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. Instead of using dedicated function nodes in the function set of a GP problem. We introduced the concept of stochastic node, which is a versatile node that

1 Introduction

2 Background and Motivation

3 Symbolic Regression: Example Problem

4 Symbolic Regression: Our Approach

5 The Occupancy Classification Problem: Example

In this section we adapt our approach to the study of the classification problem presented in []. In this study, data collected from sensors in a room, such as temperature, humidity, light, and CO2 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 without introducing the stochastic node.

Fig. 1: An abstract representation of the classification model

An abstract representation of our initial classification model is illustrated in Figure 1 to better understand our approach. The representation contains a number of functions (in this case study, $<$, $>$, $\leq$ and $\geq$) and a number of terminals

(*1* and *0* representing if a room is occupied or not). When the tree is evaluated, data from a single data set (the set containing the data from all the sensors at a specific point in time) is selected and used to determine if a room is occupied. In our implementation, an ideal classification model is *searched* using the data provided in the study []. The types of data used in our implementation are *temperature*, *humidity*, *light*, *co2*, *humidity ratio*, *NSM* (the number of seconds from midnight for each day) and *WS* (week status, derived from the time stamp provided, which yields 1 if weekdays and 0 otherwise).

The function nodes in our implementation has a number of attributes:

- A data type to compare. With the data type, the node is able to select the data by its type from the data set provided.
- A value to compare with the actual data of the data type. For example the root node in Figure 1 compares the light level (actual data) with a value (365). This value is likely to be determined by trial or automatically by evolution.
- Two child nodes which can be either terminals or function nodes. In our implementation, child 0 (the child on the left) always represent the *false* value, so that when the result of the evaluation is false, the evaluation goes to child 0. The sample principle applies to child 1. It is to be noted that if a terminal is reached, the evaluation of the whole tree is completed, and the evaluation result is whatever value of the terminal.

Based on the abstract representation, we implement our representation of the classification problem in ECJ.

Atomic Types and Set Types In our implementation, there are four atomic types, *nil*, *bool*, *data* and *int*. There is also a set type, *nil-or-bool*. In ECJ, this means that when an edge is declared to accept *nil-or-bool*, it accepts nodes that return either *nil* or *bool*. The usage of atomic types and the set type will be discussed later.

Node Constraints for Terminals In our implementation, three node constraints for terminals have been defined, which are shown in Figure 2. *nc0*, *nc1* and *nc2* respectively defines that for these types of terminals, their return types should be *bool*, *data* and *int* respectively.

Fig. 2: Terminals used in the classification problem

Terminals With the node constraints, we define 11 terminal nodes in the function set.

- Terminals *Result_Zero* and *Result_One* with node constraint *nc0*. These two nodes represent the boolean values 1 and 0 where 1 represents true and 0 represents false.
- Terminals *Temperature*, *Humidity*, *Light*, *CO2*, *HumidityRatio*, *NSM* and *WS* with node constraint *nc1*. These terminals are used to extract the data from the classification problem.
- Terminals *Numeral_Zero* and *Numeral_One* which represent integer values 1 and 0, which are used to denote a binary digit (bit).

Node Constraints for Non-Terminals One node constraint (named *nc4*) is defined for the non-terminals used in our implementation, which is shown in Figure 3. The constraints are:

Fig. 3: Terminals used in the classification problem

- The return type of the node is *nil*.
- The children at index 0 and index 1 (ordered specifically in this way) accepts set type *nil-or-bool*. This means that the return types of the child nodes can be either *nil* (in this instance only the nodes that conform to *nc4*) or *bool* (the nodes that conform to *nc0*).
- Child(2) to child(26) are used to compute the *value* of the node. These children can only be terminals that conform to *nc2*. The *value* as previously mentioned, is used to compare with the actual data of the node of a certain type (temperature, humidity, etc.). The value v is formed of a decimal part and a fraction part. Child(2) to child(18) are used to derive the decimal part (denoted as d), each child represents a bit in place, with a total number of 17 bits. The reason for this is because that the data type *NSM* can get up to 86399 (seconds in a day). Child(19) to child(22) (4 bits) are used to derive a number (denoted by $f1$) and then used to derive a numerator (denoted by fn) which is calculated by repeatedly dividing $f1$ by 10 until it is less than 1. Child(23) to child(26) (4 bits) are used to derive another number (denoted by $f2$), this number is then convert to a denominator (denoted by fd) where

  $$fd = 10^{f2}$$

  Thus, the value v is calculated as such:

  $$v = d + fn/fd$$

  In this sense, v covers at least the granularity of the possible numbers that may be used to compare with the actual data in this case study.
- Child[27] is used to extract the data from the classification problem. This child can only be terminals that conform to *nc1*.

Non Terminals With the node constraints, we define 4 non-terminal nodes in the function set:

- *GreaterThan*
- *GreaterThanOrEqualTo*
- *LessThan*
- *LessThanOrEqualTo*

For each node, it selects a data type and its associated data first, it then generates the *value* to be compared aforementioned, and then finally, the operator ($<$, $>$, $\leq$ and $\geq$) are used to compare these values. If the evaluation is true, the child at index 1 is evaluated. If the evaluation is false, the child at index 0 is evaluated. The evaluation of the tree completes when a terminal is reached throughout the evaluation.

Tree Constraint The tree constraint is default for ECJ where the tree return type is *nil*, this guarantees that the root of a tree can only be a non-terminal node that conforms to *nc4*