Applied Soft Computing 9 (2009) 49–60

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

An autonomous GP-based system for regression and classification problems

Mihai Oltean [*], Laura Dioşan

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Kogalniceanu 1, 400084 Cluj-Napoca, Romania

ARTICLE INFO

Article history:
Received 14 November 2006
Received in revised form 27 February 2008
Accepted 11 March 2008
Available online 20 March 2008

PACS:
01.30. $-y$

Keywords:
Genetic Programming
Adaptive strategies
Autonomous systems
Symbolic regression
Classification

ABSTRACT

The aim of this research is to develop an autonomous system for solving data analysis problems. The system, called Genetic Programming-Autonomous Solver (GP-AS) contains most of the features required by an autonomous software: it decides if it knows or not how to solve a particular problem, it can construct solutions for new problems, it can store the created solutions for later use, it can improve the existing solutions in the idle-time it can efficiently manage the computer resources for fast running speed and it can detect and handle failure cases. The generator of solutions for new problems is based on an adaptive variant of Genetic Programming. We have tested this part by solving some well-known problems in the field of symbolic regression and classification. Numerical experiments show that the GP-AS system is able to perform very well on the considered test problems being able to successfully compete with standard GP having manually set parameters.

1. Introduction

Autonomous systems [5] are of high interest due to their ability to perform various tasks without relying on human interference. A computer program is autonomous if the user does not have to change its parameters when a new problem has to be solved. For achieving this goal the following features must be implemented:

- The ability to perform well under significant uncertainties in the system and environment for extended periods of time.
- The ability to recognize if it knows how to solve a problem or not.
- The ability to create new solutions for new problems.
- The ability to learn from previous experience.
- The ability to use the computer resources in a efficient manner.
- The ability to identify failure conditions.
- The ability to improve the existing solutions during the idle-time.

The purpose of this research is to build a system meeting the previously described criteria. We have limited our attention to symbolic regression and classification problems due to several reasons:

- These problems are of great interest because they arise in many real-world applications.
- The input and output has a well-defined structure, which is easy to handle: arrays of symbols.

Our system, called Genetic Programming-Autonomous Solver (GP-AS) consists of six main parts: a Decision Maker, a Trainer, a Solver Repository, a Repository Manager, an Idle-time Manager and a Failure Manager. When a problem is presented to the system, the Decision Maker will decide which Solver will try to solve that problem by sending a request to the Repository Manager which in turn will query its database. If no suitable Solver is found, the Decision Maker will activate the Trainer, which will try to train a Solver for that problem. This new Solver will be added to the Repository for later use. The training of new Solvers is performed by using examples which are requested from the user. When no request is sent to the system it will try to improve the existing solutions by calling Idle-time Manager. Another component is able to detect possible failure of the systems and to act accordingly.

[*] Corresponding author.
E-mail addresses: moltean@cs.ubbcluj.ro (M. Oltean), lauras@cs.ubbcluj.ro (L. Dioşan).

1568-4946/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2008.03.008

The Trainer has been used for solving several interesting and difficult problems: even-parity and other 22 real-world problem taken from PROBEN1 [41].

It is difficult to compare the GP-AS system with some other problem solvers because the experimental conditions are different. Comparisons between GP and other techniques have been previously performed by other authors [10,17]. Here we have performed a raw comparison for the numerical experiments required to evolve Solvers. The results obtained by GP-AS are generally worse then those obtained by the systems that use a fixed population size and a fixed maximal tree height. There are still few cases where GP-AS Trainer performs better than standard GP. However, the comparison is not fair because experimental conditions are different. The GP-AS system uses an adaptive mechanism for population size and chromosome size and this means that it does not know which are the optimal values of these parameters for a given problem. This is different from other systems where several experimental trials have been performed in order to find this information.

The paper is organized as follows: related work is reviewed in Section 2. The extension of Genetic Programming, used for training Solvers, is described in Section 3. The way in which multiple outputs can be easily handled by GP is minutely discussed in Section 3.2. Fitness assignment in the case of regression (classification) problems is described in Sections 3.3.1 and 3.3.2. The proposed GP-AS system is presented in Section 4. The structure of the input required by the GP-AS system is thoroughly discussed in Section 4.1. Decision Maker is unveiled in Section 4.2. The Trainer and its underlying algorithm are presented in Section 4.3. The Idle-time and Failure Managers are described in Sections 4.6 and 4.7. Several numerical experiments used for solving problems are performed in Section 5 where we also discuss their results. Conclusions and future research directions are outlined in Section 6.

2. Related work

Developing automated problem solvers is one of the central themes of mathematics and computer science.

The source of inspiration in most of these approaches was the nature and the human brain. In this section, we will make a brief review of existing work in the field of general problem solvers and adaptive techniques.

2.1. Evolutionary adaptive models

The main engine of our system is based on an adaptive GP mechanism. This is why we start by reviewing some relevant work in this field.

In the early stages, evolutionary algorithms have been seen as problems solvers that exhibit the same performance over a wide range of problem without too much user interference [19,23]. The modern view say that there is no guarantee that an algorithm (having the same parameter settings) will perform similarly well on multiple problems [19,53]. This is why many techniques for adapting the parameters were proposed.

According to [3], adaptive evolutionary computations are distinguished by their dynamic manipulation of selected parameters or operators during the course of evolving a problem solution. Adaptive ECs have an advantage over standard ECs in that they are more reactive to the unanticipated particulars of the problem and, in some formulations, can dynamically acquire information about regularities in the problem and exploit them.

Mainly, there are two taxonomy schemes [3,19] which group adaptive computations into distinct classes—distinguishing by the type of adaptation (i.e., how the parameter is changed), and by the level of adaptation (i.e., where the changes occur).

In [3] the adaptive evolutionary computations have been divided into algorithms with absolute update rules (which compute a predetermined function over a set of generations or populations and use the changes in this heuristic to determine when and how to modify the algorithm's adaptive parameters) and empirical update rules (which use the same variation and selection process of evolving the problem solutions to also modify the adaptive parameters).

Both classes of adaptive evolutionary algorithms can be further subdivided based on the level the adaptive parameters operate on. Angeline distinguished between population, individual, and component-level adaptive parameters [3].

The classification scheme of Eiben et al. [19] has extended and broaden the concepts introduced by Angeline in [3]. Adaptation schemes are again classified firstly by the type of adaptation and secondly – as in [3]– by the level of adaptation. Considering the different levels of adaptation, a fourth level, environment level adaptation was introduced in order to take into account the cases where the responses of the environment are not static. Concerning the adaptation type, in [19] the algorithms are first divided into static (i.e., no changes of the parameters occur) and dynamic algorithms. Based on the mechanism of adaptation three subclasses are distinguished: deterministic, adaptive, and finally self-adaptive algorithms. The latter comprise the same class of algorithms as in [3].

The first proposals to adjust the control parameters of a computation automatically date back to the early days of evolutionary computation. In 1967, Reed et al. [43] have experimented with the evolution of probabilistic strategies playing a simplified poker game. Also in 1967, Rosenberg [45] has proposed to adapt crossover probabilities. Concerning genetic algorithms, Bagley [8] has considered incorporating the control parameters into the representation of an individual. Although Bagley's suggestion is one of the earliest proposals of applying classical self-adaptive methods, self-adaptation as usually used in ES appeared relatively late in genetic algorithms. In 1987, Schaffer and Morishima [47] introduced the self-adaptive punctuated crossover adapting the number and location of crossover points. Some years later, a first method to self-adapt the mutation operator was suggested by Back [7,6]. He has proposed a self-adaptive mutation rate in genetic algorithms similar to evolution strategies.

Some previous studies have investigated adaptive and self-adaptive representations in genetic programs. The adaptive representation genetic program [44] is a population-level adaptive genetic program that uses statistics gathered over all sub-trees occurring in the population to determine more advantageous crossover points and preserve high-fitness sub-trees. The genetic library builder (GLiB) [1,2] is an individual-level self-adaptive genetic program that co-evolves a hierarchical representation for each individual in the population.

Automatically defined functions (ADFs) [28] is an individual-level self-adaptive genetic program where each individual adapts its definitions for a predetermined set of subroutines. Koza and Andre [29] have extended this method to allow the number and interface of an individual's subroutines to adapt as well. Teller [52] has investigated a self-adaptive crossover scheme for a variant of genetic programming and Iba and de Garis [26] has described an adaptive crossover operation that