Probabilistic Smoothing of Genetic Programming

Ran Wei and John A. Clark

Department of Computer Science,
University of York, United Kingdom
ran.wei, john.clark@york.ac.uk

Abstract.

1 Introduction

2 Background and Motivation

3 Symbolic Regression: Example Problem

At first we discuss a case study which is an extended version of the multivalued symbolic regression problem provided as an tutorial for ECJ (Evolutionary Computation in Java)[1].

3.1 Representation

In the symbolic regression problem, there are six nodes representing the terminals nodes **X** and **Y**, and the function nodes **add** (+), **sub** (-), **mul** (*), and **div** (/).

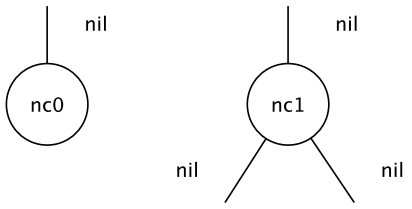

Fig. 1: Node constraints for the symbolic regression problem.

The terminal nodes conform to the node constraint *nc0*, and the function nodes conform to the node constraint *nc1* in Figure 1. *nc0* specifies that nodes

[1] https://cs.gmu.edu/~eclab/projects/ecj/docs/tutorials/tutorial4/index.html

which conform to it (Nodes **X** and **Y**) should have their return types as *nil*, which is ECJ's default type. *nc0* nodes should have 0 children. *nc1* specifies that the nodes which conform to it (Nodes **add**, **sub**, **mul**, and **div**) should have their return types as *nil*, *nc1* nodes should have two children, for which their return type should also be *nil*. The function **div** is added atop the example provided by ECJ and needs additional handling which will be discussed later.

3.2 Fitness

The objective of the symbolic regression is to evolve an equation which satisfies a set of $(X, Y, f(X, Y))$ data. In the current implementation, an upfront equation is provided: $x^2y + xy + y$. For each tree, the evaluation proceeds by giving **X** and **Y** 20 random values and calculating their expected results (denoted by ev). Then, the tree generated by GP is evaluated (of which value is denoted by v). The *fitness* of the current generation is the sum of the the difference between ev and v. The optimal would be an equation which is equivalent to $x^2y + xy + y$. The introduction of the **div** function introduces the possibility of division by 0. To handle such situations, the *fitness* of the tree is set to the maximum value if a division by 0 is detected, such that a tree which contains division by zero is discarded very early in the evolution process.

The global optimal is obtained within 20 generations of evolution by ECJ.

4 Symbolic Regression: Our Approach

With the extended symbolic regression implementation, we move on to the stochastic approach. In this approach, we introduce the concept of *stochastic node*, which is a versatile node that integrates the functions **add**, **sub**. **mul** and **div** in it. An probability is associated to each of the functions so that when a stochastic node is evaluated, the function of the node is determined (randomly) by their probabilities.

4.1 Representation

In our approach, there is only one function node: **OPNode**. There are two terminal nodes **X** and **Y**. In order to model the probability mechanism, we introduce four additional terminals: **One** (1), **Two** (2), **Three** (3) and **Four** (4). Instead of giving these nodes the *nil* type (ECJ's default type), we give these nodes the *int* type, so that they can be distinguished from the *nil* type during evolution.

To implement the stochastic node we define the node constraints shown in Figure 2. *nc0* is unchanged as it is used for terminals **X** and **Y**. *nc1* specifies that the nodes which conform to it should have no children and their return types as *int*. In our approach, nodes **One**, **Two**, **Three** and **Four** conform to *nc1*. *nc2* specifies that the nodes which conform to it should have 2 nodes of type *nil*, 10 nodes of type *int* and its own return type as *nil*. In our approach, the function nodel **OPNode** conforms to *nc2*.

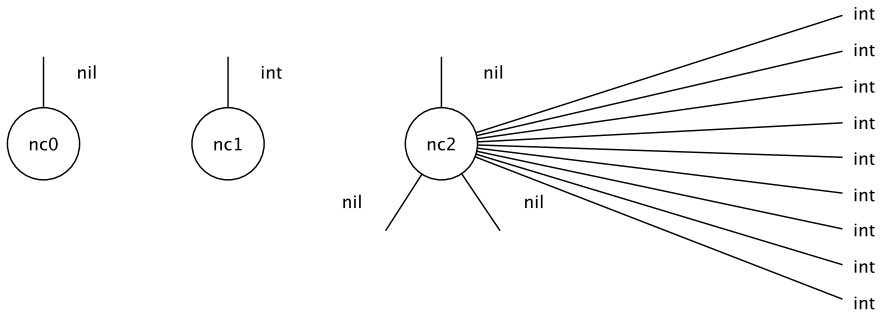

Fig. 2: Node constraints for the symbolic regression problem.

4.2 Function weightings and stochastic costs

The probability and the selection of the function of an *OPNode* is explained as follows;

- Each *OPNode* keeps track of the *weight*s of functions **add**, **sub**, **mul** and **div**, denoted by W_{add}, W_{sub}, W_{mul} and W_{div}. Initially they are all 0.
- When an *OPnode* is evaluated, the ten nodes of type *int* are evaluated first. When node **One** is encountered, W_{add} is increased by 1. W_{sub}, W_{mul} and W_{div} are increased by 1 if nodes **Two**, **Three** and **Four** are encountered respectively.
- The probability of the functions are calculated based on their weights. In particular, let P_{add}, P_{sub}, P_{mul} and P_{div} denote the probabilities of **add**, **sub**, **mul** and **div**. Such that

$$P_{add} = W_{add}/(W_{add} + W_{sub} + W_{mul} + W_{div})$$
$$P_{sub} = W_{sub}/(W_{add} + W_{sub} + W_{mul} + W_{div})$$
$$P_{mul} = W_{mul}/(W_{add} + W_{sub} + W_{mul} + W_{div})$$
$$P_{div} = W_{div}/(W_{add} + W_{sub} + W_{mul} + W_{div})$$

- The function of the *OPNode* is then selected randomly by the probabilities P_{add}, P_{sub}, P_{mul} and P_{div}.
- Each *OPNode* is also associated with a *stochastic cost*, denoted by $N_{stochastic}$, such that:

$$N_{stochastic} = 1 - max(P_{add}, P_{sub}, P_{mul}, P_{div})$$

  to calculate the stochastic cost of a tree ($T_{stochastic}$), in our approach, we use the sum of the *stochastic cost*s of all the *OPNode*s in a tree:

$$T_{stochastic} = \sum\nolimits_{N \in OPNodes} N_{stochastic}$$

4.3 Fitness

The objective of the symbolic regression is the same. In our approach we use the same experiment settings (same upfront equations and 20 randomised values for **X** and **Y** and their expected results *ev*). In addition, we add another objective: to minimise the *stochastic cost* ($T_{stochastic}$) of the tree. Let v denote the value retured by evaluating a tree, we define our secondary fitness:

$$fitness = abs(ev - v) + T_{stochastic}$$

Thus, for each pair of **X** and **Y**, the tree is evaluated 100 times and the lowest fitness cost is kept.

The global optimal is obtained within 80 generations of evolution by ECJ.

5 The Occupancy Classification Problem: Example

In this section we adapt our approach to the study of the classification problem presented in [1]. In this study, data collected from sensors in a room, such as temperature, humidity, light, and CO_2 levels (and other fields derived from the aforementioned) are used to determine if the room is occupied. The data and the occupancy are used to train the classification models. In this section, we first present the classification model we implemented using ECJ, we then adapt our approach to the classification problem and evaluate our hypothesis.

5.1 Representation

In order for our approach to apply, we first represent the problem using ECJ with only deterministic nodes.

In the occupancy classification problem, there are eight different types of data that are used in the study: *temperature*, *humidity*, *light*, CO_2, *humidity ratio*, *number of seconds from midnight* (denoted by *nsm*), week status (denoted by *ws*) and *occupancy*. In our representation, there are seven terminal nodes **T** (temperature), **H** (humidity), **L** (light), **HR** (humidity ratio), **NSM** (number of seconds from midnight) and **WS** (week status). There are also four function nodes **add** (+), **sub** (-), **mul** (*), and **div** (/).

The node constraints are the same as it is illustrated in Figure 1, where nodes **T**, **H**, **L**, **HR**, **NSM** and **WS** conform to *nc0* and nodes **add**, **sub**, **mul**, and **div** conform to *nc1*.

5.2 Fitness

The objective of the classification problem is to determine if a room is occupied with the data. In our implementation we use an threshold approach. In the

training data provided[2], there are 8143 data entries. We obtained the threshold (denoted by θ) by summing the means of all types of data:

$$\theta = \overline{T} + \overline{H} + \overline{L} + \overline{HR} + \overline{NSM} + \overline{WS}$$

When the evaluated value of a tree (denoted by v) is obtained, we compare v with the threshold θ to form our functional cost ($cost_f$):

$$v \leq \theta \Rightarrow cost_f = 1;$$
$$v > \theta \Rightarrow cost_f = 0;$$

In order to guarantee that all seven terminals are used, we check for existence of the terminals regardless of repetitions (denoted as the parameter cost $cost_{param}$). If all seven terminals are used, $cost_{param}$ is 0, otherwise $cost_{param}$ equals the number of terminals that are not used.

We then define our fitness:

$$fitness = cost_f + cost_{param}$$

References

1. Luis M. Candanedo and Veronique Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and CO_2 measurements using statistical learning models. *Energy and Buildings*, 112:28 – 39, 2016.

[2] https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+