Structural Difficulty in Estimation of Distribution Genetic Programming

Kangil Kim
Seoul National University
Structural Complexity Laboratory
Building 302, Gwanangno 599
Seoul 151744, Korea
kangil.kim.01@gmail.com

Min Hyeok Kim
Seoul National University
Structural Complexity Laboratory
Building 302, Gwanangno 599,
Seoul 151744, Korea
rniritz@gmail.com

Bob McKay
Seoul National University
Structural Complexity Laboratory
Building 302, Gwanangno 599,
Seoul 151744, Korea
rimsnucse@gmail.com

ABSTRACT

Estimation of Distribution Algorithms were introduced into Genetic Programming over 15 years ago, and have demonstrated good performance on a range of problems, but there has been little research into their limitations. We apply two such algorithms – scalar and vectorial Stochastic Grammar GP – to Daida's well-known Lid problem, to better understand their ability to learn specific structures. The scalar algorithm performs poorly, but the vectorial version shows good overall performance. We then extended Daida's problem to explore the vectorial algorithm's ability to find even more specific structures, finding that the performance fell off rapidly as the specificity of the required structure increased. Thus although this particular system has less severe structural difficulty issues than standard GP, it is by no means free of them.

Track: Genetic Programming

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; I.2.6 [**Artificial Intelligence**]: Learning—*Concept Learning*

General Terms

Experimentation

Keywords

Estimation of Distribution Algorithms, Genetic Programming, Structural Difficulty

1. INTRODUCTION

Estimation of Distribution Algorithms (EDA) [1, 7] have formed a very successful strand in evolutionary optimisation, as a result of both their efficiency in particular contexts, and of their greater theoretical transparency leading to a better understanding of their behaviour than is achievable for more typical evolutionary algorithms. The same considerations have led a number of authors to explore their use in Genetic Programming (GP) problems [11, 14] in the field often known as EDA-GP. In this paper, we examine the so-called 'structural difficulty' problem for EDA-GP.

Some years after the introduction of EDA-GP, Daida [3] showed that it was surprisingly difficult for ordinary GP to find specific tree structures when the trees were either very full or very narrow. While not a lot has been subsequently published on this issue, it has been the subject of substantial informal discussion. Initially, it was suggested that the result might simply be the result of the sparsity of full and narrow trees – that they are hard to find because they are rare. However Hoai et al's demonstration [5] that the difficulty was very much reduced in GP search using TAG trees showed that this could not be the case, since the sparsity is similar in TAG search spaces – the major difference lies instead in the connectivity of the search space, with the TAG space being much more densely connected.

If this were all there was to it, Daida's work could have spelt the death-knell of GP: he showed that there were large classes of reasonable problems where GP could not find the solutions. But there is an easy 'out'. Were Daida's difficulties of practical relevance? Typical GP problem spaces differ from the Daida spaces in having enormous numbers of redundant solutions containing introns (Daida's spaces have no introns). Even if the simplest solutions to a specific problem may be difficult to find, there will generally be other solutions of intermediate fullness that GP (because of bloat) can readily find. Thus the general consensus today is that the issue is intriguing, but of little practical relevance because of bloat (i.e. in this case, bloat, far from being a problem, comes to the rescue of GP). In many applications of GP, so the argument goes, we are equally happy to accept larger solutions so long as they are accurate. It really does not matter whether specific solutions are hard to find.

One of the key arguments for EDA-GP, repeatedly put forward by its proponents [10, 9, 12], has been its propensity to find small, compact solutions: it avoids the bloat so often generated by purely evolutionary GP systems. But in doing so, it also avoids the easy 'out' above. If EDA-GP generally finds the simplest solutions, then it had better be able to find full or narrow trees, since these might well be those simplest solutions. We explore this issue in this paper, specifically testing the ability of two well-known EDA-GP systems, Ratle and Sebag's Scalar and Vectorial Stochastic Grammar GP (SG-GP) to solve Daida's problems and some natural extensions, and comparing them with Koza-style GP.

The rest of the paper is structured as follows. In section 2 we provide additional background on Daida's Lid problem, and more detail on EDA and EDA-GP, with particular reference to SG-GP. In section 3, we detail the experimental methods we used, and detail the experimental regime and parameters. Section 4 presents the results of the experiments, including some extended experiments that further explore the structural exploration ability of SG-GP. In section 5, we discuss the implications of these experiments for SG-GP in particular, and more generally how they might relate to other forms of EDA-GP, concluding in section 6 with discussion of the assumptions and limitations of our study, a summary of the general conclusions, and directions for further work.

2. BACKGROUND

In this paper, we compare the performance of EDA-GP on Daida's problems [3] with Daida's own experiments with GP. Here, we briefly background Daida's problems and experiments. We also provide a brief introduction to EDA-GP in general, and detail the specific forms we are using.

2.1 Structural Difficulty and Genetic Programming

2.1.1 *Daida's Genetic Programming Experiments*

Table 1: Daida's GP Parameter Settings

Runs	1,000
Generations	200
Population	500
Crossover Rate	90%
Crossover Bias	90% internal
Reproduction Rate	10%
Maximum Depth	512

Daida used Koza's classic form of GP [6], in which individuals are represented as trees, and the evolutionary operators are ramped half-and-half initialisation, subtree crossover and mutation, and roulette wheel selection. His detailed GP parameter settings are shown in table 1.

2.1.2 *The Lid Problems*

Daida's Lid problems explore the space of trees built from two components, a single binary function J (join) and a single terminal X. The fitness function varies with problem, and is defined so as to pick out a specific target shape of trees in terms of depth and number of terminals. The fitness is defined as in equation 1:

$$fitness_{raw} = metric_{depth} + metric_{terminal} \quad (1)$$

where the component metrics are defined in equations 2 and 3:

$$metric_{depth} = W_{depth}(1 - (\frac{|d_{target} - d_{actual}|}{d_{target}})) \quad (2)$$

$$metric_{terminals} = \begin{cases} W_{terminals}(1 - (\frac{|t_{target} - t_{actual}|}{t_{target}})) & \text{if } metric_{depth} = W_{depth} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

and

$$W = W_{depth} + W_{terminals} = 100$$

2.1.3 *Searching Sparse Tree Shapes*

Daida's results on these problems are illustrated in figures 1 and 2.[1] Figure 1, known as the horizontal cut, shows the success rate (the proportion of individuals finding a correct solution) for different target depths while the target number of terminals is held constant at 256; Figure 2 (the vertical cut) shows the reverse – the success rate for different target numbers of terminals when the target depth is 15.

2.2 Estimation of Distribution Algorithms in Genetic Programming

2.2.1 *Estimation of Distribution Algorithms*

Estimation of Distribution Algorithms (EDAs) are model-based search algorithms, which build a statistical model of the solution space by sampling a new population from the current model, and then updating the model from good solutions. The key concepts of EDAs are derived from two different research streams: (probabilistic) graphical models (GM) and evolutionary algorithms (EA). The outer loop includes the two basic stages of a conventional evolutionary algorithm:

1. Construct a new population from selected members of the previous population

2. Select the best solutions in the new population

However it interposes a further step in the loop: after the best solutions have been selected, a probability model is built (or more usually, updated) from them. The next generation's individuals are then constructed, not directly from the best individuals of the previous generation, but rather by sampling from this (now updated) probability model. In a sense, this sampling stage may be viewed as a huge generalisation of crossover, in which information is combined, not just from two parents, but from all the previous populations of the process. The probability model, and its update mechanism, are thus crucial in determining what kinds of information are to be incorporated into this sampling, and how the information is aged. The probability model includes both probabilistic nodes (representing the probability distributions of the values of specific random variables – usually,

[1]We plotted Daida's results with SG-GP anew in figures 1 and 2. In his paper, only the points with 100% success ratio are described in detail, so we estimated the intermediate and zero ratio points as accurately as possible from his original figures. It is possible that there are small differences from his data, which is unavailable, so we refer readers to his original paper [3] for accrurate comparisons.

the value of a specific gene) and dependencies between them, often represented as a Bayesian Network (BN), a kind of GM.

The overall process may be described as in algorithm 1:

Algorithm 1 Estimation of Distribution Algorithm

```
Initialise a probability model $M_0$ from prior knowledge
{Usually, some form of uniform distribution}
$i \leftarrow 0$
while not termination condition do
    $i \leftarrow i + 1$
    Sampling: Generate new population $P_i$ by sampling from $M_{i-1}$
    Evaluation: Evaluate individuals with the problem fitness function
    Selection: select a subset $S_i$ of individuals from $P_i$
    Update: Generate the new probabilistic model $M_i$ from $M_{i-1}$ and $S_i$
end while
```

Compared with a conventional evolutionary algorithm, it has two additional processes, sampling and update. In the sampling step, EDAs construct individuals by selecting a value for each gene, based on the probability model (usually, from a random variable describing the value at that gene). Fitter individuals are selected from them, and used to update the statistical model. In the simplest case, only probability values are updated; in more sophisticated algorithms, the structure of the model may also be updated to better reflect dependencies between the genes. Their major application has been in fixed-complexity optimisation – the problem domains that are the focus of Genetic Algorithms (GA) or Evolutionary Strategies (ES). Domains requiring variable-complexity solutions (rather than variable-complexity solution models) have been much less studied. GP is one of the few exceptions, and a number of model based approaches have been developed, often under the names EDA-GP or Probabilistic Model Building GP. General surveys are available in [14]. Compared to GP, EDA-GP has received far less research attention, and many theoretical issues have not yet been studied – including its ability to learn different GP tree structures.

2.2.2 Grammar Guided GP and EDA-GP

Grammars as a formalism for GP were introduced in the mid-1990s [18, 16], and have been widely used since, in the form of Grammar-Guided GP (GGGP). In most GGGP, solutions are represented as parse trees in a specific grammar, and the parse trees are evolved in a similar way to the expression trees in classical GP. They are popular for a variety of reasons, notably the ease of specifying new problem domains by defining an appropriate grammar. Subsequently, they have formed one of the main strands in EDA-GP work as well.

Ratle and Sebag's stochastic grammar GP (SG-GP) [9] was one of the first EDA-GPs. It uses a grammar to define the target search space, but the grammar is a Stochastic Context Free Grammar (SCFG), incorporating a probabilistic weight for each production. Thus the grammar can represent a probability distribution over the search space, rather than merely (as in GGGP) delineating the boundary. In its simplest version, scalar SG-GP (vSG-GP), that is all there is to the probability model. It is a stochastic grammar representing the whole search space. However Ratle and Sebag rapidly found this limiting, and introduced an extension, vectorial SG-GP (vSG-GP), in which the probability distributions are vectorised over the depth in the GP tree. Each production has a separate copy for each depth in the tree, with a separate probability distribution, so that different probability distributions may be learnt (and used) at different depths.

We used scalar and vectorial SG-GP in this work, as simple, easy-to-understand representatives of a whole strand of EDA-GP [17, 9, 12, 13, 2, 15, 8, 4].

3. EXPERIMENTS

3.1 Fitness Functions

We used two problems for our experiments. One was the original Lid Problem described in Daida's [3], summarised in section 2. For reasons which will become apparent, we extended this with a more complex problem, not merely finding specific size/depth combinations, but amongst those, finding specific ratios of right and left branches (we call this the 'balance' of the tree). Thus we extend equation 1 with an additional term, $metric_{balance}$, as in equation 4:

$$\begin{aligned} fitness_{bal} = \quad & metric_{depth} \\ & + metric_{terminal} \\ & + metric_{balance} \end{aligned} \tag{4}$$

This new $metric_{balance}$ is defined in equation 5:

$$metric_{balance} = W_{balance}\left(1 - \left(\frac{|B_{target} - B_{actual}|}{B_{target}}\right)\right) \tag{5}$$

where as before, $W_{balance}$, W_{depth} and $W_{terminal}$ are arbitrarily selected to satisfy equation 6:

$$W = W_{depth} + W_{terminal} + W_{balance} = 100 \tag{6}$$

and B_{actual} is defined in equation 7:

$$\begin{aligned} & B_{actual} \\ & = \begin{cases} \frac{\#(\text{Unbalanced to left})}{\#(\text{Unbalanced nodes})} & \text{if } |B_{actual} - B_{target}| > \epsilon \\ B_{target} & \text{otherwise} \end{cases} \end{aligned} \tag{7}$$

where $\epsilon = \frac{0.5}{\#(\text{Unbalanced nodes})}$ and B_{target} is a number between 0 and 1. $B_{target} = 0.5$ implies that the tree has the same number of extensions to left and right, while values of 0 and 1 correspond to right and left oriented trees. Apart from its use of ϵ, $metric_{balance}$ is entirely consistent in form with Daida's metrics.

3.2 EDA-GP Test Algorithms

The grammars used for sSG-GP and vSG-GP are shown in table 2.

Both these grammars generate exactly the same search space as in Daida's work. Depth Limit is the available depth for GP trees, set to 512 in Daida's work. In the case where generating an individual within this depth fails, we re-do sampling that individual from the model distribution. In each case, the initial probability distribution over the possible right-hand-sides for each production is set as uniform.

We used a maximum depth, DepthLimit, set to 512 as in Daida's work, for model sampling. When, in generating

Table 2: Grammars for SG-GP

Scalar		
E	$\rightarrow$	(E,E) | (E,X) | (X,E) | (X,X)
Vectorial		
E_i	$\rightarrow$	$(E_{i+1}, E_{i+1})\|(E_{i+1}, X)$
		$\|(X, E_{i+1})\|(X, X)$
		$i = \{0, 1, \cdots, (\text{Depth Limit} - 2)\}$
$E_{\text{Depth Limit}-1}$	$\rightarrow$	(X, X)

an individual, we exceeded this depth bound, we re-did the sampling using the same model distribution. In the EDA-GP systems, especially in the early stages before the system had started to learn models that restrict the size of individuals, there was a tendency to generate individuals so large as to cause computer memory problems. We applied a similar approach as with depth, imposing a maximum limit of node size, and aborting the generation of individuals once they exceed this size, re-sampling from the model distribution.

3.3 Experimental Parameters Setting

Table 3: Parameter Settings for Experiments

	Set 1	Set 2	Set 3
Runs	100	30	30
Generations	200	200	200
Population	500	500;50	50
Selection ratio	40%	2%	2%
Discount ratio	50%	90%	90%

We performed three different set of experiments.

1. Using sSG-GP to solve Daida's problem
2. Using vSG-GP to solve Daida's problem
3. Using vSG-GP to solve the modified problem incorporating balance

For the first and second set, we followed a similar approach to Daida's, fixing one of the targets (horizontal cut : depth, vertical cut : terminal set size) and varying the other.

For the horizontal cut, we set the terminal size to 256 (2^8) and varied the depth from 8 to 15 with an interval of one, and from 20 to 250 with an interval of 10. For sSG-GP, we ran 100 trials, but computational cost prevented this for vSG-GP, where we were only able to run 30 trials for each treatment. However memory limits caused severe problems for vSG-GP: in the early stages of evolution, before it had learnt that relatively small trees were fitter, it tended to generate extremely large trees (note that with the initial uniform distribution over the productions, the expected size of individuals is unbounded). We handled this in two ways, neither of which is completely satisfactory for comparison with GP. Both under-estimate the actual performance of vSG-GP, so should not cause too many problems. In the first, we set a size bound of 32,784 (2^{15}, aborting the construction of individuals larger than this, and re-sampling them from the distribution as with the depth bound. Since this is far larger than the target tree size, we expected that it would have very little effect on the evolution; but memory bounds meant that we could only run this version with a population of 50, thus crippling its performance by comparison to GP. In the second, we restricted the size too a much lower bound – 2,048 (2^{11}) – which is closer to the target size, so that it might have a small effect on evolution, but also small enough that we were able to run the same population (500) for vSG-GP as for GP.

For the vertical cut, we set the depth to 15 and varied the target terminal set size from 16 to 32 with an interval of one (narrow trees), and from 64 to 16,384 (2^{14}) (full trees), multiplying by 2 to obtain the next point. For each configuration, we ran 100 trials for the first case (narrow), but could only afford 30 trials for the second case because of the computational cost for vSG-GP. As with the horizontal cut, we set a size limit of 32,784 (2^{15}), and correspondingly smaller population of 50. Since vSG-GP substantially out-performed GP in this setting, even with the reduced computational resources available, we saw no advantage in re-running with a smaller size limit and larger population.

For the third set of experiments, we used three target balance values, 1, $\frac{2}{3}$ and $\frac{1}{2}$. For each value, we tested all combinations of target depth and terminal size which were used in the preceding experiments. As before, we ran 30 trials for each configuration with a 32,784-node size limitation.

All experiments used typical EDA settings of probabilistic logic sampling, truncation selection and incremental learning for sampling, selection, and update. The selection ratio used in truncation selection, and the discount ratio used to determine the effect of old information in the model, were determined based on preliminary experiments

Detailed parameter settings are shown in Table 3.

4. RESULTS

4.1 Horizontal and Vertical Cuts for EDA-GP

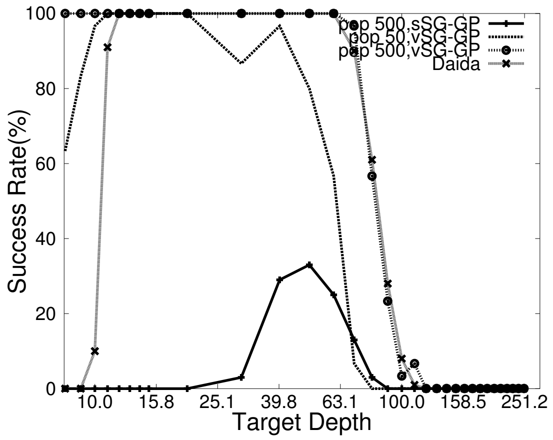

Figure 1: Success Rate, Horizontal Cut (EDA-GP)

Figure 1 shows the results for EDA-GP corresponding to Daida's horizontal cut with his results. That is, it is the result of holding the target number of terminal nodes constant, and varying the target depth. It is immediately apparent that the performance of scalar SG-GP was very poor relative to GP (a result that would surprise no-one in the field:

scalar SG-GP has no way to model the important properties of the target solution). The scalar system was unable to find any solutions at all outside the target depth range 20 . . . 100, and even within this range, search was rarely successful.

Vectorial SG-GP did substantially better than scalar, outperforming GP for small target depths even with reduced computational resources (i.e. it was slightly more effective at finding full trees), but less effective (with reduced computational resource) for greater depths. Using the same computational resources as GP, it maintained 100% success rate up to depth 60 (i.e. quite narrow trees), which is equivalent to GP performance within the limitations of the information available to us (i.e. figure 1).

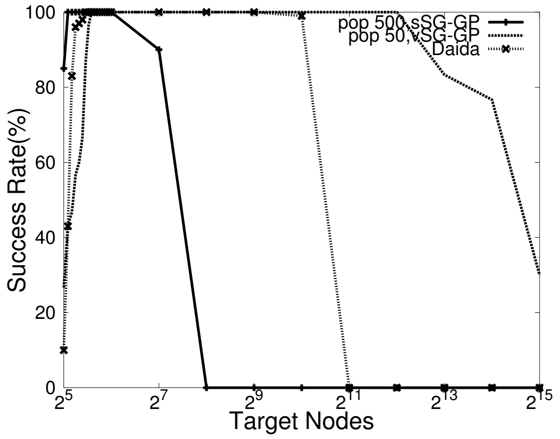

Figure 2: Success Rate, Vertical Cut (EDA-GP)

Figure 2 shows the corresponding results for the vertical cut (c.f. Daida's results in the figure). That is, it is the result of holding the target depth constant, and varying the target number of terminal nodes. In this case, there was a small surprise in the performance of scalar SG-GP: it gave very good performance for small target sizes – substantially better than GP, and even than vectorial SG-GP. However performance rapidly degraded with increasing target size, with the success rate dropping below 100% beyond target size 2^6, and never finding any solutions beyond a target size of 2^8. While vectorial SG-GP performance was slightly worse for small target sizes, it was still comparable to that of GP despite the much more modest computational resources provided to it. At larger target sizes, its effectiveness was even more pronounced; it had a 100% success rate at a target size of 2^{12} (GP was completely unsuccessful), and still maintained a respectable 25% success rate at a target size of 2^{15}, where our runs had to stop because of memory limitations.

4.2 Searching Tree Balance with vSG-GP

In the preceding experiments, vectorial SG-GP demonstrated a relatively good ability to explore different regions of the structural search space, in terms of tree fullness – it was able to find both full and narrow trees with a level of reliability at least equal to, and generally better than, GP. But fullness is not the only structural constraint GP trees might need to solve. For commutative operators such as '+' or '×', tree balance is unimportant. For non-commutative operators – for example arithmetic '−' and '/', or Boolean '→', but also including many operators in real-world problems – tree balance is important. For example, the left-heavy expression $(a * b) - c$ cannot readily be re-expressed in balanced or right-heavy form without a significant increase in complexity. Thus it is of interest to see whether vectorial SG-GP can handle different levels of balance, given a specific target depth and target node size.

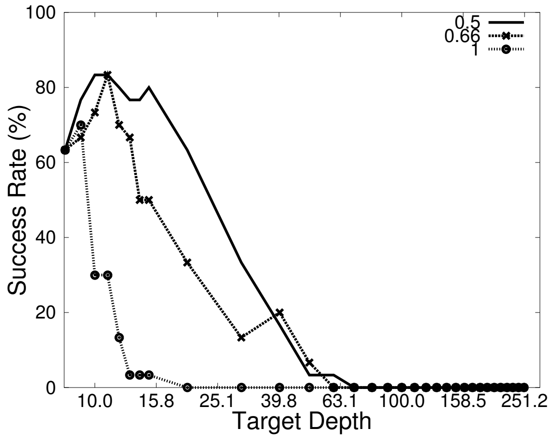

Figure 3: Success Rate, Balance-Horizontal Cut (vSG-GP)

Figure 3 shows an analogue of Daida's horizontal cut, in which both the target number of nodes and the level of balance were held constant, and the target depth was varied.

Achieving even a specific balance of 0.5 (i.e. completely balanced) made the problem much harder than just finding the right combination of number of nodes and depth, and at no setting was vSG-GP 100% successful. For depths greater than 60, vSG-GP found no solutions at all. A skew of 50% (i.e. balance setting 0.66) made the problem a little harder, while increasing the skew to 100% (i.e. a completely unbalanced tree with a setting of 1.0) made the problem very substantially harder, and no solutions at all were found for depths greater than 15 with a balance of 1.0, or greater than 50 with balance of 0.66.

Correspondingly, figure 4 shows an analogue of Daida's vertical cut, in which both the target depth and the level of balance were held constant, and the target number of nodes was varied. Achieving a specific balance of 0.5 was fairly reliable around 25 terminal nodes, and the success rate remained around 80% from a target of 25 right up to 512, declining fairly rapidly thereafter and finding no solutions at all for targets greater than 16,383 (8,192 terminals). A balance of 0.66 was harder to achieve, succeeding over 60% of the time up to 255 nodes, but decreasing as with the 0.5 balance thereafter, and failing completely after 16,383 nodes. There was an almost compete failure to find completely unbalanced trees, with the success rate remaining below 5% for all target sizes, and no success at all larger than 8,191 nodes.

4.3 How does vSG-GP solve the Balance Problem?

One of the important characteristics of EDA-GP is that once we solve a problem, we have useful information avail-

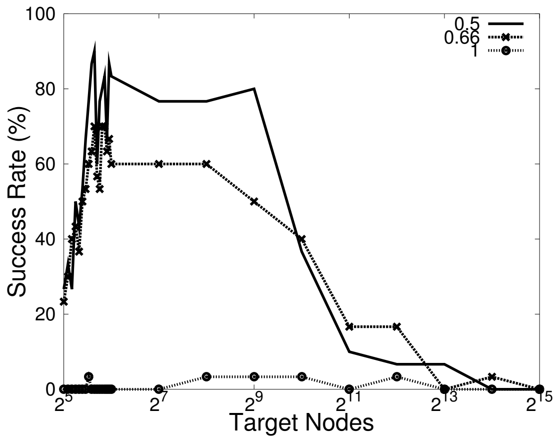

Figure 4: Success Rate, Balance-Vertical Cut (vSG-GP)

able to understand the solution: not merely the successful solutions, but the probability model that generated them (for this reason, it is often useful to continue an EDA-GP beyond its first hitting time, to obtain a probability model that generates solutions with high reliability)

Tables 4, 5 and 6 show typical examples of the final model distributions of successful runs, with respective balance targets of 1.0, 0.5 and 0.66. They are collected from the depth 10, 256-terminal target setting. By the definition of vSG-GP, all models have a 512-component probability model, storing the distributions of productions. In all cases, vSG-GP has learnt the depth limit of 10 perfectly, selecting the right hand side (X,X) with probablity 1.0 (as is required to terminate the tree at that level).[2] We discuss the components slightly out-of-order for ease of explanation (simplest first).

Table 4: Successful Model Distribution, Target Balance 1.0)

depth	Probability for each Production			
	(E,E)	(E,X)	(X,E)	(X,X)
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0
4	1	0	0	0
5	1	0	0	0
6	1	0	0	0
7	0.853	0.147	0	0
8	0.206	0.423	0	0.371
9	0.825	0	0.004	0.171
10	0	0	0	1
11-512	0.25	0.25	0.25	0.25

In table 4 (corresponding to balance 1.0), at depths up to 6, the distribution has converged to a probability of 1.0 for (E,E), ensuring that any sampled tree is full (and hence balanced) up to that depth. The distribution at lower depth converges to (E,E), which does nott affect the balance. From level 7 to 9, there are a variety of distributions, but levels 7 and 8 are skewed to the left by 0.147 and 0.423 (and no right skew), while level 9 is skewed to the right by only 0.004. Thus the probability of generating trees with only left productions (E,X) and no right productions (X,E) is quite high, leading to samples containing perfect solutions with a balance of 1.0.

[2]We do not show deeper components here because they are uninteresting. They start with a uniform distribution; this may be perturbed in the first few generations, but once the system learns the required depth, they are no longer sampled, so exponentially rapidly restore the uniform distribution.

Table 5: Successful Model Distribution, Target Balance 0.5

depth	Probability for each Production			
	(E,E)	(E,X)	(X,E)	(X,X)
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0
4	1	0	0	0
5	1	0	0	0
6	1	0	0	0
7	0.164	0.537	0.300	0
8	0.078	0	0.372	0.551
9	0	0.346	0.016	0.638
10	0	0	0	1
11-512	0.25	0.25	0.25	0.25

Table 5 (balance 0.5) shows an identical distribution to the preceding for depths up to 6. The distributions at depths 7, 8 and 9 are very different, though. It might appear that the distribution is somewhat unbalanced, since depths 7 and 9 are more skewed to the left than depth 8 is to the right. However we can calculate the expected distribution exactly. The expected number of internal nodes at depth i, denoted by N_i, is given by equation 8:

$$N_i = N_{i-1} * (2 * P_i(E,E) + P_i(E,X) + P_i(X,E)) \qquad (8)$$

Since the probability of generating (E,E) from depth 5 nodes, shown at depth 6, is 1.0, repeatedly applying equation 8, we get the expected number of nodes using (E,X) as equation 9:

$$N_6 \times 0.537 + N_7 \times 0 + N_8 \times 0.346 = 0.750 \qquad (9)$$

The corresponding calculation for (X,E) gives an expectation of 0.743, so that the balance calculated from the expectations is

$$\frac{0.750}{0.750 + 0.743} = 0.502$$

which is quite close to the target balance of 0.5, so that the probability of generating perfect solutions is high.

From table 6, by the same process we can compute an expected balance value of:

$$\frac{17.596}{17.596 + 8.621} = 0.671$$

which again is similar to the $\frac{2}{3}$ target balance.

From these examples, we can see that the vSG-GP model distribution can converge to a distribution that leads to a high probability of success in sampling perfect solutions.

Table 6: Successful Model Distribution, Target Balance 0.66

depth	Probability for each Production			
	(E,E)	(E,X)	(X,E)	(X,X)
1	1	0	0	0
2	1	0	0	0
3	0	0.675	0.325	0
4	1	0	0	0
5	1	0	0	0
6	1	0	0	0
7	0.820	0.002	0.177	0
8	0.282	0.194	0.376	0.148
9	0.060	0.850	0.084	0.006
10	0	0	0	1
11-512	0.25	0.25	0.25	0.25

5. DISCUSSION

The first point to note about these results, as in all EDA work, is the huge dependence on the underlying probability model. Scalar SG-GP is almost entirely unable to handle the Lid problem, except in some very narrowly defined cases, whereas vectorial SG-GP – differing only in the probability model – overall performs rather better than standard GP. With its simple probability model, sSG-GP is unable to even represent the solution space, let alone learn it.

However the probability model does not have to be able to explicitly represent the problem structure. While vectorial SG-GP has an explicit representation of the depth aspect of the Lid problem, it only represents the fullness (and balance) aspects of solutions implicitly through the differing probabilities in the grammar.

Specifically with respect to vectorial SG-GP, while it does perform quite well on the basic Lid problem, showing less bias against full or narrow trees than does GP, this does not mean that it is without shape bias. It is much less effective at finding specific levels of imbalance. This may possibly be because of a high level of epistasis in this problem with vSG-GP representation, but this hypothesis requires further verification. It is quite possible that other probability models would reduce such epistasis, and thus perform better on this problem. Especially, this may be the case for systems such as GMPE [13] which learn the structure of the probability model, rather than merely the probabilities.

6. CONCLUSIONS

6.1 Summary

In this paper, we argued that whatever may be the case for GP, EDA-GP systems need to be able to find arbitrary structural forms. If they cannot, their strong parsimony bias will lead them to miss solutions to some problems. We showed that their ability to do this depended heavily on the probability model: a simple probability model such as in scalar SG-GP gave very poor performance on Daida's Lid problem. On the other hand, a slightly more sophisticated model, such as in vectorial SG-GP – permitting direct representation of the depth-dependence of solutions – performed somewhat better than GP, though by no means perfectly.

While it might thus seem that shape biases are a minor issue for vectorial SG-GP, we introduced an extension of the Lid problem incorporating the balance of the tree shape being sought. We found that performance of vectorial SG-GP on this problem was much reduced from that on the basic Lid problem, and thus that vectorial SG-GP still suffers from serious problems in finding specific tree shapes. We hypothesised that this poorer performance may result from epistasis in this problem.

6.2 Future Directions

The most promising future direction in this work is to explore in more detail the relationship between the probability model in EDA-GP and its performance in learning specific tree shapes. We note that the probability model depends not only on the learning system, but also on the specific implementation. For example, the grammar in table 2 is one way to represent the Lid problem in vectorial SG-GP. But it is far from the only one. To what extent would changes in the grammar (without any other change in the learning system, or the problem space) affect performance in learning specific structures? Equally interesting is the possibility of extension to EDA-GPs that learn the structure of the probability model. Would these algorithms be able to learn the structure of models that could represent structural constraints, while at the same time learning the probabilities themselves?

7. ACKNOWLEDGEMENTS

Seoul National University Institute for Computer Technology provided research facilities for this study, which was supported by the Basic Science Research Program of the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Project No. 2010-0012546), Research Expenses for Foreign Professors funded by Seoul National University(Project No. 400-20100189), and the BK21-IT program of MEST.

8. REFERENCES

[1] S. Baluja. Population-based incremental learning: A method for integrating genetic searching based function optimization. Technical Report CMU-CS-94-163, Computer Science Dept, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[2] P. A. N. Bosman and E. D. de Jong. Learning probabilistic tree grammars for genetic programming. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. T. A. Kabán, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 192–201, Birmingham, UK, 18-22 Sept. 2004. Springer-Verlag.

[3] J. M. Daida, H. Li, R. Tang, and A. M. Hilss. What makes a problem GP-hard? validating a hypothesis of structural causes. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1665–1677, Chicago, 12-16 July 2003. Springer-Verlag.

[4] Y. Hasegawa and H. Iba. Latent variable model for estimation of distribution algorithm based on a

probabilistic context-free grammar. *IEEE Transactions on Evolutionary Computation*, 13(4):858–878, Aug. 2009.

[5] N. X. Hoai, R. I. B. McKay, and D. Essam. Representation and structural difficulty in genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):157–166, Apr. 2006.

[6] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[7] P. Larrañaga and J. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Netherlands, 2002.

[8] M. Looks. Scalable estimation-of-distribution program evolution. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 1, pages 539–546, London, 7-11 July 2007. ACM Press.

[9] A. Ratle and M. Sebag. Avoiding the bloat with probabilistic grammar-guided genetic programming. In P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, and M. Schoenauer, editors, *Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001*, volume 2310 of *LNCS*, pages 255–266, Creusot, France, Oct. 29-31 2001. Springer Verlag.

[10] R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.

[11] R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution: Stochastic search through program space. In M. van Someren and G. Widmer, editors, *Machine Learning: ECML-97*, volume 1224 of *Lecture Notes in Artificial Intelligence*, pages 213–220. Springer-Verlag, 1997.

[12] Y. Shan, R. I. McKay, H. A. Abbass, and D. Essam. Program evolution with explicit learning: a new framework for program automatic synthesis. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 1639–1646, Canberra, 8-12 Dec. 2003. IEEE Press.

[13] Y. Shan, R. I. McKay, R. Baxter, H. Abbass, D. Essam, and N. X. Hoai. Grammar model-based program evolution. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 478–485, Portland, Oregon, 20-23 June 2004. IEEE Press.

[14] Y. Shan, R. I. McKay, D. Essam, and H. A. Abbass. A survey of probabilistic model building genetic programming. In M. Pelikan, K. Sastry, and E. Cantu-Paz, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, volume 33 of *Studies in Computational Intelligence*, chapter 6, pages 121–160. Springer, 2006.

[15] I. Tanev. Implications of incorporating learning probabilistic context-sensitive grammar in genetic programming on evolvability of adaptive locomotion gaits of snakebot. In R. Poli, S. Cagnoni, M. Keijzer, E. Costa, F. Pereira, G. Raidl, S. C. Upton, D. Goldberg, H. Lipson, E. de Jong, J. Koza, H. Suzuki, H. Sawai, I. Parmee, M. Pelikan, K. Sastry, D. Thierens, W. Stolzmann, P. L. Lanzi, S. W. Wilson, M. O'Neill, C. Ryan, T. Yu, J. F. Miller, I. Garibay, G. Holifield, A. S. Wu, T. Riopka, M. M. Meysenburg, A. W. Wright, N. Richter, J. H. Moore, M. D. Ritchie, L. Davis, R. Roy, and M. Jakiela, editors, *GECCO 2004 Workshop Proceedings*, Seattle, Washington, USA, 26-30 June 2004.

[16] P. A. Whigham. Grammatically-based genetic programming. In J. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, 1995.

[17] P. A. Whigham. Inductive bias and genetic programming. In A. M. S. Zalzala, editor, *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, volume 414, pages 461–466, Sheffield, UK, 12-14 Sept. 1995. IEE.

[18] M. L. Wong and K. S. Leung. An induction system that learns programs in different programming languages using genetic programming and logic grammars. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, 1995.