http://dx.doi.org/10.5755/j01.eee.20.5.7117 ELEKTRONIKA IR ELEKTROTECHNIKA, ISSN 1392–1215, VOL. 20, NO. 5, 2014

Incremental Approach to Structurally Difficult Problems in Genetic Programming

M. Sprogar[1], V. Podgorelec[1]
[1]*Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova St. 17, SI-2000 Maribor, Slovenia*
matej.sprogar@uni-mb.si

***Abstract*—Paper inspects the reasons for the structural difficulty of genetic representations demonstrated by a well known tunably difficult genetic programming problem. For this type of problem we believe the tree-like structure is not the major cause for problem hardness. Following this we propose a workaround, which is based on simple repetition of the evolution using pre-evolved initial populations and therefore provides closer focus points for evolution. This way the problem is solved using a manageable amount of processing, which was not possible using traditional approach. It also requires no change to the traditional genetic code base compared to other published solutions, which require substantial changes both in encoding and genetic operators. The idea is not bound to structural problems, but can be applied to other problem domains, too.**

***Index Terms*—Genetic programming, knowledge discovery, tree data structures.**

I. INTRODUCTION

Genetic programming (GP) is a systematic method for getting computers to automatically solve a problem starting from a high-level statement of what needs to be done [1]. GP is a domain-independent method that genetically breeds a population of solutions to a given problem. As such, GP is often used to solve difficult optimization problems in many engineering domains. To make the GP work, a problem should be approached in sync with GP's inherent nature.

Basically, GP is about automatic generation of executable structures. The task to identify an optimal representation of such a structure for a problem of interest is extremely hard if not impossible and is an open issue in GP [2]. This means we must understand the relationship between solution representation and search clearly in order to select the best GP representation.

The most frequently used GP structure is tree-based [1]. The decision to rely on a tree-like structure has both positive and negative consequences. On the plus side it is very easy to manipulate, on the downside, however, certain anomalies arise. For one, Daida demonstrated in [3] that the evolutionary search using tree-representation is unable to effectively search all tree shapes and, in particular, that very full or very narrow tree shapes may be extremely difficult to locate. This explains why the problem hardness can not be viewed only from the *fitness landscape* perspective.

If the mere presence or absence of particular shapes in the population affects the ability of GP to solve problems, the GP search should be conducted in a way that replaces the irrelevant structures that are in abundance with important ones that are missing. This paper proposes an approach that supplies GP with structures that are critical for its progress.

Section II is about previous work in the field with focus on the Lid problem – a tunably difficult problem which demonstrates the structural issues in tree-based GP. Next, the GP performance and causes for hardness of this problem are re-examined. Section IV is about an idea of incremental learning applied to the evolving population of trees, which transforms a hard problem into a series of easier ones. The conclusion thinks about general applicability of described findings to other problem domains.

II. PREVIOUS WORK

In at least two papers [3], [4] Daida and co-workers showed that the tree structure alone can be difficult for standard GP search. In particular, they theoretically derived and empirically proved the existence of four types of regions in the search space of tree structures. They managed to verify this by creating a tunably hard Lid problem, where the problem hardness is not tuned through changes in problem contents, but exclusively in problem structure. Search space of possible solution trees is therefore defined solely by their depth and size.

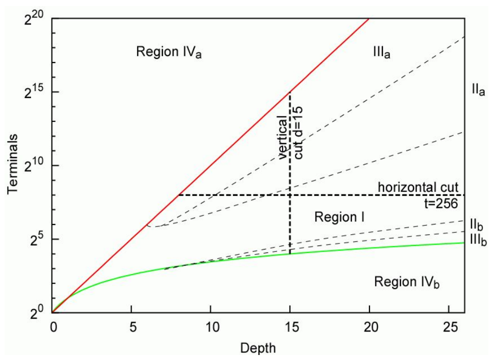

Fig. 1. Four regions and the search space of tree structures with horizontal and vertical cut. There are no trees possible in Region IV_a or IV_b.

Manuscript received September 4, 2013; accepted December 14, 2013.
This paper was produced within the framework of the operation entitled "Centre of Open innovation and ResEarch UM". The operation is co-funded by the European Regional Development Fund and conducted within the framework of the Operational Programme for Strengthening Regional Development Potentials for the period 2007–2013, development priority 1: "Competitiveness of companies and research excellence", priority axis 1.1: "Encouraging competitive potential of enterprise and research excellence".

The identified regions I-III (Fig. 1) were named according to the ability of GP to find solutions therein: Region I includes trees that are *easy* to find by GP, Region II is *transitional* space, where the GP starts having problems, and Region III is where it gets *hard* for GP to find a solution. Region IV is an *out-of-bounds* region and consists of points in space that are not attainable by means of a binary tree.

A. The Lid Problem

The objective of the Lid problem is to derive a tree of a given size and depth. In order to avoid being dependent on the content of the trees, the Lid problem has a minimal function set of single primitive function JOIN $\{J\}$, which is of arity-2, and the terminal set is the single primitive $\{X\}$.

The Lid problem requires the specification of a target tree depth (i.e., d_{target}) and a target number of tree terminals (i.e., t_{target}). The node at the root of the tree is at null (0) depth. The fitness f is defined as

$$f_{raw} = m_d + m_t, \tag{1}$$

where depth metric m_d and terminal metric m_t are defined as:

$$m_d = W_d \times \left(1 - \frac{|d_{target} - d_{actual}|}{d_{target}}\right), \tag{2}$$

$$m_t = \begin{cases} W_t \times \left(1 - \frac{|t_{target} - t_{actual}|}{t_{target}}\right), & if\ m_d = W_d, \\ 0, & otherwise, \end{cases} \tag{3}$$

and d_{actual} and t_{actual} correspond to the depth and number of terminals for the measured Lid tree, respectively. Weights W_d and W_t are chosen arbitrarily such that $W = W_d + W_t = 100$. The resulting fitness landscape is characterized by a discontinuity at d_{target}.

B. Other Work on the Lid Problem

The structural difficulty of the Lid problem was further explored by Hoai, McKay and Essam in [5], where the approach with tree-adjoining grammars (TAG) was applied. The authors speculated that structural problem difficulty occurs because of the structural step size of the structure editing operators – subtree crossover and subtree mutation are believed to be highly structurally discontinuous. This discontinuity is argued to be a consequence of the fixed arity property of standard GP representation, in that fixed arity makes it difficult to design operators with a controllable step size.

The team of [5] was better than [3] at finding solutions in Regions II and III by using complex genotype-phenotype TAG mappings and a naïve stochastic hill-climbing search. Their approach, however, is difficult to reproduce as it’s a major deviation from the standard approaches in the GP literature and available libraries (genotypes are TAG derivation trees and phenotypes are their corresponding derived trees). Also, hill climbing is well suited for the Lid’s fitness landscape once the optimal depth is found. How it would perform on other landscapes remains unclear.

C. Processing

Both [3] and [5] used the same amount of processing (100,000 evaluations of tree structures from 200 generations of 500 individuals) to arrive at their conclusions and no additional experiment was ever published (to our knowledge) that would give an estimate on how hard actually is to find the Region II/III solutions.

General statements like “*GP is effectively unable to search in this region and will not find solutions there*” [5], or “*…region that might not be accessible to GP*” leave open questions. We believe a $G = 200$ generations on a relatively small population of $M = 500$ individuals is not enough to claim extreme difficulty. After all, this is supposed to be a hard problem.

III. Issues with the Lid Problem

The results presented in this section were obtained from a by-the-book tournament based GP as described in e.g. [6], page 134. The goal was to explore what can be done to improve the GP performance on the Lid problem. If not stated otherwise, settings from [3] were used; justification for different settings is elaborated in the text. Like Daida’s, our GP was run in a single thread, also using a Mersenne Twister random number generator. Strategy parameters were set as follows: population size = 500; crossover rate = 0.9; replication rate = 0.1; population initialization with ramped half-and-half, initialization depth of 2-6 levels; maximum generations = 200. Contrary to original setup our GP used tournament-5 selection and crossover with equal bias for internal/terminal node selection, $W_d = 30$ and $W_t = 70$.

Daida described in detail two families of the Lid problem, which were used to investigate the search space of tree structures, namely “horizontal cut” and “vertical cut”. In the horizontal family the t_{target} was fixed at 256 and d_{target} was varied from 8 to 256. In the second, d_{target} was fixed at 15, while t_{target} was varied from 16 to 32,768. Both cuts in the search space are shown in Fig. 1.

Daida’s results suggest that GP has extreme difficulties finding for example even the left-most point from horizontal cut ($d = 8$, $t = 256$), which is at the beginning of Region III. The right side of the horizontal cut extends even more deeply into Region III thus it would be a complete surprise to find the extremely degenerated ($d = 255$, $t = 256$) tree. Similar observation was made in the vertical cut, where the border solution ($d = 15$, $t = 16$) is much easier than the problematic ($d = 15$, $t = 32768$).

A. Fitness and Lid GP hardness

Beside structural issues raised by Daida, experiments suggest a more “common” issue with the Lid problem hardness. Namely, the fitness function ignores the size for solutions that are not at the right depth. It starts to account for size only *after* the target depth has been reached. The problem is that for most target shapes close to the lower boundary of the search space, evolved solutions reach the target depth at a size that is more than t_{target} away from target value. Consequently the resulting metric turns negative (!) into a penalty and GP starts to select against such trees.

This is visible in Fig. 2, where the population distributions for generation 100 and 200 for target (d = 100, t = 256) are shown. We can see GP made no actual progress in 100 generations. The existing few trees of depth 100 in G_{100} and G_{200} had negative fitness compared to the majority of the population. Any-size tree of depth 99 has the constant score of *29.7*. Consequently, GP was unable to make progress and wasted 100 generations of processing.

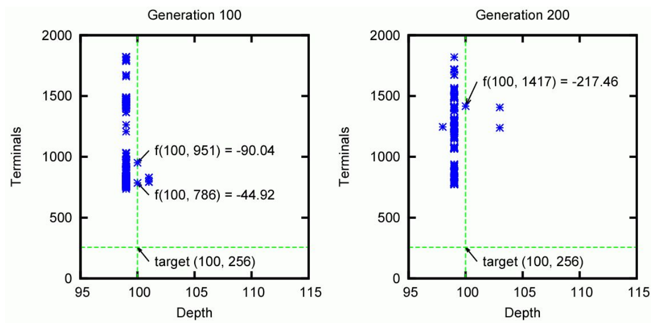

Fig. 2. Distribution of trees for the target (d = 100, t = 256) after 100 and 200 generations, respectively. The 3 marked candidate solutions of correct depth are penalized with negative fitness score. No progress is possible. Perfect score is f = 100.

Based on these observations we decided to remove the discontinuity at d_{target} from the terminals metric

$$m_t = W_t \times \left(1 - \frac{|t_{target} - t_{actual}|}{t_{target}} \right). \tag{4}$$

B. Increasing Processing Power

The performance of our GP implementation is better than that of [5] yet several points in the search space are still unavailable. The engineer's task is to find a solution thus spending more processing power is of course the first idea that is worth trying. Therefore, several pilot runs across the Region III with increased number of generations were made (M = 500, G = 10,000). No improvement in standard GP achievements was observed.

IV. Incremental Learning Approach

Fact is that crossover is able to produce any target tree shape if provided with "correct" parents (and "correct" crossover points are chosen). The Lid problem prevents GP to evolve these parents in the first place – a Catch 22 scenario. Following the building block hypothesis [7], [8] and its extension to GP [1], GP will evolve better solutions only if "appropriate" building blocks exist in adequate quantity. The supply of these is a recursive problem.

To explain by example, consider the topmost target shape from the vertical cut - a full tree (d = 15, t = 32,768). Crossover would have it easier if one of the parents were a full tree of depth 14 and another would be any tree of depth 15 with one full sub-tree. In order to have a (d = 14, t = 16,384) parent, the (d = 13, t = 8,192) parent would be "nice" to have… Lid problem actually prevents the GP to evolve these intermediate full trees in favour of others. The idea is then to provide them in the "initial" population!

A. Procedure

We propose to start with a simple problem and feed resulting population to a more advanced problem setup. This increment in difficulty can be achieved by moving the target point in search space along some pre-defined path to the desired location.

Let's denote with ${}_iP_0$ the initial population for the target point τ_i. In general, the i-th repetition of a GP run should have initial population according to (5)

$${}_iP_0 = {}_{i-1}P_{G^*}, \tag{5}$$

where G^* is a maximum number of generations needed to find a solution for a *previous* target τ_{i-1}. The incremental change of target τ is governed by a known path of points through the problem space until the desired goal is reached after n steps

$${}^2{}_i = {}^2({}^2{}_{i-1}) \xrightarrow[i \to n]{} {}^2{}_{goal}\cdot \tag{6}$$

One standard GP run is then simply replaced by a series of consecutive GP runs, each with population inherited from the previous run instead of a fresh initialization:

```
1. $i$=0, $\tau_0$=($d_0$, $t_0$)
2. create initial population ${}_0P_0$
3. do
3.1   GP_Run(G, ${}_iP_0$, $\tau_i$) $\rightarrow$ ${}_{i+1}P_{G*}$
3.2   i=i+1
4. **while** $\tau_i \neq \tau_{goal}$
```

B. Paths through the Problem Space

The most obvious thing to try is to increment the Lid difficulty along the presented horizontal or vertical cut in the search space. However, measurements showed that horizontal and vertical increments are not adequate. Rather, d and t have to be modified in a balanced way that follows the shape of the search space. This is pre-determined by the GP designer. For the Lid problem, best *path* for search among sparse trees turned out to be $t(d) = d + 1$, and for the search among full trees $t(d) = 2^d$ is recommended (Fig. 3).

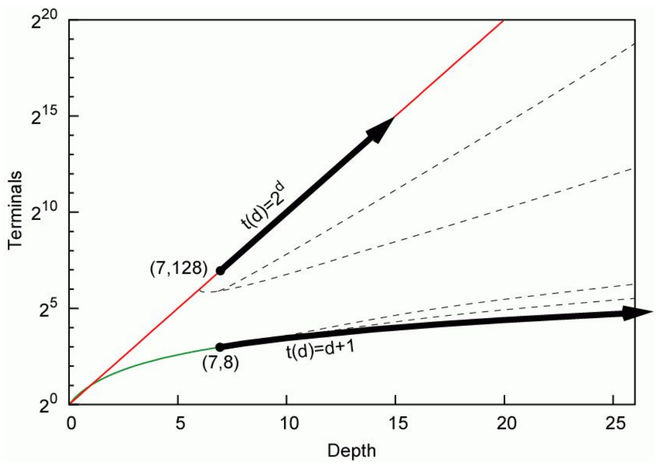

Fig. 3. Suggested starting points and incremental path through the Lid search space for solving Region II and III problems. We can sample every target point on this path (slower, safer) or proceed in greater increments (faster, riskier). Starting points (i.e. (7,128) and (7,8)) are initialization dependent (in our case ramped half-and-half, 2–6 levels).

C. Example Run

For the (*d = 15, t = 32768*) problem the GP run consists of random initialization $_0P_0$ with ramped half-and-half 2–6 and initial target $\tau_0 = (d = 7, t = 128)$. GP needs for example 19 generations to find τ_0 solution. Next, *d* is incremented by 1 and $t = 2^d$, resulting in intermediate target $\tau_1 = (d = 8, t = 256)$; $_0P_{19}$ is fed into GP as $_1P_0$ for the τ_1 target etc.

D. Results

First, we tested the proposed incremental evolution by gradually moving along both problem dimensions simultaneously. We started with a simple problem (*d = 15, t = 16*) and constantly advanced in depth by 1 (a smallest *step* increment) and adjusted the terminals $t(d) = d + 1$ accordingly, until the goal target (*d = 255, t = 256*) was reached. Regardless of the step size, the final solution was found much faster than using traditional GP. Figure 4 presents some of the evolutionary runs. While the traditional GP required 917 generations on average to find the solution, our approach required only 253 generations. The Mann-Whitney statistical test confirmed highly significant advantage of our approach with $p < 0.001$.

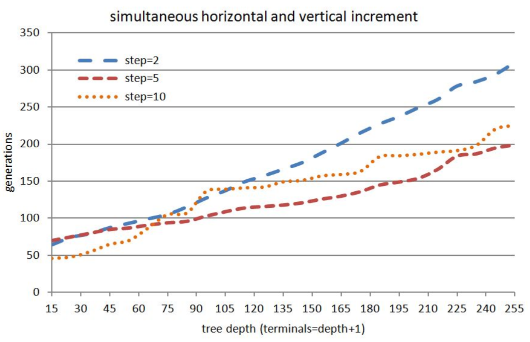

Fig. 4. The results of simultaneous incremental evolution for solving the problem (*d = 255, t = 256*).

Next we tested the incremental evolution in only horizontal direction (*t(d)* = 256). For this purpose, we chose a fairly hard problem (*d = 200, t = 256*) according to [3]. While traditional GP found the solution in 706 generations on average, our incremental approach failed to find the solution within 5,000 generations (see Fig. 5).

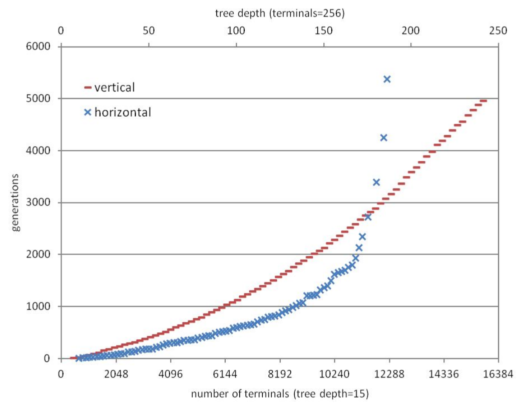

Fig. 5. The results of single horizontal (solving problem (*d = 200, t = 256*)) and vertical (solving problem (*d = 15, t = 16384*)) incremental evolution.

Finally, we tested the incremental evolution in only vertical direction. For this purpose, we chose another hard problem (*d = 15, t = 16,384*). While the traditional GP was not able to find the solution within 10,000 generations, our approach starting at (*d* = 15, *t* = 16) and incrementing number of terminals by *t* = *t* + 64 was able to find it in 5,370 generations on average. Even more, the scatter plot (see Fig. 5) revealed that the number of generations, required to find a solution, increases almost linearly with the size of the problem (number of terminals) after some size (approx. 11,000 in our experiment).

V. CONCLUSIONS

A tunably hard GP problem was approached in a fresh way, which proved to find solutions even in “hardest-to-search” regions of the search space. While the incremental change of fitness arguments d_{target} and t_{target} values, which results in gradual increase in problem difficulty, may seem Lid problem specific at first, it is not. The same approach can be applied for example in classification or regression domains, frequently addressed with the use of evolutionary techniques [9], where the fitness is typically used to evaluate a model solution on a set of training data. There, the increment can be applied to size of the dataset exposed to learning. By starting with a small subset of training data we start with a simple(r) problem and slowly proceed towards full problem by incrementally providing more and more training instances to the fitness function. Which training cases to add, however, is a scope for a new paper.

The described approach is a basis for incremental learning and is necessary when approaching hard problems in real life, too. Its benefits include simple and straight implementation using existing GP code, without much tweaking and poking of the underlying GP engine.

REFERENCES

[1] J. R. Koza, *Genetic programming: on the programming of computers by natural selection*. Cambridge, MA: MIT Press, 1992.

[2] M. O’Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf, “Open issues in genetic programming”, *Genetic Programming and Evolvable Machines*, vol. 11, no. 3–4, pp. 339–363, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10710-010-9113-2

[3] J. M. Daida, H. Li, R. Tang, A. Hilss, “What makes a problem gp-hard? Validating a hypothesis of structural causes”, in *Proc. Genetic and Evolutionary Computation* (*GECCO 2003)*, 2003, vol. 2724, pp. 1665–1677.

[4] J. M. Daida, R. R. Bertram, S. A. Stanhope, J. C. Khoo, S. A. Chaudhary, O. A. Chaudhri, J. A. I. Polito, “What makes a problem gp-hard? Analysis of a tunably difficult problem in genetic programming”, *Genetic Programming and Evolvable Machines*, vol. 2, no. 2, pp. 165–191, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1011504414730

[5] N. X. Hoai, R. I. Mckay, D. Essam, “Representation and structural difficulty in genetic programming”, *IEEE Trans. Evolutionary Computation*, vol. 10, no. 2, pp. 157–166, 2006. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2006.871252

[6] W. Banzhaf, P. Nordin, R. E. Keller, F. D. Francone, *Genetic programming – an introduction*. San Francisco, CA: Morgan Kaufmann, 1998. [Online]. Available: http://dx.doi.org/ 10.1007/BFb0055923

[7] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*. Boston, MA: Addison–Wesley, 1989.

[8] J. H. Holland, *Hidden order – how adaptation builds complexity*. Redwood City, CA: Addison–Wesley, 1995.

[9] V. Podgorelec, S. Karakatic, “A multi-population genetic algorithm for inducing balanced decision trees on telecommunications churn”, *Elektronika ir Elektrotechnika*, vol. 19, no. 6, pp. 121–124, 2013. [Online]. Available: http://dx.doi.org/10.5755/j01.eee.19.6.4578