to find a satisfactory solution. Moreover, a significant improvement (in terms

of computational cost reduction) has been obtained in comparison with CGP without coevolution.

The coevolution adapted for CGP has been used in the evolutionary design of image filters, where the standard CGP has been successful so far. Using coevolutionary CGP, a computational cost reduction has been obtained too [9]. However, this utilization brings some potential problems. The process of finding the most advantageous setting in terms of the fitness predictor size for this particular task was the most time consuming part of the experiments. Too many independent runs had to be performed to observe that 15–20% (about 10 thousand of TOVs) of original training data are needed to find an image filter of the same quality of filtering as the best filter evolved using the standard CGP utilizing the original training data. While using GA chromosomes as long as thousands genes, the so-called *scalability problem* has been observed. In the context of EAs the scalability problem refers to the situation in which the evolutionary algorithm is able to provide a very good solution to a small problem instance, but only unsatisfactory solutions can be generated for larger problem instances.

4 Proposed Method

The number of TOVs required to obtain a satisfactory solution is variable from benchmark to benchmark. To simply apply a coevolutionary CGP to a new, unknown task, we should consider a fitness predictor with the dynamic size which can be adapted during the coevolutionary process. Although the direct encoding of the predictor involves a simpler encoding which is suitable for basic applications, more complex tasks need sizable predictors that are sorely handled by GA. Several possible encodings of fitness predictor have been mentioned in [6]. Fitness predictors, in this work are not, however, encoded as the constant-size array of TOVs. Instead, we use an indirect encoding in the form of functional expression selecting particular TOVs. TOVs used for fitness prediction are selected by means of indexes that are generated using this expression.

The evolution of this expression can be seen as a form of a symbolic regression task which is a typical task for genetic programming. We have considered to employ CGP due to a simpler and faster operation on chromosomes.

4.1 Indirectly Encoded Predictor

In this paper, the evolution of fitness predictors is based on the principles of CGP as introduced in Sect. 2. The predictor chromosome encodes a Cartesian grid of two-input functional nodes operating over one primary input and returning two primary outputs. In addition to the Cartesian grid, an initializing value x_0 is encoded in the chromosome. It is operated using a special mutation operator – value x_0 is multiplied by a randomly generated real number in the user-defined range.

While composing the array of TOVs for fitness prediction, the initializing value x_0 is used as a primary input of the predictor. In response to the primary

input x_i, the candidate predictor returns two outputs - $out_0(x_i)$ and $out_1(x_i)$. Index $j(x_i)$ of selected TOV is then calculated as:

$$j(x_i) = out_0(x_i) \mod n, \tag{1}$$

where n is the total number of TOVs in the training data. TOVs selection continues with the next iteration using $out_0(x_i)$ as a new primary input of the predictor, until the $out_1(x_i)$ is out of the user-defined range r_{out1} or the maximum size of the array of TOVs for fitness prediction is reached.

4.2 Predictor Training

Predictors have to be coevolved with the solution evolution in order to adapt them to the solved problem. Predictor training data consists of *fitness trainers*, which are selected copies of candidate solutions occurred during the solution evolution and their corresponding exactly measured fitness values $\mathrm{f}_{\mathrm{exact}}$ (i.e. fitness evaluated using the original training data).

The archive of trainers has a well defined structure. The number of trainers in the archive is kept constant during the evolution. While initializing the coevolution, solutions from the first generation are chosen and copied to the archive of trainers. If the archive of trainers is larger than the solution population, missing trainers are generated randomly. Trainers in the archive are updated periodically – the top-ranked candidate solution is copied to the trainers archive if its predicted fitness value differs from the top-ranked candidate solution in the previous generation; the next trainer is updated using a random solution. A new trainer t replaces the oldest one in the trainers archive and the exact fitness of the new trainer $\mathrm{f}_{\mathrm{exact}}(t)$ is evaluated. This approach to the predictor training data structure leads to maintaining a representative sample of the current solution population (due to the copies of top-ranked candidate solutions) as well as maximizing fitness diversity of solutions in the archive of trainers (due to the randomly generated trainers).

4.3 Fitness of Predictor

While designing the fitness function of the indirectly encoded predictor, two main options should be considered: (1) prediction precision and (2) prediction cost. Prediction precision of predictor p is calculated using the *relative error* of the exact and predicted fitness values of solutions in the trainers archive:

$$\mathrm{prec}\,(p) = \frac{1}{u}\sum_{j=1}^{u} u\frac{|\mathrm{f}_{\mathrm{exact}}(t_j) - \mathrm{f}_{\mathrm{predicted}}(t_j)|}{\mathrm{f}_{\mathrm{exact}}(t_j) + c}, \tag{2}$$

where u is the number of solutions in the trainers archive, parameter c allows to moderate a sharp increase of relative error while the $\mathrm{f}_{\mathrm{exact}}(t_j)$ is very close to 0.

Prediction cost of a predictor is depended on how many TOVs have to be used while evaluating the predicted fitness. The number of TOVs in the array for

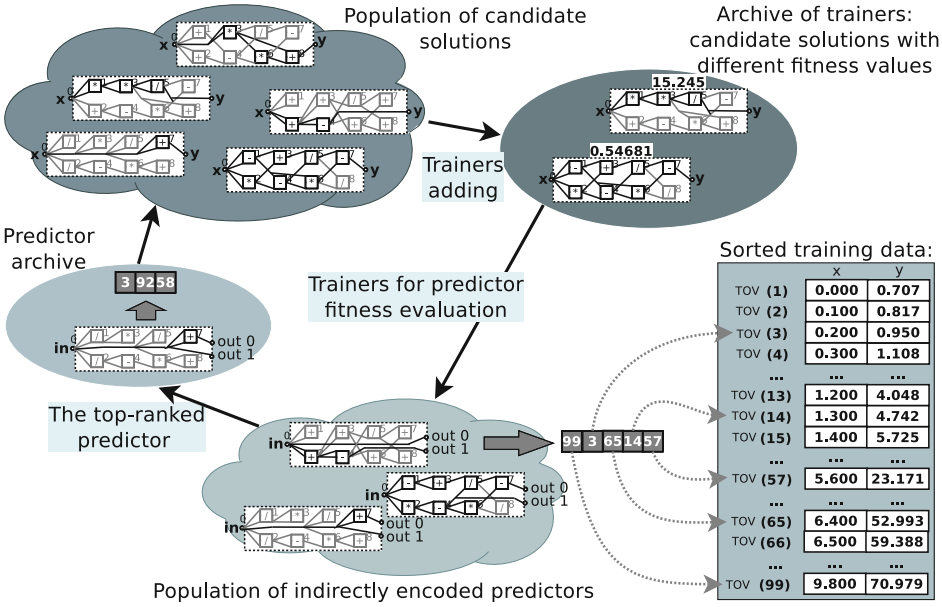


Fig. 2. Coevolution of candidate solutions and fitness predictors.

fitness prediction – size (p) – is employed for this purpose. To simplify the fitness evaluation process, both objectives are embedded in the single fitness function. To establish the fitness function for predictor p, a desired function flow has been processed by the Eureqa software [5], and the following function has been obtained:

$$\mathrm{f}(p) = (a \cdot \mathrm{prec}(p))^4 + b \cdot \mathrm{size}(p) \left(1 + a \cdot \mathrm{prec}(p)^2\right), \tag{3}$$

where size (p) is the number of TOVs addressed by predictor p. Parameters a and b control the preference between the prediction precision and the prediction speed (i.e. the number of TOVs addressed by predictor).

4.4 Coevolution of Solutions and Predictors

In the first step, the candidate solutions and candidate predictors are randomly initialized. Then the solution evolution waits for the first top-ranked predictor (obtained in the first generation of predictors). The evolution of candidate solutions is based on the principles of CGP. TOVs addressed by the predictor loaded from the *predictor archive* are used for fitness prediction of the solutions. The top-ranked predictor for solution fitness prediction is then updated periodically in a user-defined number of solution generations. While evolving solution, top-ranked solutions with different fitness are copied to the one half of the trainers archive. In each generation of predictors, one trainer from the second half of trainers archive is updated randomly.

Predictors are also evolved using CGP. Using each predictor in the current generation, the predicted fitness of trainers is evaluated and the fitness of the predictor is established. The top-ranked predictor is then updated in a predictor archive and also used for producing a new generation of predictors. The overall scheme of the proposed coevolutionary algorithm is shown in Fig. 2.

If a satisfactory solution is found or the user-defined maximum number of solution generations is reached, the coevolution is terminated.

5 Results

This section presents benchmark problems, experimental setup and experimental evaluation of the proposed approach and its comparison with the original directly encoded fitness predictors and standard CGP without coevolution.

5.1 Benchmark Problems

Five symbolic regression benchmark functions (F1–F5) were selected as TOV sources for evaluation of the proposed method:

$$\begin{aligned}
&\text{F1}: f(x) = x^2 - x^3, && x = [-10 : 0.1 : 10]\\
&\text{F2}: f(x) = e^{|x|}\sin(x), && x = [-10 : 0.1 : 10]\\
&\text{F3}: f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right), && x = [-10 : 0.1 : 10]\\
&\text{F4}: f(x) = e^{-x}x^3 \sin(x)\cos(x)\left(\sin^2(x)\cos(x) - 1\right), && x = [0 : 0.05 : 10]\\
&\text{F5}: f(x) = \frac{10}{(x-3)^2 + 5}, && x = [-2 : 0.05 : 8].
\end{aligned}$$

In order to form a training data, 200 equidistant distributed samples were taken from each function. Functions F1, F2 and F3 are taken from [7], functions F4 and F5 from [12] and all functions F1–F5 were used in order to evaluate coevolution of CGP and directly encoded predictors [9].

5.2 Experimental Setup

The setup of the solution evolution is used according to literature [10], i.e. $\lambda = 12, n_i = 1, n_o = 1, n_c = 32, n_r = 1, l = 32$, every node has two inputs $(i_1, i_2), \Gamma = \{i_1 + i_2, i_1 - i_2, i_1 \cdot i_2, \frac{i_1}{i_2}, \sin(i_1), \cos(i_1), e^{i_1}, \log(i_1)\}$ and the maximum number of mutation per individual is $h = 8$. The solution fitness function is defined as the relative number of hits. There are, in fact, two fitness functions for candidate solution s. While the exact fitness function $f_{exact}(s)$ utilizes the complete training set, the fitness function for fitness prediction $f_{predicted}(s)$ employs only selected TOVs. Formally,

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

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

$$\mathrm{g}(y(j)) = \begin{cases} 0 \text{ if } |y(j) - t(j)| \geq \varepsilon \\ 1 \text{ if } |y(j) - t(j)| < \varepsilon \end{cases} \tag{6}$$

where $y(j)$ is a candidate program response to TOV $j, t(j)$ is the target response, n is the number of TOVs in the training data, m is the number of TOVs addressed by predictor and ε is a user-defined acceptable error – for benchmarks F1, F2: 0.5; F3: 1.5; F4, F5: 0.025. The acceptable number of hits is 97 %.

To find the most advantageous setting of the predictor evolution, over 160,000 independent runs were performed. The results were obtained using the following setup of the predictor evolution: $\lambda = 4, n_i = 1, n_o = 2, n_c = 15, n_r = 2, l = 4$, every node has two inputs (i_1, i_2), $\Gamma = \{i_1 + i_2, i_1 - i_2, i_1 \cdot i_2, \frac{i_1}{i_2}, \sin(i_1), \max(i_1, i_2), \min(i_1, i_2), -i_1, i_1 \mod i_2, |i_1|$ and number of mutation per individual is $h = 30$. The range of out_1 (affecting the number of TOVs addressed by predictor) is set as $-1000 < r_{out1} < 1000$; the minimum number of TOVs addressed by predictor is 5 (2,5 % of the complete training data) and the maximum number is 50 (25 %). Parameters of predictor fitness function were empirically set as follows: Predictor precision (Formula 2) parameter $c = 0.002$ and the predictor fitness function (Formula 3) parameters $a = 17$ and $b = 0.04$. Every 2,000 generations of the solution evolution, a new predictor has been loaded for the solution fitness evaluation.

5.3 Comparisons of the Algorithms

The goal of this experiment is to compare the proposed coevolution of indirectly encoded fitness predictors evolved using CGP (FP_{CGP}) with the original directly encoded fitness predictors evolved using GA (FP_{GA}) and standard CGP without coevolution (CGP_{STD}). In all the algorithms, solutions are evolved using the equivalent setup as presented in Sect. 5.2. FP_{GA} is used according to literature [9], i.e. 12 TOVs in chromosome, 32 individuals in predictor population, 2-tournament selection, a single point crossover and the mutation probability 0.2. The algorithms are compared in terms of the success rate (the number of runs, giving a solution with predefined quality), the number of generations and the number of TOV evaluations to converge (in order to compare the computational cost). Table 1 gives the median values calculated of 50 independent runs for each benchmark function F1–F5.

It can be seen from Table 1 that both coevolutionary approaches have reached a satisfactory solution using a significantly fewer TOV evaluations than the standard CGP. Despite the fact that during FP_{CGP} evolution predictors with a large number of TOVs have to be evaluated, the number of TOV evaluations

Table 1. Comparison of standard CGP (CGP_{STD}) and coevolutionary CGP with directly encoded predictors FP_{GA} and indirectly encoded predictors FP_{CGP}.

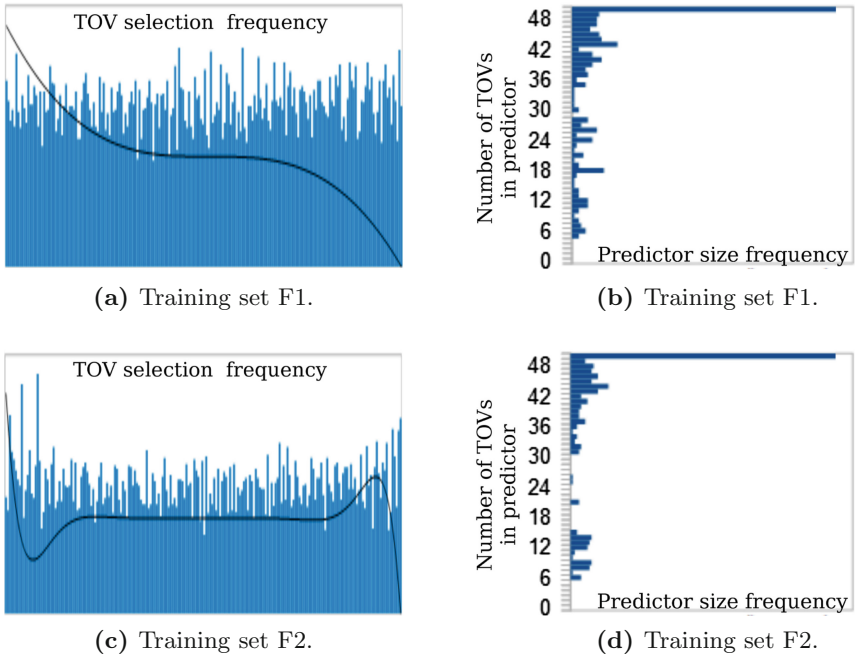
	Algorithm	F1	F2	F3	F4	F5
Success rate	CGP_{STD}	100 %	100 %	100 %	80 %	24 %
	FP_{GA}	100 %	100 %	100 %	100 %	100 %
	FP_{CGP}	100 %	100 %	100 %	100 %	90 %
Generations to converge (median)	CGP_{STD}	$1.11 \cdot 10^3$	$4.46 \cdot 10^3$	$1.76 \cdot 10^5$	$7.15 \cdot 10^5$	$1.36 \cdot 10^6$
	FP_{GA}	$2.62 \cdot 10^3$	$2.53 \cdot 10^3$	$1.10 \cdot 10^5$	$1.00 \cdot 10^6$	$1.34 \cdot 10^6$
	FP_{CGP}	$1.00 \cdot 10^3$	$2.25 \cdot 10^3$	$4.11 \cdot 10^4$	$1.47 \cdot 10^6$	$1.74 \cdot 10^6$
TOV evaluations to converge (median)	CGP_{STD}	$2.68 \cdot 10^6$	$1.08 \cdot 10^7$	$4.24 \cdot 10^8$	$1.72 \cdot 10^9$	$3.28 \cdot 10^9$
	FP_{GA}	$5.20 \cdot 10^5$	$5.01 \cdot 10^5$	$2.19 \cdot 10^7$	$2.00 \cdot 10^8$	$2.67 \cdot 10^8$
	FP_{CGP}	$7.43 \cdot 10^5$	$1.60 \cdot 10^6$	$1.90 \cdot 10^7$	$8.05 \cdot 10^8$	$8.78 \cdot 10^8$

to converge is similar for both FP_{CGP}-evolved and FP_{GA}-evolved predictors. Although CGP_{STD} evaluates the whole TOVs set in every fitness function call, the number of generations is comparable for all three methods.

5.4 Predictor Behaviour

In this section we discuss how the predictors are able to select a representative sample of TOVs which allows for obtaining a satisfactory solution. However, it should be pointed out that to facilitate an indirectly encoded predictor to maintain eventual geometries or peaks and valleys in training data, the training set should be well sorted (if it is possible).

In order to observe the behaviour of predictor data samples, we plot (see Figs. 3 and 4) the number of TOVs and the frequency of TOVs addressed by predictors, which were used during the course of evolution for solution fitness prediction (50 independent runs considered). It can be seen from Table 1 that the satisfactory solution for benchmarks F1 and F2 can be obtained by $2 \cdot 10^3$ generations of the solution fitness prediction, which is the time when only the first co-evolved predictor is ready for solution fitness prediction. Then Fig. 3 shows the number and the frequency of TOVs addressed by the top-ranked predictor taken from the very first (randomly generated) generation. While the evolution wasn't allowed to suit to the training data, sizable predictors were selected in order to accomplish a better prediction precision – see Fig. 3b and d. Despite this fact, satisfactory solutions for benchmarks F1 and F2 have been found using a comparable number of TOV evaluations (and less number of generations of solutions) in comparison with directly encoded predictors using only 12 TOVs for solution fitness prediction (FP_{GA}).



(a) Training set F1. **(b)** Training set F1.

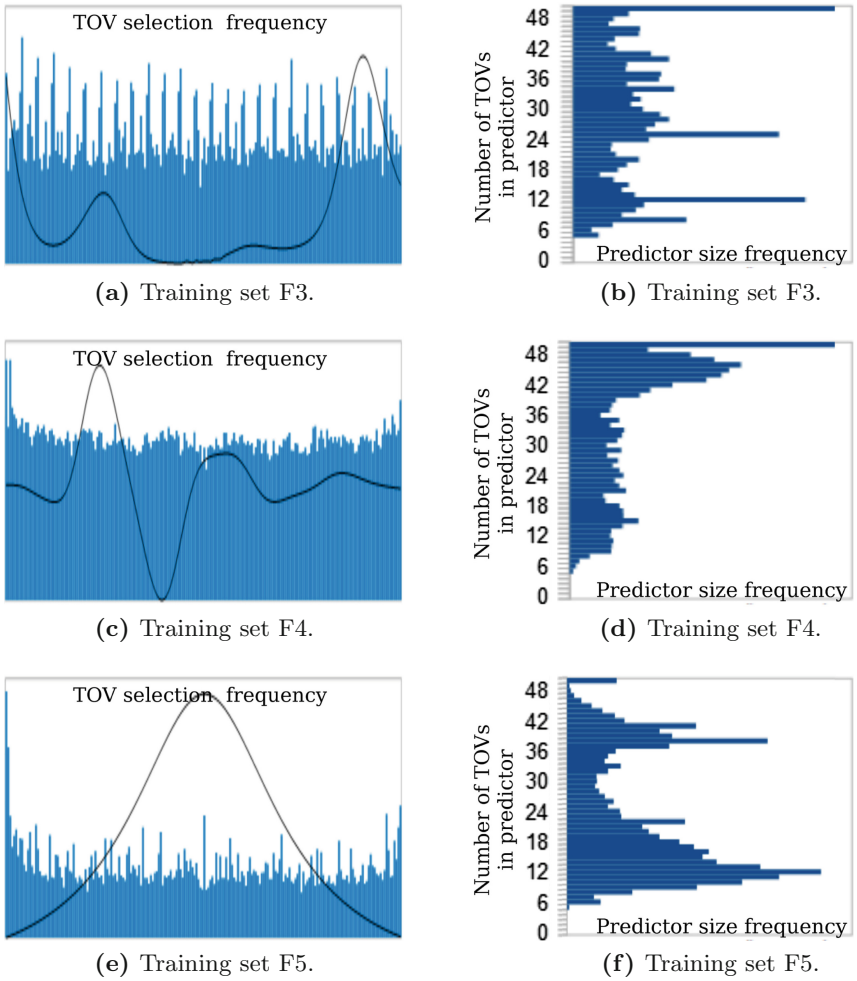
(c) Training set F2. **(d)** Training set F2.

Fig. 3. Frequency and the number of TOVs in predictors used for candidate solution fitness prediction (training set F1 and F2).

In case of benchmarks F3, F4 and F5, coevolution exhausted many more generations to converge. Then predictors were able to adapt to the training data. Figure 4b, d and f shows that each benchmark prefers a different number of TOVs for fitness prediction. It can be seen from Fig. 4f that about 10 TOVs and about 38 TOVs were preferred in order to predict the fitness of candidate solutions while solving the benchmark F5.

It can be seen in Fig. 4a, c and e that sample points do not focus entirely on the peaks and valleys of the training data, but are well distributed over the data set during the coevolution, however some geometries have been observed. If all TOVs addressed by predictor focus on the interesting regions (peaks and valleys) of the training data, the predictor would represent the maximum error (which is improper while requiring the predicted fitness corresponding to the exact fitness). Furthermore, TOVs addressed by the fitness predictors are variable in response to the solution evolution. The solution evolution forces the predictors to contain two types of TOVs, some of them are easy others difficult for particular solutions.

The number of TOVs addressed by predictors also changes during the coevolution in response to the course of solution evolution. Figure 5 shows the exact fitness of the top-ranked candidate solutions during the course of coevolution and the size of the predictor used to predict their fitness during a typical run for the benchmark F3.



(a) Training set F3. **(b)** Training set F3.

(c) Training set F4. **(d)** Training set F4.

(e) Training set F5. **(f)** Training set F5.

Fig. 4. Frequency and the number of TOVs in predictors used for candidate solution fitness prediction (training set F3–F5).

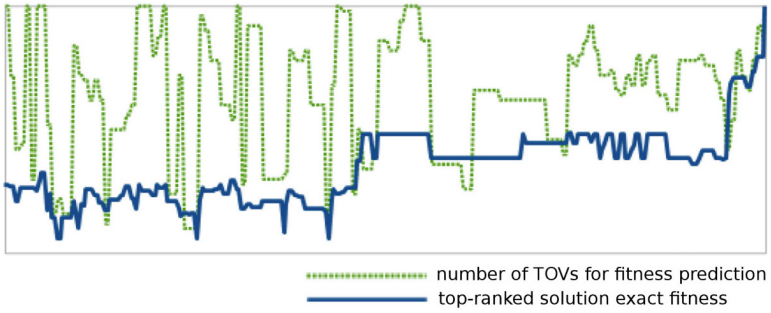


Fig. 5. Exact fitness of top-ranked candidate solutions during the course of evolution and the size of predictor during a typical run for the F3 data set.

6 Conclusions

In summary, we have introduced the use of coevolution of fitness predictors in CGP with a new type of indirectly encoded predictors. Indirectly encoded predictors are operated using the CGP and provide a variable number of TOVs used for solution fitness prediction during the coevolution in response to the solved problem. When applied to the symbolic regression problem, this approach was found to be comparable with the original directly encoded predictors using just 12 TOVs for the solution fitness prediction in terms of the number of evaluated TOVs to converge. We have shown using 5 benchmarks that proposed predictors are able to adapt the size of TOVs array for solution fitness prediction in response to the particular training data. This property enables to use the coevolution of fitness predictors for solving a new, unknown task, without the need to find the most advantageous size of the TOVs array experimentally.

However, as symbolic regression has not been considered as a typical application domain for CGP, our future work will be devoted to the utilization of the proposed fitness prediction algorithm in the evolutionary image filter design where the original directly encoded predictors have been successful so far. Considering the fact that the evolutionary design using CGP has been successfully accelerated in *field programmable gate array* (FPGA), another goal will be to implement coevolutionary CGP with indirectly encoded predictors to FPGA and thus accelerate the search process and use it in a real-world application.

Acknowledgments. This work was supported by the Czech science foundation project 14-04197S, the Brno University of Technology project FIT-S-14-2297 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

References

1. Hillis, W.D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D **42**(1), 228–234 (1990)
2. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput. J. **9**(1), 3–12 (2005)
3. Miller, J.F.: Cartesian Genetic Programming. Springer, Heidelberg (2011)
4. Popovici, E., Bucci, A., Wiegand, R., De Jong, E.: Coevolutionary principles. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 987–1033. Springer, Heidelberg (2012)
5. Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. Science **324**(5923), 81–85 (2009)
6. Schmidt, M.D., Lipson, H.: Co-evolving fitness predictors for accelerating and reducing evaluations. In: Riolo, R., Soule, T., Worzel, B. (eds.) Genetic Programming Theory and Practice IV. Genetic and Evolutionary Computation, vol. 5, pp. 113–130. Springer, Ann Arbor (2006)
7. Schmidt, M.D., Lipson, H.: Coevolution of fitness predictors. IEEE Trans. Evol. Comput. **12**(6), 736–749 (2008)

8. Sekanina, L., Harding, S.L., Banzhaf, W., Kowaliw, T.: Image processing and CGP. In: Miller, J.F. (ed.) Cartesian Genetic Programming, pp. 181–215. Springer, Heidelberg (2011)
9. Sikulova, M., Sekanina, L.: Acceleration of evolutionary image filter design using coevolution in Cartesian GP. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 163–172. Springer, Heidelberg (2012)
10. Šikulová, M., Sekanina, L.: Coevolution in Cartesian genetic programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 182–193. Springer, Heidelberg (2012)
11. Vanneschi, L., Poli, R.: Genetic programming – introduction, applications, theory and open issues. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 709–739. Springer, Heidelberg (2012)
12. Vladislavleva, K.: Toy benchmarks. In: Symbolic Regression: Function Discovery and More (2011). http://www.symbolicregression.com/?q=toyProblems

Paper D

Plastic Fitness Predictors Coevolved with Cartesian Programs

Michal Wiglasz and Michaela Drahosova

In *Genetic Programming - 19th European Conference, EuroGP 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings*, volume 9594 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2016.

Plastic Fitness Predictors Coevolved with Cartesian Programs

Michal Wiglasz( ) and Michaela Drahosova

Faculty of Information Technology, Brno University of Technology,
Božetěchova 2, 612 66 Brno, Czech Republic
{iwiglasz,idrahosova}@fit.vutbr.cz

Abstract. Coevolution of fitness predictors, which are a small sample of all training data for a particular task, was successfully used to reduce the computational cost of the design performed by cartesian genetic programming. However, it is necessary to specify the most advantageous number of fitness cases in predictors, which differs from task to task. This paper introduces a new type of directly encoded fitness predictors inspired by the principles of phenotypic plasticity. The size of the coevolved fitness predictor is adapted in response to the learning phase that the program evolution goes through. It is shown in 5 symbolic regression tasks that the proposed algorithm is able to adapt the number of fitness cases in predictors in response to the solved task and the program evolution flow.

Keywords: Fitness predictors · Cartesian genetic programming · Coevolution · Phenotypic plasticity

1 Introduction

Cartesian genetic programming (CGP) is a specific form of genetic programming (GP) and has been successfully applied to a number of challenging real-world problem domains [7]. In CGP, as well as in GP, every evolved program must be executed to find out what it does. Each program in the population is assigned a fitness value, representing the degree to which it solves the problem of interest. Often, but not always, the fitness is calculated over a set of *fitness cases*. A fitness case consists of potential program inputs and target values expected from a perfect solution as a response to these program inputs. The outputs of the evolved program are then compared with the desired outputs for given inputs. The choice of how many fitness cases (and which ones) to use is often a crucial decision since whether or not the evolved program will generalize over the entire domain depends on this choice.

In the case of digital circuit evolution, which is a typical task for CGP, 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 fitness cases does not lead to correctly working circuits [5].

In the *symbolic regression* tasks, the goal of GP system design and GP parameters' tuning is to obtain a solution with predefined accuracy and robustness.

M. Heywood et al. (Eds.): EuroGP 2016, LNCS 9594, pp. 164–179, 2016.
DOI: 10.1007/978-3-319-30668-1_11

In this case, 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, in order to find a robust and acceptable solution a large number of fitness evaluations has to be performed. In order to reduce the evaluation time, *fitness approximation* techniques have been employed, e.g. fitness modeling [6].

Closely related concept to the fitness modeling is a *fitness prediction*, which is a low cost adaptive procedure utilized to replace the 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 [9]. 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 [4]. The state of the art of coevolutionary principles has recently been summarized in the chapter of Handbook of Natural Computing [8].

Inspired by [9], we have introduced coevolving fitness predictors to CGP and have shown that by using them, the execution time of symbolic regression can significantly be reduced [12]. Fitness predictors have been represented as a constant-size array of pointers to elements in the fitness case set and operated using a simple genetic algorithm. The same coevolutionary CGP and Hillis' *competitive coevolution* approach [4] adapted for CGP have been used in the evolutionary image filter design [11]. Although the time of evolution has also been reduced, a large number of experiments had to be accomplished in order to find the most advantageous size of the fitness predictor (the number of fitness cases in predictor) for this particular task.

To solve this problem, we have introduced a new type of indirectly encoded fitness predictors which can automatically adapt the number of fitness cases used to evaluate the candidate programs [10]. However, during the evolution of fitness predictors, also large fitness predictors have to be evaluated (and then refused for a larger size), and thus plenty of fitness case evaluations have been wasted.

In this paper, we integrate *phenotypic plasticity* principles into coevolution. The phenotypic plasticity is the ability of an individual to learn how to utilize its genotype in order to adapt to the environment [1]. It was shown that a proper rate of environmental change may reduce the learning cost while evolving the solution [2,3]. Inspired by these principles, we introduce a new type of fitness predictors, operated using a simple genetic algorithm (GA), using the phenotypic plasticity in order to adapt the number of fitness cases for candidate solution evaluations and thus regulate the rate of environmental change. In the case of fitness prediction, a stable environment contains a complete fitness cases set, a highly changing environment only a few of them.

The paper is organized as follows. Section 2 introduces cartesian genetic programming and coevolution of fitness predictors. In Sect. 3, a new approach to fitness predictor encoding is presented. The proposed approach is evaluated using 5 symbolic regression benchmarks. Experimental results are discussed in Sect. 4. Finally, conclusions are given in Sect. 5.

2 Fitness Prediction in CGP

In standard CGP, candidate programs are represented in the form of directed acyclic graph, which is modeled as a matrix of $n_c \times n_r$ programmable elements (nodes). Each node is programmed to perform one of n_a-input functions defined in the set Γ. The number of primary inputs, n_i, and outputs, n_o, of the program is defined for a particular task. Each node input can be connected either to the output of a node placed in previous l columns or to one of the program inputs. Feedback is not allowed. The search is usually performed using a simple $(1 + \lambda)$ evolutionary algorithm, where usually $\lambda = 4$. Every new population consists of the best individual of the previous population and its λ offspring created using a mutation operator which modifies up to h genes of the chromosome. The state of the art of CGP has recently been summarized in a monograph [7].

In the case of *symbolic regression*, the set of fitness cases is usually constructed from experimentally obtained data. Then each of k fitness cases from the set is used to evaluate each candidate program (see Fig. 1). The fitness function of candidate program is often defined as the relative number of hits. Formally,

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

$$\mathrm{g}(y(j)) = \begin{cases} 0 \text{ if } |y(j) - t(j)| \geq \varepsilon \\ 1 \text{ if } |y(j) - t(j)| < \varepsilon \end{cases} \tag{2}$$

and y is a candidate program response, t is a target response and ε is a user-defined acceptable error. The fitness evaluation is the most time consuming part in standard CGP (as well as tree-based GP).

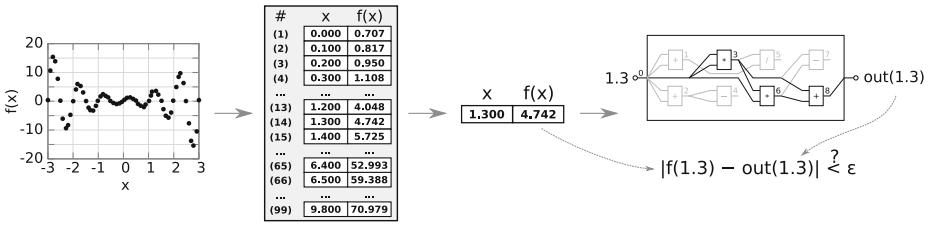


Fig. 1. Fitness evaluation of a candidate cartesian program.

2.1 Fitness Predictor

In order to reduce the total number of evaluations during each one fitness function call, fitness predictor in the form of small subset of the fitness case set have been introduced to CGP [12]. An optimal fitness predictor is sought using a simple genetic algorithm (GA) which operates with a population of fitness

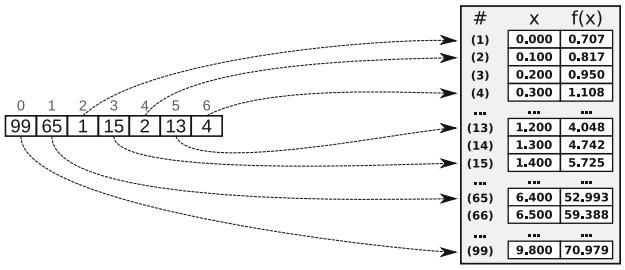


Fig. 2. Fitness predictor representation.

predictors. Every predictor is encoded as a constant-size array of pointers to elements in the training data (see Fig. 2). 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 goal of the evolution of predictors is to minimize the relative error of fitness prediction and the expensive exact fitness evaluation.

2.2 Coevolution of Cartesian Programs and Fitness Predictors

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. There are two concurrently working populations: (1) candidate programs (syntactic expressions) evolving using CGP and (2) fitness predictors evolving using GA. The overall scheme of the coevolutionary algorithm is shown in Fig. 3.

Evolution of candidate programs is based on principles of CGP. 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 set of fitness cases, the predicted fitness function $f_{predicted}(s)$ employs only selected fitness cases. Formally,

$$\mathrm{f}_{\mathrm{exact}}(s) = \frac{1}{k}\sum_{j=1}^{k} g\left(y\left(j\right)\right) \tag{3}$$

$$\mathrm{f}_{\mathrm{predicted}}(s) = \frac{1}{m}\sum_{j=1}^{m} g\left(y\left(j\right)\right) \tag{4}$$

where k is the number of fitness cases in the set of fitness cases and m is the number of fitness cases in the fitness predictor. The $f_{predicted}$ is used to evaluate the candidate programs in the population. The f_{exact} is used during the predictor training.

The predictor training is accomplished as follows. The *archive of trainers* is generated and updated in response to the candidate program evolution. It consists of candidate programs with evaluated f_{exact} and is divided into two

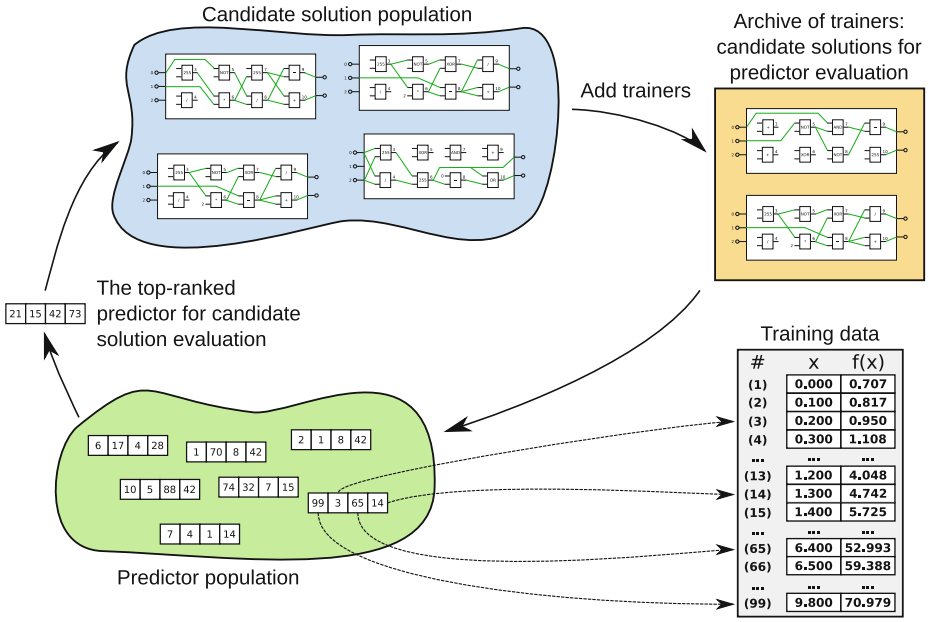


Fig. 3. Coevolution of candidate solutions and fitness predictors.

parts: The first part contains copies of top-ranked programs (with different fitness) obtained during the program evolution and the second part is periodically updated with randomly generated programs to ensure genetic diversity of the archive. The size of the archive is kept constant during the coevolution and each new trainer replaces the oldest one in the corresponding part of the archive.

The fitness value of predictor p is calculated using the *mean absolute error* of the exact and predicted fitness values of programs in the archive of trainers:

$$\mathrm{f}(p) = \frac{1}{u} \sum_{i=1}^{u} |\mathrm{f}_{exact}(i) - \mathrm{f}_{predicted}(i)| \tag{5}$$

where u is the number of candidate programs in the archive of trainers. The predictor with the best fitness value is used to predict the fitness of candidate programs in the population of candidate programs [9].

3 Proposed Method

In this paper, we propose a new approach to fitness predictor encoding. The number of fitness cases required to obtain a satisfactory solution varies from benchmark to benchmark. In order to apply coevolutionary CGP to different tasks, it is required to perform numerous experiments to find the most advantageous number of fitness cases in fitness predictors.

It can be observed that the population of solutions goes through various phases as the population's ability to adapt to the problem changes over the time [2]. A *lower fitness* phase needs less stimuli to improve solutions, but the same amount of stimuli does not lead to converge during the *higher fitness* phase. This property is discussed by Ellefsen [2], in order to reduce the learning cost. In this paper, the number of fitness cases in predictors is changed according to the latest development in the population of the candidate programs.

3.1 Plastic Directly Encoded Predictor

We propose directly encoded fitness predictors with an adaptive number of fitness cases for candidate solution evaluations. To be able to modify their size, we employ the principles of phenotypic plasticity. This allows the individual to produce different phenotypes from the same genotype, depending on the environmental conditions [2]. In the plastic fitness predictors, the phenotype is constructed by including only selected subset of genes.

The predictor genotype is a constant-size circular array of pointers to elements in the training data. Its size is equal to the total number of fitness cases. In order to produce the phenotype, the genes are read sequentially from specified position (*offset*). The genotype may contain duplicate gene values. Therefore, the gene with a value, which is already included in the phenotype, is skipped in order to prevent duplicate fitness case in predictor. The reading stops after it has processed the number of genes specified by the *readLength* variable. The *readLength* value is determined by the flow of the candidate program evolution.

The *offset* is determined by an extra gene included in the genotype, evolved by a special mutation operator, which adds a small Gaussian random number to the current value. Figure 4 shows an example of phenotype construction when 6 out of 10 available genes are used.

The evolution of predictors is directed by the genetic algorithm (GA). The crossover operator is modified so the split point is always selected within the active part of the genotype, which increases phenotype diversity.

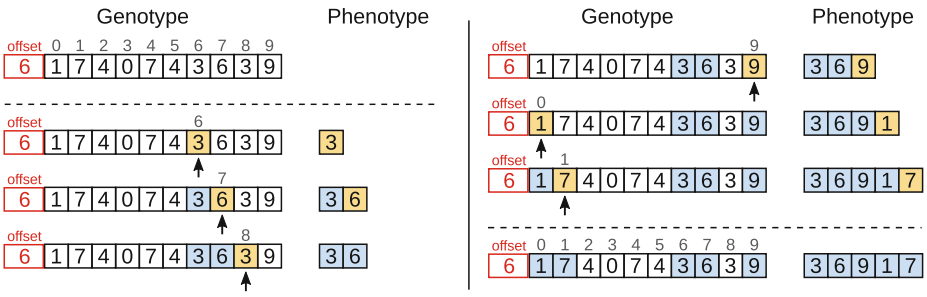


Fig. 4. Predictor phenotype construction with *offset* = 6 and *readLength* = 6.

3.2 Predictor Size Adaptation

The predictor size is adapted through the *readLength* variable. Its value is changed according to the latest development in the population of the candidate programs. It can be observed that the population goes through various phases as the population's ability to adapt to the problem changes over the time. If the ability is higher, the overall fitness increases towards better solutions, if it is lower, the fitness remains almost constant. In this case the evolution probably reached some local optimum.

The phase of evolution can be described in terms of the evolution speed which we express as follows:

$$v = \frac{\Delta f_{exact}}{\Delta G}, \tag{6}$$

where ΔG is the number of generations between two last fitness changes of CGP population parent (top-ranked programs) and Δf_{exact} is the difference of exact fitness values of these parents. Although the evolution of programs is guided by the predicted fitness, the speed can be negative, because it is calculated from the exact fitness.

It is necessary to set the lower boundary of the predictor size. If the prediction is based on only a few fitness cases (in extreme cases on only one fitness case), over-fitting of predictors occurs. The prediction inaccuracy can be expressed as the absolute difference between predicted and exact fitness:

$$I = |f_{predicted} - f_{exact}|, \tag{7}$$

In the case the prediction inaccuracy exceeds given threshold I_{thr}, the number of fitness cases should be increased.

The *readLength* value is updated each time a new solution with better predicted fitness than parent individual is found. It can be also updated after a user-specified number of generations during which a new solution is not found. The evolution speed and prediction inaccuracy is updated and a corresponding rule is selected. The rules are based on the following assumptions:

1. If the inaccuracy exceeds the threshold ($I > I_{thr}$), the size is increased.
2. If the fitness remains unchanged ($v \approx 0$), the predictor size is decreased, which should help the evolution to leave a local optimum.
3. If the fitness decreases ($v < 0$), the evolution is probably leaving a local optimum and decreasing the size can accelerate this process.
4. If the fitness increases ($v > 0$), the predictor size is increased to make the prediction more accurate.

The purpose of these rules is to find the lowest possible predictor size while the evolution still converges. The new *readLength* value is obtained by multiplication of the previous value and a coefficient, which is selected using described rules. Experimentally obtained values of the coefficients are specified in Sect. 4.2.

4 Results

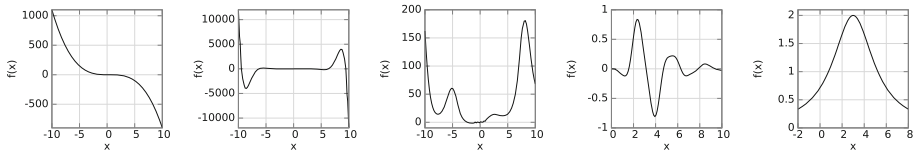
In this section, 5 symbolic regression benchmarks are introduced. Next, we present experimental results, in particular the proposed predictor behaviour and the comparisons of the proposed approach with the previously presented approaches to coevolutionary and standard CGP.

4.1 Benchmark Problems

Five symbolic regression benchmark functions (F1 – F5, see Fig. 5) were selected as training data sources for evaluation of the proposed method:

$$\begin{aligned}
&\text{F1} : f(x) = x^2 - x^3, && x = [-10 : 0.1 : 10] \\
&\text{F2} : f(x) = e^{|x|} \sin(x), && x = [-10 : 0.1 : 10] \\
&\text{F3} : f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right), && x = [-10 : 0.1 : 10] \\
&\text{F4} : f(x) = e^{-x} x^3 \sin(x) \cos(x) \left(\sin^2(x) \cos(x) - 1\right), && x = [0 : 0.05 : 10] \\
&\text{F5} : f(x) = \frac{10}{(x-3)^2 + 5}, && x = [-2 : 0.05 : 8] .
\end{aligned}$$

To form the training data, 200 equidistant distributed samples were taken from each function. Functions F1 – F5 are taken from [12] and all functions F1 – F5 were used in order to evaluate coevolution of CGP and both directly and indirectly encoded predictors [10,12].



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

Fig. 5. Symbolic regression benchmark functions used for evaluation.

4.2 Experimental Setup

The setup of the program evolution is used according to literature [12], i.e. $\lambda = 12$, $n_i = 1$, $n_o = 1$, $n_c = 32$, $n_r = 1$, $l = 32$, every node has two inputs (i_1, i_2), $\Gamma = \{i_1 + i_2,\ i_1 - i_2,\ i_1 \cdot i_2,\ \frac{i_1}{i_2},\ \sin(i_1),\ \cos(i_1),\ e^{i_1},\ \log(i_1)\}$ and the maximum number of mutations per individual is $h = 8$. The program fitness function is defined as the relative number of hits (see Eqs. 3 and 4). For the benchmarks, the user-defined acceptable errors ε are as follows: F1, F2: 0.5; F3: 1.5; F4, F5: 0.025. The acceptable number of hits is 96 %.

Table 1. Rules used to adapt the *readLength* value.

Priority	Condition	Coefficient
1	$I > I_{thr}$	1.2
2	$\|v\| \leq 0.001$	0.9
3	$v < 0$	0.96
4	$0 < v \leq 0.1$	1.07
5	$v > 0.1$	1

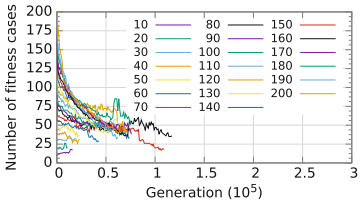
The predictor size is adapted as follows: The *readLength* value is initialized with 5 genes (the influence of the initial value is discussed in Sect. 4.3), its minimum is limited to 5 and the maximum is the total number of fitness cases. The value is updated after a new top-ranked program is found, or after 5000 generations since last update. The new *readLength* value is given as $readLength \cdot coefficient$. Experimentally obtained coefficient values are shown in Table 1. The threshold $I_{thr} = 15$ is chosen. Conditions are set according to assumptions in Sect. 3.2. If more conditions are fulfilled at the same time, the value is updated according to the priority (see Table 1).

4.3 Ability to Adapt the Number of Fitness Cases

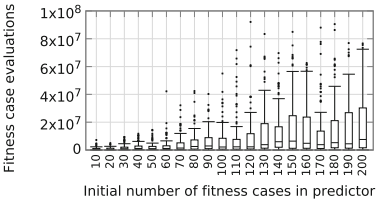
In order to confirm that the proposed algorithm is able to adapt the predictor size on a given task, we plot the progress of the average number (out of 100 independent runs) of fitness cases in top-ranked predictor during the evolution flow with respect to the initial predictor sizes. It can be seen in Fig. 6 that the size converges to the similar value independently of an initial size and the final predictor size differs for each benchmark.

The success rate is the same for each initial size setting. In the case of benchmarks F1 – F3, a larger initial size leads to more fitness case evaluations required to find an acceptable solution, see Fig. 6. This does not hold for benchmarks F4 and F5, where all settings lead to a comparable number of evaluations. The reason is that the predictor size converges in approximately 10^5 generations, while it takes much more time (approx. $3.7 \cdot 10^6$ generations) to find a satisfactory solution (see Table 2), so the effect of different predictor size in the beginning of the evolution is negligible. Note that a satisfactory solution for the benchmark F1 is found in less generations than it is necessary for the predictor size to converge.

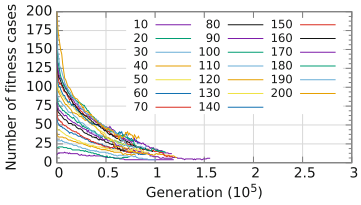
In general, it is advantageous to begin with a lower number of fitness cases in predictor, which in some cases leads to a lower number of evaluations and thus the design process acceleration. On the other hand, if the initial size is too low to find an acceptable solution, it will be automatically increased without a significant impact on the run time.



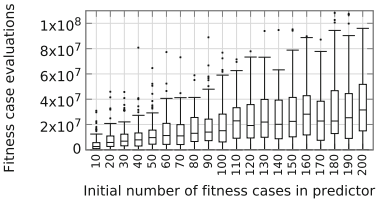
(a) Benchmark F1.



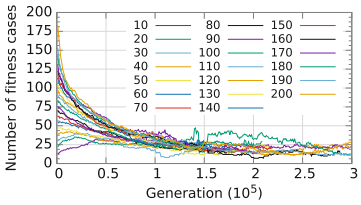
(b) Benchmark F1.



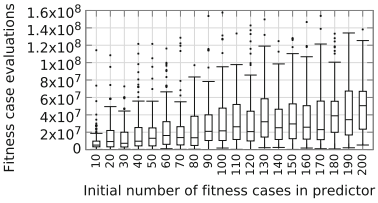
(c) Benchmark F2.



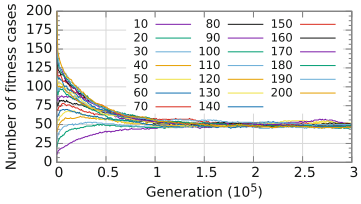
(d) Benchmark F2.



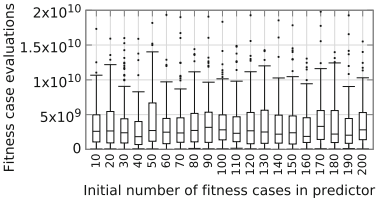
(e) Benchmark F3.



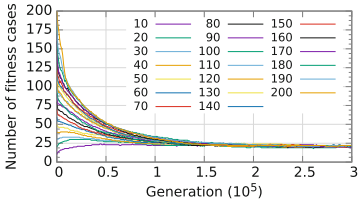
(f) Benchmark F3.



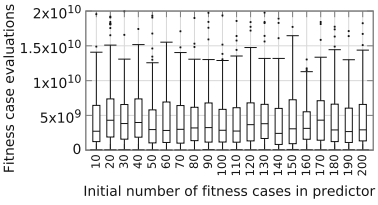
(g) Benchmark F4.



(h) Benchmark F4.



(i) Benchmark F5.

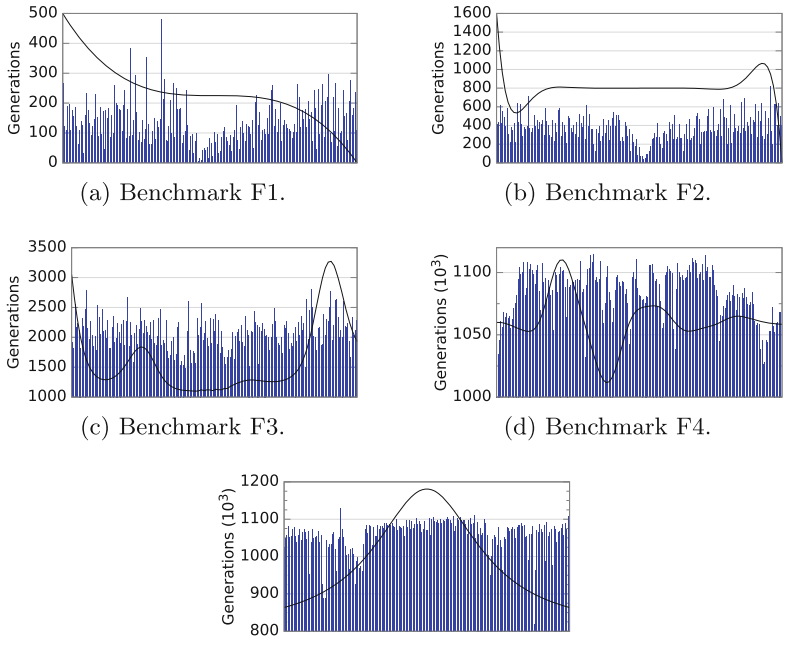


(j) Benchmark F5.

Fig. 6. Different initial predictor sizes: The average number of fitness cases in predictors and the number of fitness case evaluations necessary to find an acceptable solution.

4.4 Predictor Behaviour

In this section, we discuss how a predictor selects a subset of training data capable of guiding the evolution towards the satisfactory solution. We plot the distribution of fitness cases selected by predictors during the whole coevolutionary process out of 100 independent runs. Figure 7 show the frequency of fitness cases addressed by the top-ranked predictors during the coevolution flow. It can be seen that for benchmarks F1 and F2 predictors focus more on peaks and valleys than on flexes. On the other hand, in the case of F3 – F5, the samples are well distributed over the data set. Considering all fitness cases addressed by the predictor focused on the interesting regions (peaks and valleys) of the training data, the predictor would represent the maximum error. Note that this characteristic is desired in the Hillis' competitive coevolutionary approach [4], but is improper while requiring the predicted fitness corresponding to the exact fitness. Furthermore, fitness cases addressed by the fitness predictors are variable in response to the program evolution flow. The program evolution forces the predictors to contain two types of fitness cases, some of them are easy, others difficult, for a particular program.



(a) Benchmark F1.

(b) Benchmark F2.

(c) Benchmark F3.

(d) Benchmark F4.

(e) Benchmark F5.

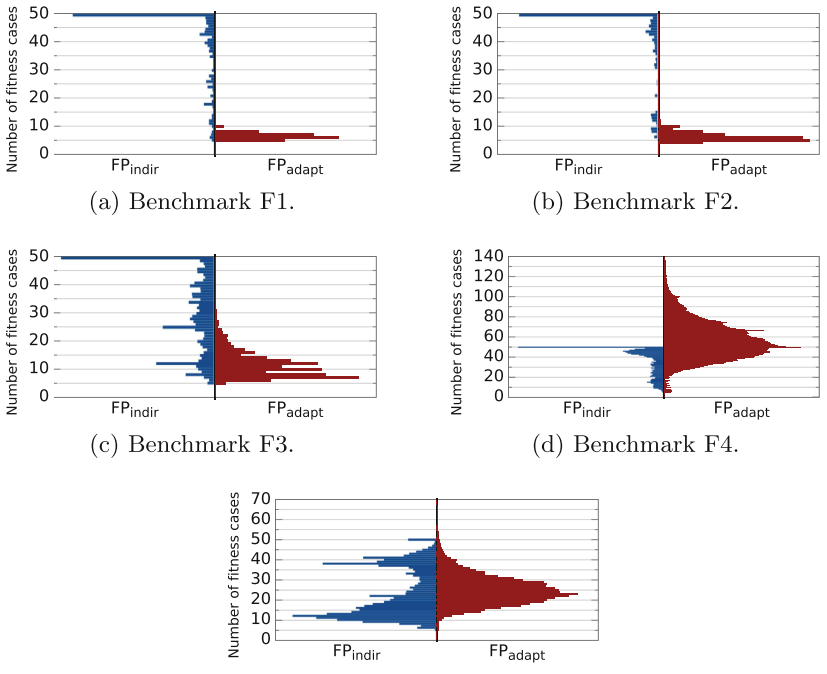
Fig. 7. Frequency of fitness cases in predictors used for programs evaluations.

4.5 Comparison of the Predictor Size

Indirectly encoded fitness predictors based on the principles of CGP (below FP_{indir}) were proposed in order to overcome the problem with selection of the most advantageous number of fitness cases used for fitness evaluation. In FP_{indir}, the predictor size parameter is included in the fitness function. Most of the sizable predictors are then rejected, but evaluated during the predictor training, which results into wasted evaluations. In order to reduce the computational cost of predictor fitness evaluations during the training, a limit of predictor size was introduced. Then, the maximum size of fitness predictor evolved using FP_{indir} was 50 fitness cases.

The size of the proposed adaptive directly encoded predictors (FP_{adapt}) varies only a little in the following generations, depending on coefficients (see Table 1). The number of fitness cases in the active predictor is thus changed only in small steps and no limit of the predictor size is necessary.

Figure 8 shows the number of fitness cases in the top-ranked predictor during the coevolution flow (left part of figures shows FP_{indir}, right part FP_{adapt}). In general, the preferred number of fitness cases differs from benchmark to benchmark. It can be seen that for the benchmarks F1 and F2 (in which only some of the first predictors are used) the preferred size of fitness predictor is the maximum value (50 fitness cases) for FP_{indir} and near to the initializing value for FP_{adapt} approach (6 – 7 fitness cases). For the benchmark F3, the maximum



(a) Benchmark F1.

(b) Benchmark F2.

(c) Benchmark F3.

(d) Benchmark F4.

(e) Benchmark F5.

Fig. 8. Number of fitness cases in predictors used for program evaluations.

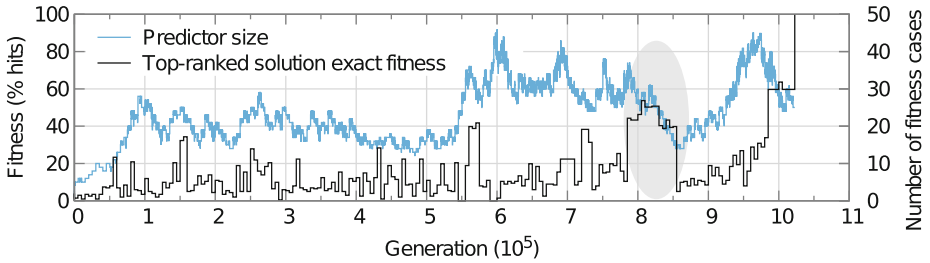


Fig. 9. Relation between the exact fitnesses of top-ranked candidate program and the size of predictor during a typical run for the F5 benchmark.

value is also preferred in FP_{indir} approach (because the evolution of predictors does not have enough time to adapt), but in FP_{adapt} the preferred value is between 7 and 12 fitness cases. Benchmark F4 is an example of how the limit of the predictor size in FP_{indir} could be restrictive. FP_{adapt} predictor size is distributed around 52 fitness cases, whereas FP_{indir} leads to predictors using 45 to 50 fitness cases and must not exceed 50. For benchmark F5, we can observe two peaks (in 12 and 38 fitness cases) for FP_{indir} predictors, but only one peak distributed around 25 fitness cases for FP_{adapt}. Note that FP_{indir} evolution allows fast changes of predictor size in contiguous generations and thus cause skips between distant values of predictor size in response to the program evolution flow. Conversely, FP_{adapt} evolution provides only small changes of predictor size in contiguous generations. The FP_{adapt} preferred predictor size, for benchmark F5, lies between preferred predictor sizes of the FP_{indir} approach, in the middle.

Although the average preferred size of predictor (out of 100 runs) in FP_{adapt} approach converges to the single value for a particular task, this trend is not so obvious while analyzing a single run. During a single run, the predictor size changes in response to the current development in the program population. Figure 9 shows the exact fitness of top-ranked program and the number of fitness cases in predictor used for program evaluation during a typical coevolutionary run for the F5 benchmark. It can be seen that the predictor size is first increased towards the preferred value and then it reacts on the development of candidate program. In this example the evolution seems to have reached a local optimum after approximately $8 \cdot 10^5$ generations which leads to decreasing of the predictor size. Around generation $8.5 \cdot 10^5$ the fitness of top-ranked program drops significantly as the evolution left the local optimum and the number of fitness cases starts to increase again, to increase accuracy of the fitness prediction.

4.6 Comparisons of Various Approaches to Fitness Prediction in CGP

The proposed coevolution employing adaptive directly encoded fitness predictors (FP_{adapt}) is compared with the original fixed-size directly encoded predictors

Table 2. Comparison of standard CGP (CGP_{STD}), coevolutionary CGP with directly encoded constant-size (FP_{const}) and adaptive predictors (FP_{adapt}) and coevolutionary CGP with indirectly encoded CGP-based predictors (FP_{indir}). For each benchmark, the best result is marked in bold font.

	Algorithm	F1	F2	F3	F4	F5
Success rate	CGP_{STD}	100 %	100 %	91 %	5 %	27 %
	FP_{const}	100 %	100 %	100 %	33 %	43 %
	FP_{adapt}	100 %	100 %	100 %	99 %	87 %
	FP_{indir}	100 %	100 %	100 %	100 %	90 %
Generations to converge (median)	CGP_{STD}	$8.66 \cdot 10^3$	$3.09 \cdot 10^4$	$1.17 \cdot 10^5$	$4.13 \cdot 10^6$	$3.25 \cdot 10^6$
	FP_{const}	$2.08 \cdot 10^3$	$1.07 \cdot 10^4$	$\mathbf{2.60 \cdot 10^4}$	$1.13 \cdot 10^7$	$7.32 \cdot 10^6$
	FP_{adapt}	$3.06 \cdot 10^3$	$1.24 \cdot 10^4$	$4.10 \cdot 10^4$	$2.42 \cdot 10^6$	$5.00 \cdot 10^6$
	FP_{indir}	$\mathbf{1.00 \cdot 10^3}$	$\mathbf{2.25 \cdot 10^3}$	$4.11 \cdot 10^4$	$\mathbf{1.47 \cdot 10^6}$	$\mathbf{1.74 \cdot 10^6}$
Fitness case evaluations to converge (median)	CGP_{STD}	$2.09 \cdot 10^7$	$7.45 \cdot 10^7$	$2.82 \cdot 10^8$	$9.96 \cdot 10^9$	$7.84 \cdot 10^9$
	FP_{const}	$\mathbf{4.41 \cdot 10^5}$	$3.09 \cdot 10^6$	$7.40 \cdot 10^6$	$2.31 \cdot 10^9$	$2.18 \cdot 10^9$
	FP_{adapt}	$6.27 \cdot 10^5$	$\mathbf{1.26 \cdot 10^6}$	$\mathbf{4.60 \cdot 10^6}$	$1.47 \cdot 10^9$	$1.53 \cdot 10^9$
	FP_{indir}	$7.43 \cdot 10^5$	$1.60 \cdot 10^6$	$1.90 \cdot 10^7$	$\mathbf{8.05 \cdot 10^8}$	$\mathbf{8.78 \cdot 10^8}$

(FP_{const}), indirectly encoded CGP-based predictors (FP_{indir}) and standard CGP without coevolution (CGP_{STD}).

FP_{const} is used according to literature [12], i.e. 12 fitness cases in chromosome, 32 individuals in predictor population, 2-tournament selection, a single-point crossover and the mutation probability 0.2. The same setup is used for FP_{adapt}, except the number of fitness cases, which is variable.

The algorithms are compared in terms of the success rate (the number of runs, giving a solution with predefined quality), the number of generations and the number of fitness case evaluations to converge (in order to compare the computational cost). Table 2 gives the median values calculated of 100 independent runs for each benchmark F1 – F5.

It can be seen in the Table 2 that both adaptive approaches, FP_{adapt} and FP_{indir}, have the highest success rate in all benchmarks. The difference is in the number of generations and fitness case evaluations required to converge. As described in Sect. 4.4, the FP_{adapt} uses fewer fitness cases than FP_{indir} for benchmarks F1 – F3. For benchmarks F1 and F2, this leads to a larger number of generations to converge using FP_{adapt} compared to FP_{indir}, fewer fitness case evaluations have to be performed using FP_{adapt}. This does not hold for the benchmark F3, where the number of generations is similar for both approaches. Nevertheless, for benchmarks F4 and F5, FP_{indir} needs fewer fitness case evaluations to converge, but still comparable in the order of magnitude. The size of fitness predictors in the FP_{indir} approach is limited to 50 fitness cases, to reduce larger predictor evaluations. However, FP_{adapt} approach prefers, for benchmark F4, more fitness cases in the predictor for this particular task (see Fig. 8), therefore the cost-reducing limit in FP_{indir} approach might be restrictive for more complex tasks.

In comparison with both FP_{adapt} and FP_{indir} approaches, FP_{const} required the lowest number of evaluations for the benchmark F1. In this case the satisfactory solution is found before the predictor size can adapt. However, let's note that many experiments have to be performed to find the most advantageous size of the predictor using FP_{const} approach for these benchmark tasks, while the FP_{adapt} and FP_{indir} adjust the size of the predictor during each single run in response to a particular task.

Finally, all three coevolutionary approaches beats CGP_{STD} in terms of number of fitness case evaluations to converge and thus accelerate the design process performed by CGP.

5 Conclusions

We have introduced the use of coevolution of cartesian programs with a new type of directly encoded predictors with the adaptive number of fitness cases. The proposed fitness predictors employ phenotypic plasticity and are able to modify the number of fitness cases used for program evaluation in dependence on the phase of program evolution.

Applied to the 5 symbolic regression tasks, we have found the proposed approach to outperform the original constant-size predictors, which use only 12 fitness cases for program evaluation, in terms of success rate and computational cost, expressed as the number of fitness case evaluations required to converge. We have shown that the proposed algorithm is able to adapt the predictor size on the solved problem in response to the development in candidate program evolution. As a result, it is possible to use coevolutionary CGP on a new task without the time-consuming experiments aimed at finding the most advantageous predictor size for the particular task.

Compared to coevolutionary CGP with indirectly encoded fitness predictors, the proposed predictor evolution does not produce predictors with larger predictor sizes than necessary. This reduces the number of necessary fitness case evaluations, while maintaining comparable program accuracy and robustness.

While symbolic regression is good to investigate the system behaviour, our future work will be devoted to applying the proposed approach to more complex problems, such as image filter design, and let the proposed approach and the approach employing indirectly encoded fitness predictors compete in the field in which the behaviour of the system is not so obvious.

The CGP has been applied to many different problem domains, predominantly in evolutionary design and optimization of logic networks. Hence the proposed approach will also be useful for evolvable hardware purposes and in real-world applications.

Acknowledgements. This work was supported by the Czech Science Foundation project 14-04197S. The authors thank the IT4Innovations Centre of Excellence for enabling these experiments.

References

1. Baldwin, J.M.: A new factor in evolution. Am. Nat. **30**(354), 441–451 (1896)
2. Ellefsen, K.O.: Balancing the costs and benefits of learning ability. In: Advances in Artificial Life, ECAL 2013, vol. 12, pp. 292–299. MIT Press (2013)
3. Ellefsen, K.O.: Evolved sensitive periods in learning. In: Advances in Artificial Life, ECAL 2013, vol. 12, pp. 409–416. MIT Press (2013)
4. Hillis, W.D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D **42**(1), 228–234 (1990)
5. Imamura, K., Foster, J.A., Krings, A.W.: The test vector problem and limitations to evolving digital circuits. In: Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware, pp. 75–79. IEEE Computer Society (2000)
6. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput. J. **9**(1), 3–12 (2005)
7. Miller, J.F.: Cartesian Genetic Programming. Springer, Berlin (2011)
8. Popovici, E., Bucci, A., Wiegand, R.P., de Jong, E.D.: Coevolutionary principles. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 988–1028. Springer, New York (2011)
9. Schmidt, M.D., Lipson, H.: Coevolution of fitness predictors. IEEE Trans. Evol. Comput. **12**(6), 736–749 (2008)
10. Sikulova, M., Hulva, J., Sekanina, L.: Indirectly encoded fitness predictors coevolved with cartesian programs. In: Machado, P., Heywood, M.I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., Sim, K. (eds.) Genetic Programming. LNCS, vol. 9025. Springer, Heidelberg (2015)
11. Sikulova, M., Sekanina, L.: Acceleration of evolutionary image filter design using coevolution in cartesian GP. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 163–172. Springer, Heidelberg (2012)
12. Šikulová, M., Sekanina, L.: Coevolution in cartesian genetic programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 182–193. Springer, Heidelberg (2012)

Paper E

Coevolutionary Cartesian Genetic Programming in FPGA

Radek Hrbacek and Michaela Sikulova

In *Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems: Advances in Artificial Life, ECAL 2013, Sicily, Italy, September 2-6, 2013*, pages 431–438. MIT Press, 2013.

Coevolutionary Cartesian Genetic Programming in FPGA

Radek Hrbáček and Michaela Šikulová

Brno University of Technology, Faculty of Information Technology,
Božetěchova 2, 612 66 Brno, Czech Republic
xhrbac01@stud.fit.vutbr.cz, isikulova@fit.vutbr.cz

Abstract

In this paper, a hardware platform for coevolutionary cartesian genetic programming is proposed. The proposed two-population coevolutionary algorithm involves the implementation of search algoritms in two MicroBlaze soft processors (one for each population) interconnected by the AXI bus in Xilinx Virtex 6 FPGA. Candidate programs are evaluated in a domain-specific virtual reconfigurable circuit incorporated into custom MicroBlaze peripheral. Experimental results in the task of evolutionary image filter design show that we can achieve significant speed-up (up to 58) in comparison with highly optimized software implementation.

Introduction

Cartesian genetic programming (CGP) – a special variant of genetic programming (GP) – has been successfully applied to a number of challenging real-world problem domains (Miller, 2011). However, the computational power that evolutionary design based on CGP (as well as on standard GP) needs for obtaining innovative results is enormous for most applications. Often, the fitness in GP is calculated over a set of *fitness cases* (Vanneschi and Poli, 2012). A fitness case corresponds to a representative situation in which the ability of a program to solve a problem can be evaluated. Fitness case consists of potential program inputs and target values expected from a perfect solution as a response for these program inputs.

A set of fitness cases is typically a small sample of the entire domain space. The choice of how many fitness cases (and which ones) to use is often crucial since whether or not an evolved solution will generalize over the entire domain depends on this choice. However, in the case of digital circuit evolution, it is necessary to verify whether a candidate n-input circuit generates correct responses for all possible fitness cases (input combinations, i.e. 2^n assignments). It was shown that testing just a subset of 2^n fitness cases does not lead to correctly working circuits (Imamura et al., 2000). Recent work has indicated that this problem can partialy be eliminated in real-world applications by applying formal verification techniques (Vasicek and Sekanina, 2011).

Hillis (1990) introduced an approach that can automatically evolve subsets of fitness cases concurrently with problem solution. Hillis used a two-population coevolutionary algorithm (CoEA) applied to a test-based problem in the task of minimal sorting network design. Subsets of test cases used to evaluate sorting networks evolved simultaneously with the sorting networks. Evolved sorting networks were used to evaluate the test cases subsets. The fitness of each sorting network was measured by its ability to correctly solve fitness cases while the fitness of the fitness cases subsets was better for those that could not be solved well by currently evolved sorting networks.

Coevolutionary algorithms are traditionally used to evolve interactive behavior which is difficult to evolve with an absolute fitness function. The state of the art of coevolutionary algorithms has recently been summarized in (Popovici et al., 2012). A *test-based problem* is defined as a co-search or co-optimization problem with two populations – population of candidate solutions and population of *tests* (subsets of the fitness cases set).

In our previous work, inspired by *coevolution of fitness predictors* (Schmidt and Lipson, 2008) and the principles of the *competitive coevolution* introduced by Hillis (1990), we proposed a two-population coevolutionary CGP algorithm running on an ordinary processor in order to accelerate the task of symbolic regression (Sikulova and Sekanina, 2012b) and the evolutionary image filter design (Sikulova and Sekanina, 2012a). For our benchmark problems (5 symbolic regression problems and salt-and-pepper noise filter design) we have shown that the (median) execution time can be reduced 2-5 times in comparison with the standard CGP.

Despite the acceleration based on fitness cases coevolution, the CGP design is still computationally very intensive design method. Therefore an FPGA based acceleration platform has been designed. Modern FPGAs provide cheap, flexible and powerfull platform, often outperforming common workstations or even clusters of workstations in particular applications. Vasicek and Sekanina (2010) introduced a new FPGA accelerator of CGP with the aim to provide both high performance and low power. The architecture

contains multiple instances of *virtual reconfigurble circuit* (VRC, Sekanina (2003)) to evaluate several candidate solutions in parallel.

Inspired by the FPGA accelerator of CGP, we propose a hardware platform for parallel two-population CoEA and show that by using this platform, the execution time of evolutionary design using CGP can be significantly reduced. The proposed hardware accelerated coevolutionary CGP is compared with hardware-accelerated standard CGP and with a highly optimized software implementation of coevolutionary CGP in the task of evolutionary image filter design.

The paper is organized as follows. The next section introduces the idea of coevolution in cartesian genetic programming. In the following section the architecture of the proposed accelerator is presented. The remaining section is devoted to experimental evaluation of the accelerator in the benchmark problem – the image filter evolution. Conclusions are given in the last section.

Coevolution in Cartesian Genetic Programming

In standard CGP (Miller, 2011), a candidate program is represented in the form of directed acyclic graph, which is modelled as an array of $n_\mathrm{c} \times n_\mathrm{r}$ (columns $\times$ rows) programmable elements (nodes). The number of primary inputs, n_i, and outputs, n_o, of the program is defined for a particular task. Each node input can be connected either to the output of a node placed in previous l columns or 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 $n_\mathrm{a} + 1$ integers where values $1 \ldots n_\mathrm{a}$ are the indexes of the input connections and the last value is the function code. Every individual is encoded using $n_\mathrm{c} \cdot n_\mathrm{r} \cdot (n_\mathrm{a} + 1) + n_\mathrm{o}$ integers.

A simple $(1+\lambda)$ 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. In each generation, an offspring with equal or better fitness than the parent's is chosen as the new parent. The offspring individuals are created using a point mutation operator which modifies up to h randomly selected genes of the chromosome, where h is a user-defined value. The algorithm is terminated when the maximum number of generations is exhausted or a sufficiently working solution is obtained.

There are two concurrently evolving populations in the proposed coevolutionary algorithm: (1) candidate programs evolving using CGP and (2) tests (*fitness cases subsets*, abb. FCSs) evolving using a simple genetic algorithm. Both populations evolve simultaneously and interact through the fitness function.

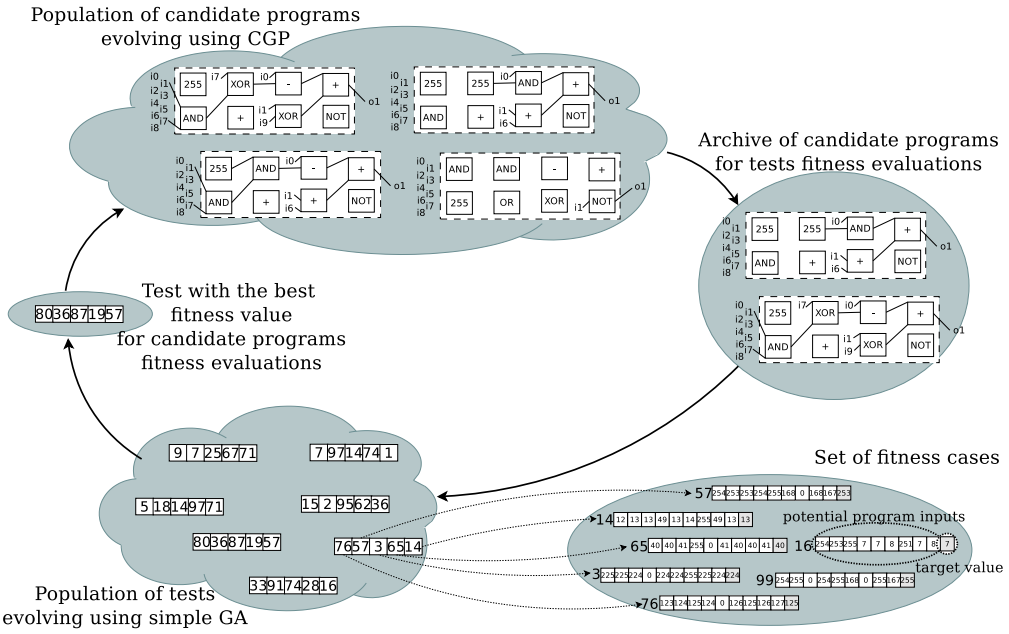


Figure 1: Populations in coevolutionary CGP – candidate programs and tests.

Test is a subset of the fitness cases set, therefore every test is encoded as a fixed-sized array of pointers to elements in the fitness cases set. In addition to one-point crossover and mutation, a randomly generated tests replacing the worst-scored tests in each generation has been used.

The aim of coevolving tests and candidate programs is to allow both candidate programs and tests to enhance each other automatically until a satisfactory problem solution has been found. Figure 1 shows the overall scheme of the proposed method. If the top-ranked candidate program fitness value (in the actual generation of candidate programs evolution) has changed against the previous generation, the top-ranked candidate program is copied to the archive of candidate programs. The archive of candidate programs is a circular list that is used for tests evaluation. Tests (in the tests evolution) are evaluated using candidate programs from the archive as follows. Each candidate program from the archive is executed for all fitness cases in the test. The test with the worst mean fitness value for candidate programs from the archive is selected as the top-ranked test in the actual generation. This test is then used to evaluate candidate programs in the candidate programs evolution. This fitness interaction approach allows to improve candidate programs using the fitness cases, which cannot be correctly solved by currently evolved candidate programs yet.

Hardware platform design

The evolutionary design includes two basic steps alternating in each generation – generation of new population and evaluation. Since the evaluation step consists in multiple running or simulating of candidate program and computing chosen fitness, a significant acceleration can be achieved by means of task or data parallelism, while the best throughput can be achieved using custom hardware.

On the contrary, the evolutionary process control is, by its nature, suitable rather for running on a universal processor, moreover in the case of CoEAs two evolutionary processes need to be executed in parallel with the ability to communicate with each other.

These requirements have been taken into account when choosing the target platform. Currently, two suitable alternatives are available – conventional FPGAs and a combination of a processor and programmable logic (e.g. Xilinx Zynq All Programmable SoC, Dobai and Sekanina (2013)). Table 1 compares several devices available in our institution as part of a development kit with respect to the configurable logic

Table 1: Target platforms comparison.

device	logic cells	block RAM
Virtex 6 XC6VLX240T	241,152	14,976 Kb
Virtex 7 XC7K325T	326,080	16,020 Kb
Zynq 7020 XC7Z020	85,000	4,480 Kb

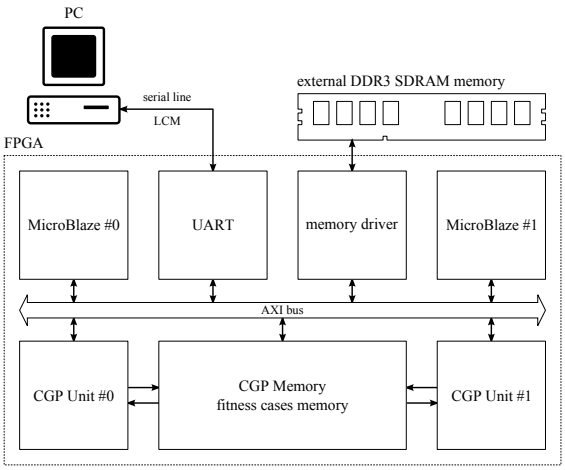


Figure 2: Hardware platform architecture.

cells count and the amount of block RAM. It is obvious, that the Zynq platform offers much less flexibility in terms of custom logic comparing to Virtex FPGA family. Therefore, a standard FPGA has been chosen as a more flexible option.

Despite the fact that standard FPGAs do not have hard processors, wide choice of soft processors under various licences are available. The most suitable choice for Xilinx devices is the MicroBlaze soft processor, offering sufficient performance while occupying a reasonable area. Figure 2 shows the proposed hardware platform architecture. The system consists of two MicroBlaze soft processors supplemented by two independent acceleration units (CGP Units) and fitness cases memory (CGP Memory). All components are interconnected by the AXI bus and additional memory channels are introduced for fitness cases transfers. Com-

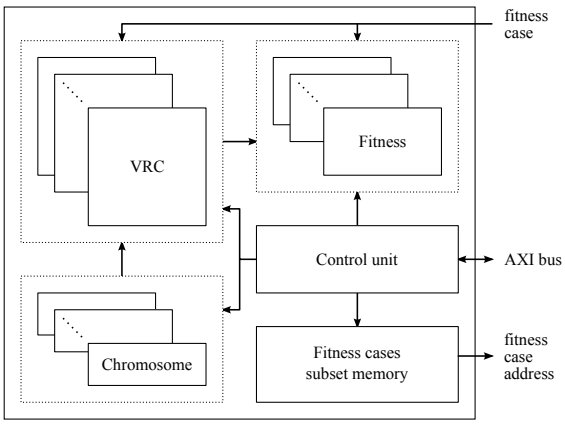


Figure 3: Detailed architecture of CGP Unit.

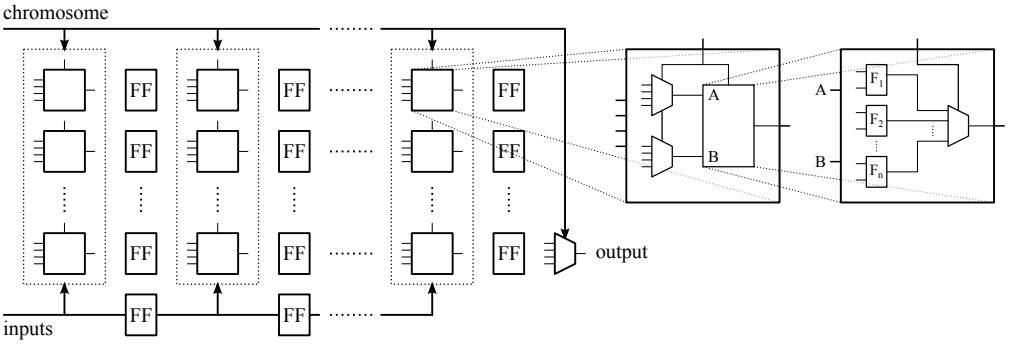


Figure 4: Virtual Reconfigurable Circuit (VRC).

munication with a service application running on a PC is performed through serial port (UART) and LCM communication library[1]. The dual MicroBlaze system utilizes AXI Mailbox component, which enables to pass simple messages between the processors (control and status messages, chromosomes, fitness values etc.).

CGP Unit (Figure 3) includes a set of subcomponents, each composed of one virtual reconfigurable circuit (VRC), fitness unit and chromosome register. Moreover, the CGP Unit includes a common control unit and FCS memory, each subcomponent is fed with the same data. The control unit is responsible for the communication between the MicroBlaze processor and the peripheral and for controlling the fitness computation. There are several configuration and status registers that are memory mapped on the AXI bus together with the test memory. By setting a specific bit in the control register, the fitness computation starts. Fitness cases are addressed indirectly using the test memory, which is addressed sequentially by the control unit. In the case of image filter design, each fitness case consists of chosen, e.g. 3×3 or 5×5, pixel neighbourhood from the noisy image and one pixel from the original image. The noisy pixels are processed in the VRCs and together with the clean pixel, properly delayed, come to the fitness unit. After a specified number of fitness cases is processed, the control unit saves the current fitness values and notifies the MicroBlaze processor by changing the status register value.

The VRC architecture is shown in Figure 4. According to the program representation in CGP, the VRC comprises a grid of nodes, called configurable function blocks (CFBs), interconnected in such a way that each block can access all other blocks in previous columns and the VRC inputs. Both VRC's inputs and CFB's outputs are registered and delayed, so that the VRC is fully pipelined while keeping the l-back parameter of arbitrary choice. Thanks to the pipelining, the VRC is able to process one fitness case per clock cycle.

[1]Lightweight Communications and Marshalling (LCM) is a set of libraries and tools for message passing and data marshalling originally designed by the MIT DARPA Urban Challenge Team.

Table 2: Functions implemented in CFBs according to Sekanina et al. (2011).

#	function	#	function
0	255	8	$i_1 \gg 1$
1	$\overline{i_1}$	9	$i_1 \gg 2$
2	$\overline{i_2}$	10	$(i_1 \ll 4) \vee (i_2 \gg 4)$
3	$i_1 \vee i_2$	11	$i_1 + i_2$
4	$\overline{i_1} \vee i_2$	12	$i_1 +^s i_2$
5	$i_1 \wedge i_2$	13	$(i_1 + i_2) \gg 1$
6	$\overline{i_1 \wedge i_2}$	14	$\max(i_1, i_2)$
7	$i_1 \oplus i_2$	15	$\min(i_1, i_2)$

Each CFB has the same structure. The input data are selected using two multiplexers and forwarded to several functions (functions used for image filter design are listed in Table 2), the output value is selected by an output multiplexer. The configuration of the multiplexers is determined by specific genes of the chromosome.

The output of each VRC is connected to separate fitness unit (Figure 7). Two different fitness functions are computed simultaneously – squared and absolute error:

$$f_{\text{sq}} = \sum_{i=1}^{N} (x_i - y_i)^2 ,$$
$$f_{\text{abs}} = \sum_{i=1}^{N} |x_i - y_i| , \tag{1}$$

where x_i is the clean pixel, y_i the VRC output corresponding to the i-th fitness case and N the number of fitness cases. These fitness functions are very similar to the MSE and MDPP functions (commonly used for image filter design), except for normalization with the number of pixels N. Since division is a very demanding operation, its removal saves a lot of resources without any impact on the application in EAs. While performing experiments, one can choose which

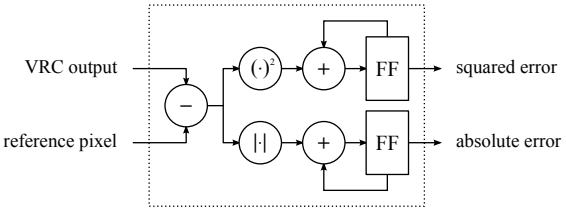


Figure 7: Fitness unit.

fitness function is used for the evaluation.

The CGP Memory component (Figure 8) is designed to achieve very high throughput. One write port (connected to the AXI bus) and two read ports enable to supply both CGP Units with data. These ports have different data widths because the AXI bus is 32 bit wide, but the width of the fitness case depends on the chosen pixel neighbourhood (80 bits for 3×3, 208 bits for 5×5). Therefore the memory has to be divided into 8 bit wide blocks and the read and write ports have to be treated in a different way. The fitness case (data output of the read port) is a concatenation of values from all these blocks from the same address. When writing from the AXI bus side, at most 4 blocks are updated at the same time. The total memory size depends on the chosen pixel neighbourhood and the maximum training image size we want to use. In our design, the number of fitness cases is limited to 65,536 due to fixed address width (16 bit), then the maximum memory capacity is $80 \cdot 65,536 \approx 5,243$ Kb for 3×3, respectively $208 \cdot 65,536 \approx 13,632$ Kb for 5×5 neighbourhood. Note that these sizes still fit into the Virtex devices, but not into the Zynq SoC (see Table 1).

Thanks to these hardware components, the fitness calculation is very efficient. The remaining steps of the evolutionary process (individuals manipulation, communication) take place on the MicroBlaze processors.

The evolutionary design is running as follows. At the beginning, original and noisy images are transfered to the external DDR3 memory, fitness cases are put together and copied to the CGP Memory. After that, the design process is initiated. Timing diagram in Figure 5 shows the steps of a single generation. The population is divided into N_{ch} chunks of P_{ch} individuals depending on the number of

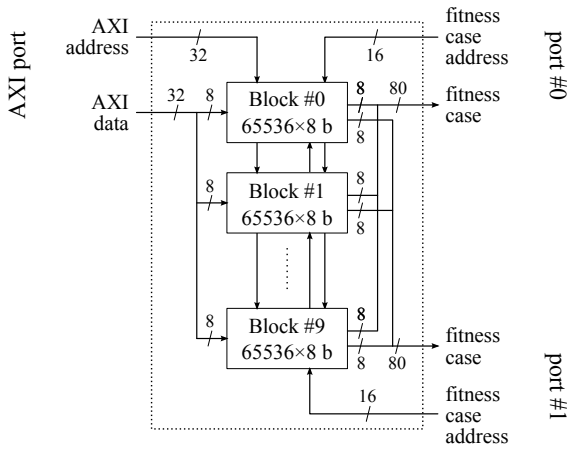


Figure 8: Architecture of CGP Memory.

VRCs N_{VRC} and the population size P:

$$N_{\mathrm{ch}} = \left\lceil \frac{P}{N_{\mathrm{VRC}}} \right\rceil, \quad P_{\mathrm{ch}} = \left\lceil \frac{P}{N_{\mathrm{ch}}} \right\rceil. \tag{2}$$

In each generation, individuals belonging to the first chunk need to be mutated and transfered to the CGP Unit before the fitness computation is executed. For every succeeding chunk (except for the last one including the last individuals of the population), the mutations and chromosome transfers can be overlapped with the fitness computation (all chromosome registers are shadowed). To achieve the best hardware utilization, the fitness computation time t_{f} has to be longer than the time t_{m} spent on the mutations and transfers. Ignoring some overhead, the total time per generation t_{g} is than:

$$t_{\mathrm{g}} = t_{\mathrm{m}} + (N_{\mathrm{ch}} - 1) \cdot \max(t_{\mathrm{m}}, t_{\mathrm{f}}) + t_{\mathrm{f}}. \tag{3}$$

Finally, when the evolution is completed, the best individual's chromosome is sent to the PC.

The coevolutionary design process is slightly more difficult, as it can be seen in Figure 6. The image filter evolution is running almost the same way except for the fitness cases subset, which is beeing evolved in parallel. For the purpose of FCSs evaluation, the best evolved filters are saved to an archive of candidate filters. The FCSs evolutionary process

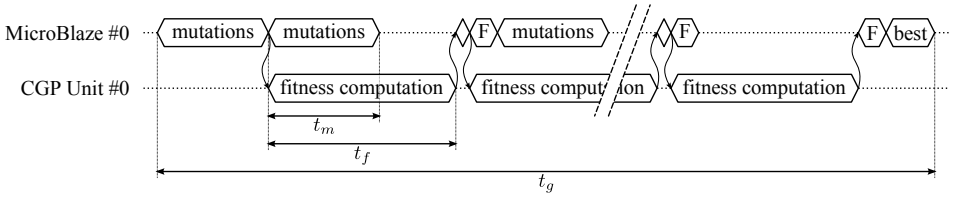


Figure 5: Timing diagram of the evolutionary process.

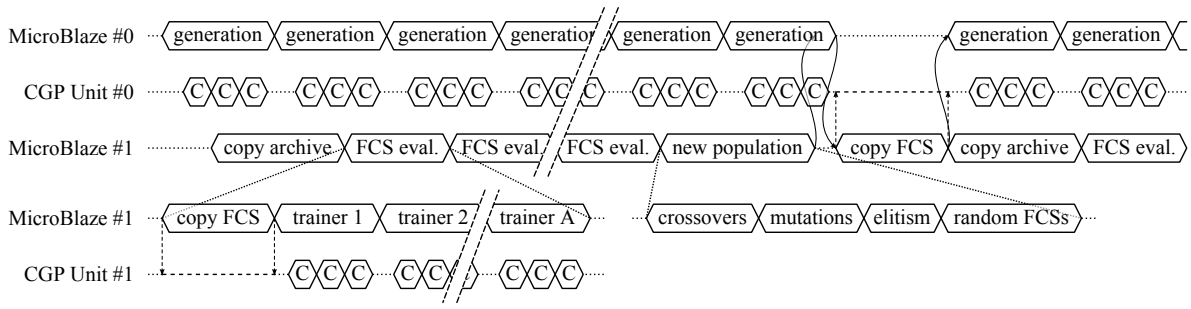


Figure 6: Timing diagram of the coevolutionary process.

is based on a simple genetic algorithm, the FCS chromosome is represented by a fixed-sized array of integers. In each generation, all FCS individuals are evaluated using all filters from the archive (trainers), the fitness value of the i-th individual is the mean value of the particular fitnesses:

$$f^i_{\mathrm{FCS}} = \frac{1}{A}\sum_{j=1}^{A} f(i,j), \tag{4}$$

where A is the archive size and $f(i,j)$ is the fitness (either squared error f_{sq} or absolute error f_{abs}) of the j-th trainer on the i-th FCS. After all FCSs are evaluated, new population is created using standard genetic operators. Specified number of individuals is obtained by one-point crossovers and the new individuals are mutated with some probability. In order to exert the selective pressure, elitism is introduced by keeping the best individual unchanged and making a few mutated clones. Finally, the rest of the population is generated randomly to preserve genetic variability. At the end of each generation, the filter evolutionary process is notified and at the right moment (after finishing the entire generation), the FCS is copied to the CGP Unit #0. No FCSs sharing between MicroBlaze processors is required.

Experimental results

This section presents benchmark problems, experimental setup and experimental evaluation of the proposed hardware accelerated approach and its comparison with the software approach.

In order to evaluate the proposed approach, salt-and-pepper noise filters were designed using standard CGP and coevolutionary CGP. This type of noise is characterized by noisy pixels with the value of either 0 or 255 (for 8-bit gray-scaled images). The Lena training image with size 256×256 pixels was corrupted by 5%, 10%, 15% and 20% salt-and-pepper noise. The evolved filters were tested on 14 different images (Gonzalez et al., 2009) containing the same type of noise.

CGP was used according to Sekanina et al. (2011), i.e. $n_{\mathrm{c}}=8$, $n_{\mathrm{r}}=4$, $l=7$, $n_{\mathrm{i}}=9$, $n_{\mathrm{o}}=1$, $\lambda=19$, every node had two inputs, the number of mutations per new individual was $h=5$ and Γ contained the functions from Table 2. The archive of candidate programs had capacity of 20 elements.

FCSs were evolved using a simple GA, where 3-tournament selection, single point crossover and mutation up to 2 % of chromosome were used. Elitism and random individuals were used to exert selective pressure and preserve genetic variability. For the GA, various chromosome lengths were tested, particularly, 1.5625 %, 3.125 %, 6.25 %, 12.5 %, 25 % and 50 % of total number of fitness cases in the training set. For each FCS size, 100 independent runs were performed and the evolution/coevolution was terminated after 100,000 generations of CGP.

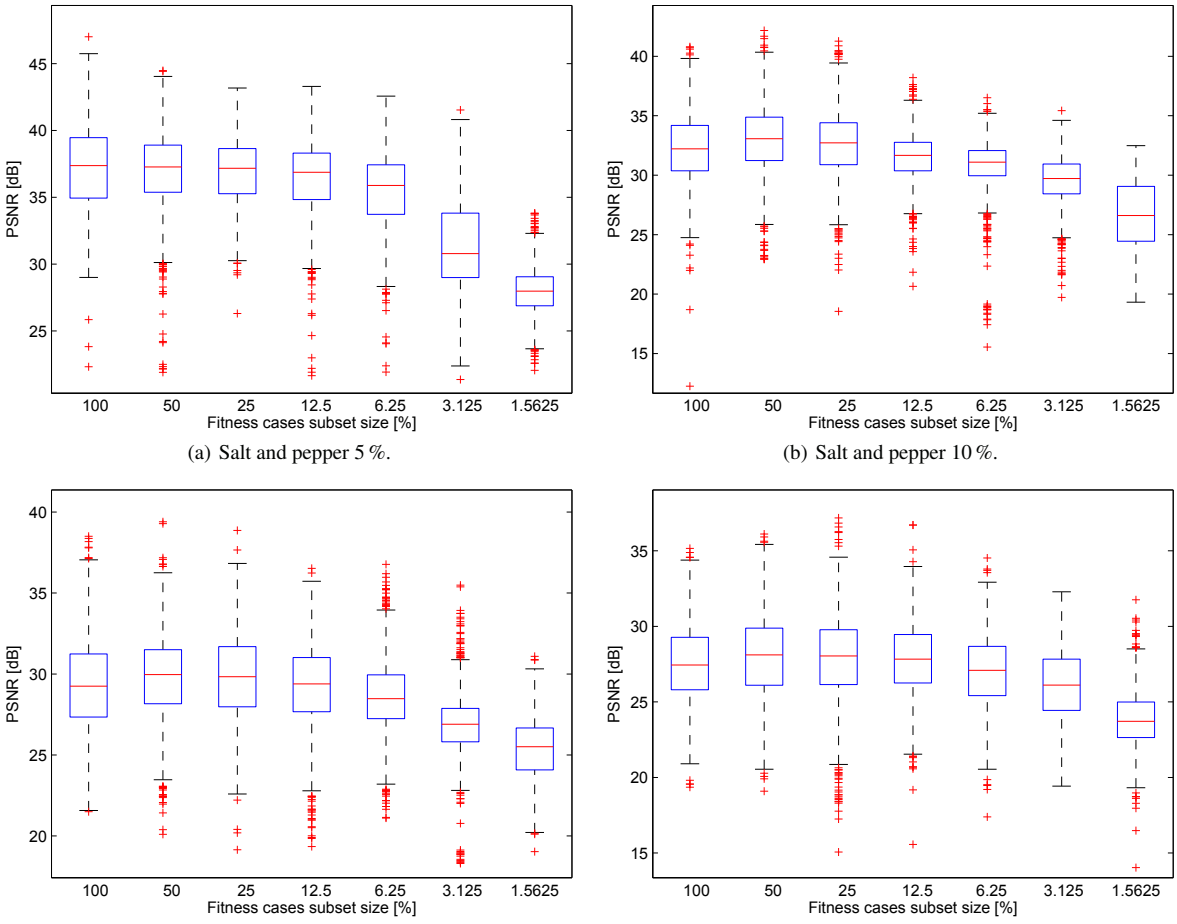
The proposed coevolutionary algorithm accelerated using FPGA was compared with the standard CGP algorithm in terms of filtering quality of evolved filters and with the higly optimized coevolution implementation running on an ordinary processor in terms of the execution time.

The quality of filtering was expressed using a measure typically used in the image processing community – as a peak signal-to-noise ratio (PSNR):

$$\mathrm{PSNR}(x,y) = 10\log_{10}\frac{255^2}{\frac{1}{MN}\sum_{i,j}(x(i,j)-y(i,j))^2}, \tag{5}$$

where $M\times N$ is the size of the image, x denotes the original image, y the filtered image and i,j are indexes of a pixel in the image. Figure 9 shows that using coevolutionary CGP running on an FPGA we are able to evolve image filters of comparable (or better) quality than standard CGP for all noise intensities. Furthermore, the higher the noise intensity, the smaller fitness cases subset can be used to get acceptable results.

Software and hardware performance comparison for standard CGP can be found in Table 3. The software imlementation is a command line tool written in `C++` utilizing OpenMP library for running in multiple threads and SSE 4.1 instruc-



(a) Salt and pepper 5 %.

(b) Salt and pepper 10 %.

(c) Salt and pepper 15 %.

(d) Salt and pepper 20 %.

Figure 9: PSNR statistics calculated from 100 evolved filters (100 independent runs using the Lena training image for each noise intensity and each FCS size) for the 14 test images.

tions enabling to process 16 fitness cases in a single step. Before evaluation, each chromosome is analyzed to exclude the inactive nodes. The performance tests were performed on the Intel Core i7-860 processor (2.8 GHz), allowing 8 threads to be running simultaneously. The hardware platform configuration was as follows: 7 VRCs in the CGP Unit #0, 6 VRCs in the CGP Unit #1, in total $N_{\mathrm{VRC}} = 13$, the entire system was running on 100 MHz frequency. Despite a very efficient software implementation and powerful processor, the hardware implementation overcomes the SW version significantly. The bigger the population, the higher the acceleration, while the most advantageous choice of the population size is a multiple of the VRC count.

The coevolutionary design performance of the hardware platform was compared to a software implementation, again optimized using OpenMP and SSE 4.1 instructions. Because of two evolutionary processes running in parallel and very poor data locality, the performance of the software implementation was vastly degraded. Therefore the speed-up

Table 3: Hardware platform evolutionary process performance (10,000 generations, image size 256×256 pixels, 1-5 mutations per chromosome) obtained by running 100 independent tests for each population size.

population size	5	10	15	20	25
SW time (s)	30.83	74.65	122.39	183.40	233.71
HW time (s)	7.21	7.86	14.17	14.44	14.83
acceleration	4.28	9.50	8.63	12.70	15.77

Table 4: Coevolutionary design performance.

FCS size	50 %	25 %	12.5 %
SW time (s)	713.23	405.47	223.18
HW time (s)	12.51	6.91	4.23
acceleration	56.99	58.64	52.82
FCS size	6.25 %	3.125 %	1.5625 %
SW time (s)	133.91	88.26	71.92
HW time (s)	3.57	4.20	3.97
acceleration	37.49	21.04	18.11

is much more significant in the case of coevolutionary design. Table 4 shows performance tests results for software and hardware approaches. The experimental setup was as follows: 10,000 generations, population of 20 individuals, image size 256×256 pixels, 1-5 mutations per CGP chromosome, up to 2 % mutations per FCS. Note that for FCS sizes lower than 12.5 %, the evolution time is similar. Due to very low fitness cases count, the fitness computation time t_{f} is shorter than the mutations time t_{m} and hardware utilization goes down. Moreover, the FCS evolution runs faster due to lower overhead and the FCS is updated more often. That is why the computation time can surprisingly grow with FCS size decrease.

Conclusions

In this paper, a hardware platform for coevolutionary CGP speed-up based on FPGA technology has been proposed. Two-population coevolutionary algorithm running on dual MicroBlaze soft processor system has been accelerated using custom peripheral based on virtual reconfigurable circuit approach. The full pipelined VRC along with a special fitness cases memory enables very efficient fitness calculation. The performance of the hardware was experimentaly evaluated in the task of evolutionary image filter design. It was shown that using custom hardware, universal processor thoughput can be greatly overcome in the task of the evolutionary design and even more in the coveolutionary case. Various sizes of fitness cases subset have been applied to demostrate the coevolutionary approach benefits. Especially for higher noise intensities, reduction of the FCS size leads to better results.

With small modifications, the hardware platform can be used to effectively evolve other digital circuits using coevolutionary CGP. In our future work, we will focus on designing image filters for other noise types as well as other image transformations, combinational logic design and other tasks suitable for coevolutionary design.

Acknowledgements

This work was supported by the Czech science foundation project P103/10/1517 and the BUT project FIT-S-11-1.

References

Dobai, R. and Sekanina, L. (2013). Towards evolvable systems based on the Xilinx Zynq platform. In *2013 IEEE International Conference on Evolvable Systems (ICES)*, pages 89–95. IEEE Computational Intelligence Society.

Gonzalez, R. C., Woods, R. E., and Eddins, S. L. (2009). "Standard" test images. *ImageProcessingPlace.com*. [online]. http://www.imageprocessingplace.com/.

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

Imamura, K., Foster, J. A., and Krings, A. W. (2000). The Test Vector Problem and Limitations to Evolving Digital Circuits. In *Proc. of the Second NASA/DoD Workshop on Evolvable Hardware*, pages 75–79. IEEE Computer Society.

Miller, J. F., editor (2011). *Cartesian Genetic Programming*. Natural Computing Series. Springer Verlag.

Popovici, E., Bucci, A., Wiegand, R., and De Jong, E. (2012). Coevolutionary principles. In *Handbook of Natural Computing*, pages 987–1033. Springer Berlin Heidelberg.

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

Sekanina, L. (2003). *Evolvable Components - From Theory to Hardware Implementations*. Natural Computing Series. Springer Verlag.

Sekanina, L., Harding, S. L., Banzhaf, W., and Kowaliw, T. (2011). Image processing and CGP. In *Cartesian Genetic Programming*, pages 181–215. Springer Verlag.

Sikulova, M. and Sekanina, L. (2012a). Acceleration of evolutionary image filter design using coevolution in cartesian GP. In *Parallel Problem Solving from Nature - PPSN XII*, LNCS 7491, pages 163–172. Springer Verlag.

Sikulova, M. and Sekanina, L. (2012b). Coevolution in cartesian genetic programming. In *Genetic Programming*, LNCS 7244, pages 182–193. Springer Verlag.

Vanneschi, L. and Poli, R. (2012). Genetic programming – introduction, applications, theory and open issues. In *Handbook of Natural Computing*, pages 709–739. Springer Berlin Heidelberg.

Vasicek, Z. and Sekanina, L. (2010). Hardware accelerator of cartesian genetic programming with multiple fitness units. *Computing and Informatics*, 29(6):1359–1371.

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