Unsupervised Problem Decomposition Using Genetic Programming

Ahmed Kattan, Alexandros Agapitos, and Riccardo Poli

School of Computer Science and Electronic Engineering
University of Essex, Colchester CO4 3SQ, UK
akatta@essex.ac.uk, aagapi@essex.ac.uk, rpoli@essex.ac.uk

Abstract. We propose a new framework based on Genetic Programming (GP) to automatically decompose problems into smaller and simpler tasks. The framework uses GP at two levels. At the top level GP evolves ways of splitting the fitness cases into subsets. At the lower level GP evolves programs that solve the fitness cases in each subset. The top level GP programs include two components. Each component receives a training case as the input. The components’ outputs act as coordinates to project training examples onto a 2-D Euclidean space. When an individual is evaluated, K-means clustering is applied to group the fitness cases of the problem. The number of clusters is decided based on the density of the projected samples. Each cluster then invokes an independent GP run to solve its member fitness cases. The fitness of the lower level GP individuals is evaluated as usual. The fitness of the high-level GP individuals is a combination of the fitness of the best evolved programs in each of the lower level GP runs. The proposed framework has been tested on several symbolic regression problems and has been seen to significantly outperforming standard GP systems.

1 Introduction

Problem decomposition aims to simplify complex real world problems in order to better cope with them. This strategy is regularly used by humans when solving problems. For example, computer programmers often organise their code into functions and classes.

Problem decomposition is important for two reasons. Firstly, it reduces the complexity of a problem and, therefore, makes the problem easier to solve by standard machine learning techniques. Secondly, automated problem decomposition may help researchers to better understand a problem domain by discovering regularities in the problem space. One way to formalise the decomposition process is to assume there exist different patterns in the problem space, each pattern has particular characteristics and therefore it needs a special solution.

Generally, problem decomposition allows a better understanding and control of the problem’s complexity. However, while it is not difficult to split a problem into several sub-problems to be solved in cooperation with different methods, using the wrong decomposition may actually increase the problems complexity.

An ideal problem decomposition system would be one that gets the data from the user and identifies different groups in the data; each of these groups should be simpler to solve than the original problem. An intelligent decomposition of problems requires understanding the problem domain and usually can only be carried out by experts. In this
paper, we propose a GP system that can evolve programs that automatically decompose a problem into a collection of simpler and smaller sub-problems while simultaneously solving the sub-problems. This is an area of GP that has not been thoroughly explored thus far.

The structure of the paper is as follows. In the next section we briefly review previous work on problem decomposition. Section 3 provides a detailed description of our proposed framework. This is followed by details on our experimental setting and results in Sections 4 and 5 respectively. Finally, conclusive remarks are given in Section 6.

2 Related Work

The solution to complex problems typically requires the construction of highly complex systems. These systems typically use hierarchical, modular structures to manage and organise their complexity. Modular structures are widespread in engineering and nature. So, it is reasonable to expect that they could be valuable in GP as well. In particular, modularity and hierarchy can be essential tools for problem decomposition. Consequently, starting from Koza’s automatically defined functions (ADFs) [1], they have been a subject of substantial empirical exploration from the early days of GP (e.g., see [2,3,4,5,6,7,8]). Due to space limitations, in this section we will review problem decomposition approaches that are based on the notion of dividing up the test cases into (possibly overlapping) subsets, since these are directly relevant to the work reported in this paper.

Rosca et al. [9] proposed a system called Evolutionary Speciation Genetic Programming (ESGP) to automatically discover natural decompositions of problems. Each individual consisted of two parts: condition and output. The condition element represents a Boolean function that receives a fitness case presented as an argument and returns feedback on whether the individual chooses to specialise in that case. The output element is a standard GP tree, which receives the chosen fitness cases as input. Naturally, some of the fitness cases may be claimed by more than one individual while others are never chosen. Thus, a fitness function was proposed which encourages individuals to fully cover the problem space and minimise the overlap of the claimed fitness cases. The approach was tested with symbolic regression problem and compared with standard GP and with GP(IF), which additionally includes if-then-else in the function set. GP(IF) is selected as it may implicitly split the problem space into different regions. Indeed, experimentation revealed that GP(IF) evolved conditions in such a way as to effectively assign different fitness cases to different pieces of code and, moreover, that GP(IF) outperformed ESGP.

Iba [10] proposed to extend GP using two well-known resampling techniques known as Bagging and Boosting and presented two systems referred to as BagGP and BoostGP. In these systems the whole population is divided into subpopulations. Each subpopulation is evolved independently using a fitness function based on a subset of the fitness cases, which are allocated by the two resampling techniques, i.e., Bagging and Boosting. Later, the best individual from each subpopulation is selected to form a voting scheme to classify unseen data. In both BagGP and BoostGP the number of subpopulations is determined by the user. Experiments on three benchmark problems showed